MÓDULO X

Introdução ao CSS3 e Estilização Básica

A Educação é o primeiro passo para um futuro melhor...

MÓDULO X

Display Inline, Block e Inline-Block

O que é o Display?

Para deixar a explicação mais simples e feliz, vamos pensar que temos 3 tags **<li>** dentro de uma **<ul>**. Todas **<li>** vão ter 150px de altura (**height: 150px**) e a largura vai ser de 33.333% (**width: 33.333%)**. Ah! A primeira **<li>** será da cor laranja (**background-color: #FAA116**), a segunda da cor verde (**background-color: #57C7C3**) e a terceira, e última, da cor azul (**background-color: #3B97D3**).

CodePen

MÓDULO X

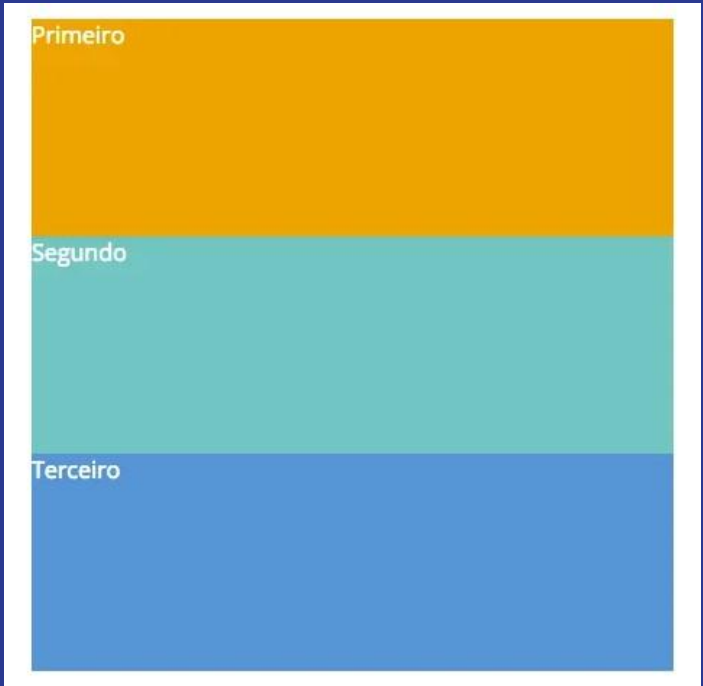
```html
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">

    <title>
        Exemplo dos valores inline, block e inline-block
        da propriedade display;
    </title>

    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/displayInlineBlock.css">
</head>
<body>
    <ul class="example">
        <li class="example-item example-item_first">Primeiro</li>
        <li class="example-item example-item_second">Segundo</li>
        <li class="example-item example-item_third">Terceiro</li>
    </ul>
</body>
</html>
```

```css
* {
    margin: 0;
    padding: 0;
    border: 0;
}
body {
    font-family: "Open Sans", sans-serif;
}
    ul, ol, li {
        list-style: none;
    }
```

```css
.example {
    color: #FFF;
}

    .example-item {
        width: 33.333%;
        height: 150px;
    }
    .example-item_first {
        background-color: #FAA116;
    }
    .example-item_second {
        background-color: #57C7C3;
    }
    .example-item_third {
        background-color: #3B97D3;
    }
```

MÓDULO X

O resultado desse código no navegador deve ficar parecido com:



Agora que temos um código de exemplo, vamos estudar os 3 valores (***inline***, ***block*** **e** ***inline-block***) da propriedade ***display***. Para tentar não deixar uma explicação chata, colocarei alguns desafios de como posicionar os elementos na tela e a ideia é discutirmos sobre quais dos 3 valores podemos utilizar.

MÓDULO X

Nosso primeiro desafio é colocar um elemento do lado do outro, dessa maneira:

Vamos implementar o *display: inline;* para todas as tags **<li>** no nosso arquivo **CSS (displayInlineBlock.css)**, ele ficará assim:

MÓDULO X

```
.example {
    color: #FFF;
}

    .example-item {
        width: 33.333%;
        height: 150px;
        display: inline; /* Só adicionamos essa linha. */
    }
    .example-item_first {
        background-color: #FAA116;
    }
    .example-item_second {
        background-color: #57C7C3;
    }
    .example-item_third {
        background-color: #3B97D3;
    }
```

Após alterar o CSS, dê uma atualizada no seu navegador. Você verá que o resultado não foi muito feliz:

Mas tudo bem, esse é um comportamento esperado quando utilizamos o ***display: inline;***.

Agora vamos dar uma pausa nos chutes e entender como funciona essa propriedade.

MÓDULO X

Como funciona o display: inline; ?

O valor **inline** permite que os elementos fiquem um do lado do outro

-- essa é a parte simples de perceber.

Nós temos definido uma largura e uma altura para todas as **<li>** em nosso código, mas as propriedades ***width*** e ***height*** não funcionaram. Esse também é um comportamento do **inline.**
Os elementos com **inline** não aceitam largura (***width***), nem altura (***height***), o tamanho do elemento é definido pelo conteúdo que ele contém.

...Nós não definimos nenhum respiro entre os elementos, mas tem algum tipo de *margin* entre eles. Poderia ser uma *margin* que o navegador colocou por conta própria mas, dado que nós temos um **reset.css** que está zerando e tirando a *margin*, *padding* e *border* de todos as tags, isso também não é possível.

MÓDULO X

Por que tem esses espaços entre tags que possuem a propriedade display: inline?

Esses espaços existem porque o elemento com ***display: inline***, ganha o **comportamento de uma palavra**. Além dos outros dois pontos que citamos acima.

O que estou querendo dizer com "**comportamento de uma palavra**"?

Para explicar um pouco melhor o comportamento de uma palavra, vamos adicionar um **<h1>** com os mesmo conteúdo das nossas tags **<li>** no nosso **HTML**:

MÓDULO X

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">

    <title>
        Exemplo dos valores inline, block e inline-block
        da propriedade display;
    </title>

    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/displayInlineBlock.css">
</head>
<body>
    <ul class="example">
        <li class="example-item example-item_first">Primeiro</li>
        <li class="example-item example-item_second">Segundo</li>
        <li class="example-item example-item_third">Terceiro</li>
    </ul>

    <h1 class="title">
        Primeiro
        Segundo
        Terceiro
    </h1>
</body>
</html>
```

Repare que as palavras dentro do **<h1>** estão separadas por um *[ENTER]*, da mesma forma que cada tag **<li>** também está separada por um *[ENTER]*. Depois de fazer a alteração na **index.html**, atualize o navegador e você terá esse resultado:

MÓDULO X

No nosso código, quando damos um *[ENTER]* para separar as **palavras** dentro do **<h1>,** essa quebra de linha não acontece no navegador pois ele troca o *[ENTER]* que está no código por um *[SPACE]*.
Lembra que cada **<li>** ganhou o **comportamento de uma palavra** após colocarmos o ***display: inline*** para eles? O espaço entre as tags **<li>** existe pelo mesmo motivo que temos os espaços entre as palavras que o **<h1>** contém.

Você deve estar se perguntando como faz para tirar os espaços entre as tags **<li>**.
Agora que entendemos o comportamento é muito simples, é só tirar o *[ENTER]* que está separando os elementos. Desta forma, nós transformaremos as três tags em uma palavra.

MÓDULO X

Nosso código ficará assim:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">

    <title>
        Exemplo dos valores inline, block e inline-block
        da propriedade display;
    </title>

    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/displayInlineBlock.css">
</head>
<body>
  <ul class="example">
    <li class="example-item example-item_first">Primeiro</li><!--
 --><li class="example-item example-item_second">Segundo</li><!--
 --><li class="example-item example-item_third">Terceiro</li>
  </ul>

 <h1 class="title">
    Primeiro
    Segundo
    Terceiro
  </h1>
</body>
</html>
```

Quando você atualizar o navegador o layout deve ficar assim:

MÓDULO X

Agora estamos sem os espaços entre os elementos, mas infelizmente, o nosso código não ficou nada elegante, e quanto mais tags **<li>** tivermos mais difícil ficará a leitura do código.
Uma outra maneira que a galera utiliza para remover os espaços tentando deixar o código um pouco mais elegante e mais fácil de ler, é utilizado comentários em vez de deixar as tags na mesma linha.
Veja como o código deve ficar com os comentários:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">

    <title>
        Exemplo dos valores inline, block e inline-block
        da propriedade display;
    </title>

    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/displayInlineBlock.css">
</head>
<body>
  <ul class="example">
    <li class="example-item example-item_first">Primeiro</li><!--
 --><li class="example-item example-item_second">Segundo</li><!--
 --><li class="example-item example-item_third">Terceiro</li>
  </ul>

 <h1 class="title">
    Primeiro
    Segundo
    Terceiro
  </h1>
</body>
</html>
```

MÓDULO X

O resultado desse código no navegador vai ser o mesmo de quando deixamos as tags **<li>** na mesma linha. Essas são duas formas de tirar o espaçamento entre as tags.

Por nossas tags **<li>** estarem **funcionando como palavras**, nós conseguimos mexer com o posicionamento delas com a propriedade *text-align*. Ela tem que ser usada no pai das tags **<li>**, por exemplo se nós quisermos que elas fiquem centralizadas, basta colocarmos na **<ul>** a propriedade *text-align: center*. Vamos implementar o *text-align: center* para a **<ul>** no nosso **CSS (displayInlineBlock.css)**:

MÓDULO X

```
.example {
    color: #FFF;
    text-align: center; /* Só adicionamos essa linha */
}

    .example-item {
        width: 33.333%;
        height: 150px;
        display: inline;
    }
    .example-item_first {
        background-color: #FAA116;
    }
    .example-item_second {
        background-color: #57C7C3;
    }
    .example-item_third {
        background-color: #3B97D3;
    }
```

Vá até o navegador e atualize a página, você terá esse resultado:

MÓDULO X

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">

    <title>
        Exemplo dos valores inline, block e inline-block
        da propriedade display;
    </title>

    <link rel="stylesheet" href="css/reset.css">
    <link rel="stylesheet" href="css/displayInlineBlock.css">
</head>
<body>
  <ul class="example">
    <li class="example-item example-item_first">Primeiro</li><!--
 --><li class="example-item example-item_second">Segundo</li><!--
 --><li class="example-item example-item_third">Terceiro</li>
  </ul>

  <h1 class="title">
    Primeiro
    Segundo
    Terceiro
  </h1>
</body>
</html>
```

Também podemos deixar nossas tags **<li>** posicionadas no lado direito.
Para fazer isso é só mudar o ***text-align*** para ***right***.
Agora você não precisa mais chutar quando for usar o valor ***inline*** da propriedade ***display***. Lembre-se de tirar o ***width*** e ***height*** do **CSS** já que essas propriedades não funcionam com ***display: inline***.
Nossos códigos finais tem que ficar assim:

MÓDULO X

CSS (displayInlineBlock.css)

```
.example {
    color: #FFF;
    text-align: center;
}

    .example-item {
        display: inline;
    }
    .example-item_first {
        background-color: #FAA116;
    }
    .example-item_second {
        background-color: #57C7C3;
    }
    .example-item_third {
        background-color: #3B97D3;
    }
```

CSS (reset.css)

```
* {
    margin: 0;
    padding: 0;
    border: 0;
}
body {
    font-family: "Open Sans", sans-serif;
}
    ul, ol, li {
        list-style: none;
    }
```

MÓDULO X

Float vs Inline Block - Qual usar em meus layouts?

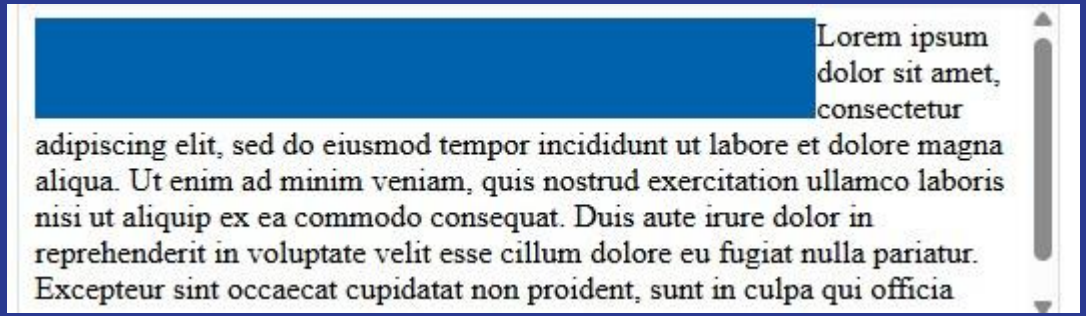
Ambos têm uma base parecida: permitir que elementos fiquem na mesma linha.
Os seus demais comportamentos são totalmente diferentes e muitos desenvolvedores acabam não sabendo quando devem usar um ou outro.

MÓDULO X

Float

As pessoas normalmente utilizam o **float** para desenvolver colunas em seus layouts ou permitir que elementos de bloco ocupem a mesma linha, mas esta propriedade foi criada originalmente para permitir que um texto envolva um elemento.

```
div{
  background-color: #0062ac;
  width: 80%;
  height: 50px;
  float: left;
}
```

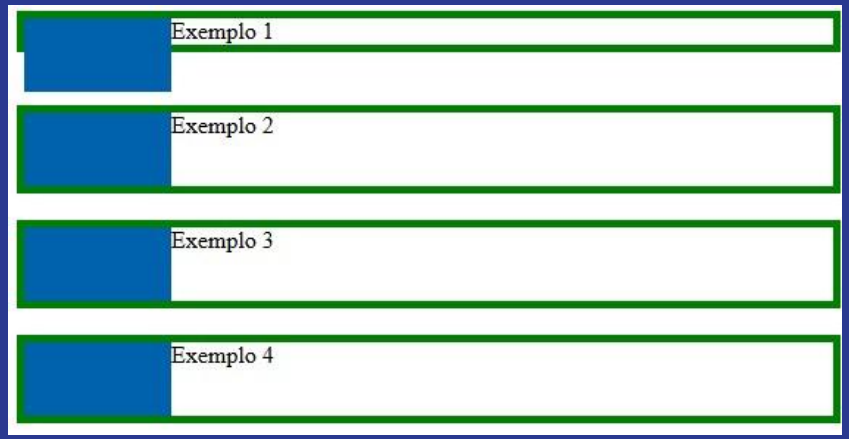


MÓDULO X

No exemplo anterior, nosso texto está envolvendo a nossa **div**.

Normalmente os elementos ocupam uma linha, e aqui temos três linhas ao lado dele.

Um dos problemas do **float** é que o elemento pai não vai saber a sua altura, já que o elemento está flutuando.



MÓDULO X

- No ***Exemplo 1*** podemos ver que a **div container**, com borda verde, apenas interpreta a altura do nosso texto. A altura do nosso elemento flutuante, a **div** azul, é ignorada.

- No ***Exemplo 2*** nós utilizamos outra **div** com a propriedade **clear** para poder limpar os efeitos do **float**, fazendo com que o fluxo dos elementos volte ao normal e o container possa interpretar a altura normalmente.

MÓDULO X

```
<div class="container">
  <div class="floating" ></div>
  Exemplo 2
  <p class="clear" ></p>
</div>
```

Muitas pessoas acham isso feio: ter que colocar um elemento vazio apenas para fazer a altura de um elemento ser interpretada. Isso também tira o sentido da semântica do código.
Muitos **frameworks**, como o ***bootstrap***, e bibliotecas de **grid**, como o ***unsemantic***, funcionam assim para poder dar suporte a navegadores mais antigos.

MÓDULO X

- No ***Exemplo 3*** nós utilizamos outro hack: a propriedade **overflow**: **auto** no container. Isso faz com que a gente não precise mais de um elemento apenas para encerrar o comportamento do **float**, mas só funciona bem se tivermos total controle das margens do **container**. Do contrário, você verá barras de rolagem.

```
.container-overflow{
  overflow: auto;
}
```

MÓDULO X

- No ***Exemplo 4*** temos algo mais complexo, porém mais confiável, muito utilizado em lugares como o **HTML5 Boilerplate**. Então, problemas com barra de rolagem resolvidos!

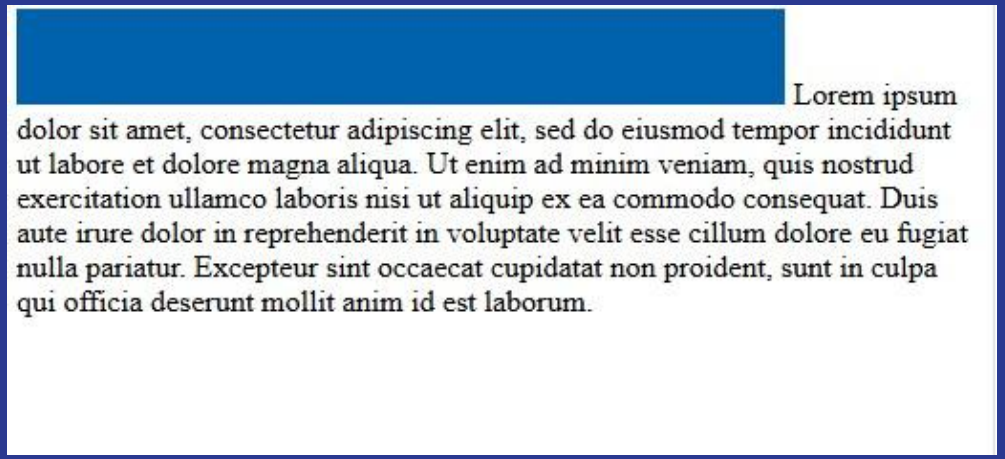
```
.clearfix:before, .clearfix:after { content: "020"; display: block; height: 0; overflow: hidden; }

.clearfix:after { clear: both; }

.clearfix { zoom: 1; }
```

Criamos uma classe **.clearfix** que cria um pseudo-elemento com o :**before** e :**after**.
Ele utiliza a ideia do **overflow** e **clear:both**, e não veremos nada estranho porque também escondemos os pseudo-elementos com **height**: **0**.

MÓDULO X

Inline Block

Quando utilizamos **display: inline-block**, também estamos fazendo com que um elemento permaneça na mesma linha.

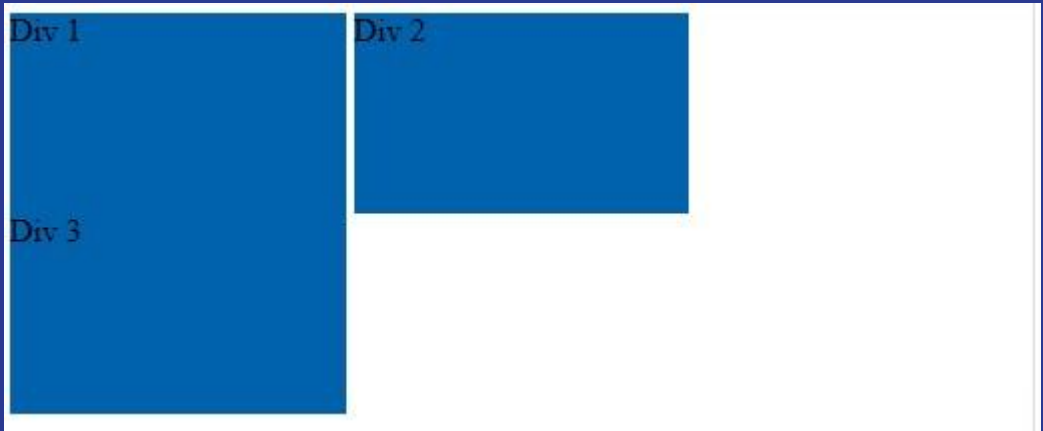


MÓDULO X

Note a diferença com o primeiro exemplo do **float**: nossa **div** está na mesma linha do texto, mas não são criadas novas linhas em volta dela.
Tudo bem, o objetivo aqui é manter elementos na mesma linha.
Aqui não teremos problemas com a altura do container ou barra de rolagem.

Outra vantagem é que como agora ele é um elemento com comportamento de linha, poderemos utilizar as propriedades **text-align** e **vertical-align** para modificar sua posição.
Teremos um probleminha quando precisarmos que os elementos estejam encostados

MÓDULO X



No exemplo acima criamos um layout que tem três colunas de larguras iguais que precisam preencher uma linha inteira. Note que entre a **Div 1** e a **Div 2** há um espaço.
Se inspecionarmos os elementos veremos que não há margens separando as divs.
Por quê esse espaço existe? Por causa dele, a nossa **Div 3** foi jogada para a linha de baixo!

MÓDULO X

Acontece que o inline-block faz o elemento ter comportamentos de um elemento de linha.

Os elementos de linha são basicamente **textos**!

```
<div class="container" >
  <div class="column">Div 1</div>
  <div class="column">Div 2</div>
  <div class="column">Div 3</div>
</div>
```

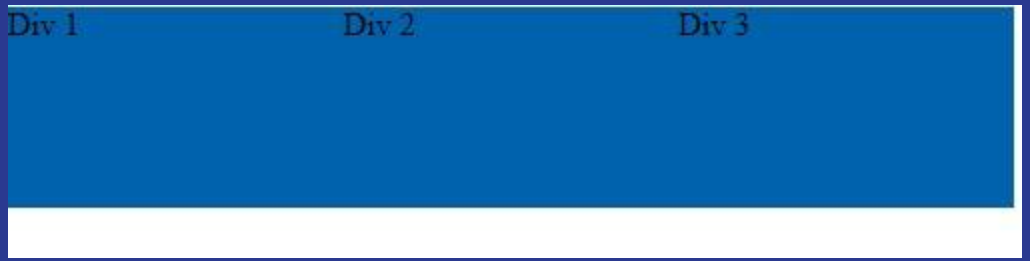
MÓDULO X

Quando escrevemos um texto, queremos que haja espaços entre as palavras, certo?
Então o navegador interpreta os espaços que damos entre as palavras.
Para organizar melhor o nosso ***HTML***, nós colocamos uma **div** em cada linha.
Essa quebra de linha no código é interpretado como espaço e acaba interferindo na renderização de elementos **inline** ou **inline-block**.

Caso você tenha o costume de minificar o seu código, os espaços serão removidos e você não terá esse problema.

MÓDULO X

Exemplo do resultado que queremos:



MÓDULO X

Aqui algumas maneiras de resolver isso com HTML:

```
<!-- Deixar tudo na mesma linha -->
<!-- Péssimo, pois deixa o código muito bagunçado -->
<div class="container" >
  <div class="column">Div 1</div><div class="column">Div 2</div><div class="column">Div 3</div>
</div>

<!-- Deixar a tag de fechamento junto à tag de abertura do elemento seguinte -->
<!-- Melhora um pouquinho, mas continua estranho -->
<div class="container" >
  <div class="column">Div 1
  </div><div class="column">Div 2
  </div><div class="column">Div 3</div>
</div>

<!-- Criar um comentário entre a quebra de linha -->
<!-- Muita gente prefere esse -->
<div class="container" >
  <div class="column">Div 1</div><!--
  --><div class="column">Div 2</div><!--
  --><div class="column">Div 3</div>
</div>
```

MÓDULO X

Aqui algumas maneiras de resolver isso com CSS:

```
/* Declarar uma margem negativa */
.column{
   margin-right: -4px;
}

/* Declarar font-size do container como 0 fará
*  o espaço ser 0 também, mas teremos que declarar
*  novamente o tamanho da fonte da nossa coluna.
*  Esse hack pode ter problemas em algumas versões
*  do Android e pode nos atrapalhar se nossos elementos
*  estiverem usando unidades como "em"
*/
.container{
   font-size: 0;
}
.column{
   font-size: 16px;
}
```

MÓDULO X

Concluindo

Float: Prefira usar quando quiser que um texto ou outros elementos contornem um determinado elemento. Essa é a real função dele.

Inline Block: Vimos aqui que o único problema do **inline-block** ocorre quando precisamos que os elementos fiquem encostados. Caso esse não seja o seu caso, o **inline-block** resolverá seus problemas sem a necessidade de algum hack.

Caso você não precise dar suporte total a navegadores mais antigos, então é mais recomendado que você monte seus layouts com as propriedades do ***flex-box***.

MÓDULO X

HTML Semântico: Conheça os elementos semânticos da HTML5

Atualmente o papel da **HTML** não é apenas estruturar documentos para a web, mas também descrever o significado do conteúdo presente nesses documentos por meio de tags semânticas. O **HTML** semântico tem como objetivo descrever o significado do conteúdo presente em documentos **HTML**, tornando-o mais claro tanto para programadores quanto para browsers e outras engines que processam essa informação.

MÓDULO X

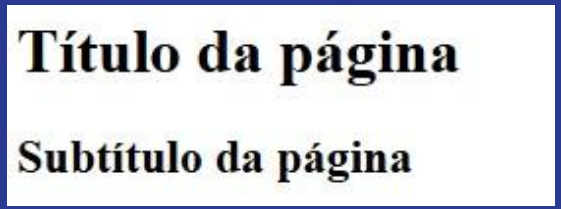
header

O <**header**> é utilizado para representar o cabeçalho de um documento ou seção declarado no **HTML**.

Nele podemos inserir elementos de <**h1**> a <**h6**>, até elementos para representar imagens, parágrafos ou mesmo listas de navegação.

Exemplo de uso de <**header**>:

```
<header>
    <h1>Título da página</h1>
    <h2>Subtítulo da página</h2>
</header>
```



MÓDULO X

section

O elemento <**section**> representa uma seção dentro de um documento e geralmente contém um título, o qual é definido por meio de um dos elementos entre <**h1**> e <**h6**>. Podemos utilizar o <**section**>, por exemplo, para descrever as seções/tópicos de um documento.

Exemplo de uso de <**section**>:

```
<section>
  <h3>Seção 1</h3>

  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
</section>
```

Seção 1

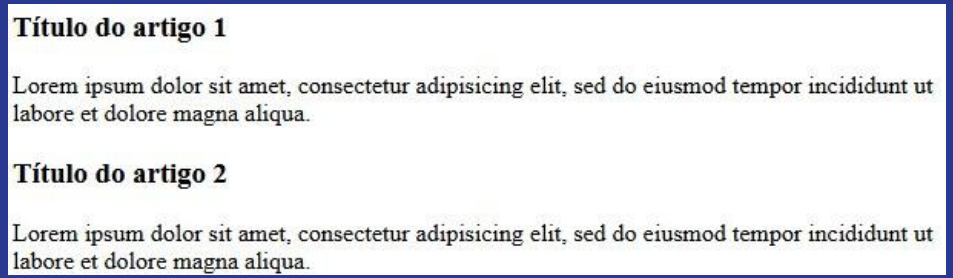
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

MÓDULO X

article

Utilizamos o elemento <**article**> quando precisamos declarar um conteúdo que não precisa de outro para fazer sentido em um documento **HTML**, por exemplo, um artigo em um blog.
É recomendado identificar cada <**article**> com um título.
Exemplo de uso de <**article**>:

```
<article>
  <h3>Título do artigo 1</h3>
  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
</article>

<article>
  <h3>Título do artigo 2</h3>
  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
</article>
```



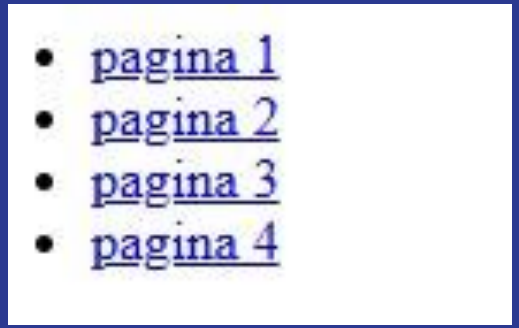
MÓDULO X

nav

O elemento <nav> é utilizado quando precisamos representar um agrupamento de links de navegação, que, por sua vez, são criados com os elementos <ul>, <li> e <a>.

Exemplo de uso de <nav>:

```
<nav>
  <ul>
    <li><a href="#">pagina 1</a></li>
    <li><a href="#">pagina 2</a></li>
    <li><a href="#">pagina 3</a></li>
    <li><a href="#">pagina 4</a></li>
  </ul>
</nav>
```



MÓDULO X

aside

O elemento <**aside**> é utilizado quando precisamos criar um conteúdo de apoio/adicional ao conteúdo principal. Por exemplo, ao falar de **HTML** semântico, podemos indicar ao leitor outros conteúdos sobre a linguagem **HTML** como sugestão de leitura complementar.

Exemplo de uso de <**aside**>:

```
<aside>
  <nav>
    <ul>
      <li>Link 1</li>
      <li>Link 2</li>
      <li>Link 3</li>
      <li>Link 4</li>
    </ul>
  </nav>
</aside>
```

- Link 1
- Link 2
- Link 3
- Link 4

MÓDULO X

main

O elemento <main> especifica o conteúdo principal e, consequentemente, de maior relevância dentro da página. Para ser considerada bem construída, uma página deve apresentar apenas um conteúdo principal.

Exemplo de uso de <main>:

```
<main>
  <h2>Titulo</h2>
  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
  <article>
    <h3>Subtítulo</h3>
    <p>Duis aute irure dolor in reprehenderit cillum dolore eu fugiat nulla pariatur. officia deserunt mollit anim id est laborum.</p>
  </article>
</main>
```

Titulo

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Subtítulo

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

MÓDULO X

figure

O elemento <**figure**> é uma marcação de uso específico para a inserção de uma figura.

Para incluir a descrição dessa figura, podemos utilizar o elemento <**figcaption**>.

Exemplo de uso de <**figure**>:

```
<figure>
  <img src="http://www.devmedia.com.br/Imagens/2013/logo_mvp.png" alt="DevMedia">
</figure>
```

MÓDULO X

Exemplo de uso de <**figure**> com <**figcaption**>:

```
<figure>
  <img src="http://www.devmedia.com.br/Imagens/2013/logo_mvp.png" alt="Imagem">

  <figcaption>Figura 1. Imagem</figcaption>
</figure>
```

Figura 1. Imagem

MÓDULO X

footer

O elemento <**footer**> representa um rodapé de um documento, como a área presente no final de uma página web. Normalmente é utilizado para descrever informações de autoria, como nome e contato do autor, e data de criação do conteúdo.

Exemplo de uso de <**footer**>:

```
<footer>
  <p>Escrito por Estevão Dias</p>
  <p>Publicado em 25/03/2017 </p>
</footer>
```

Escrito por Estevão Dias

Publicado em 25/03/2017

MÓDULO X

Desenvolvimento responsivo: como resetar os estilos com CSS

Com o alto número de navegadores no mercado, um dos principais problemas para quem constrói websites responsivos é que cada um deles pode possuir diferentes valores padrão para a margem e o padding (margem interna) dos elementos. Isso faz com que sejam obtidos resultados visuais distintos, o que acaba prejudicando, exatamente, a responsividade.

MÓDULO X

Seção 1: Entendendo o problema

Para entendermos esse problema na prática, criamos um **HTML** básico, com apenas um texto, como mostrado a seguir:

```
1 <body>
2   <h2>DevMedia - Reset com CSS para desenvolvimento responsivo</h2>
3 </body>
```

A partir desse código, conseguimos observar as diferenças entre o visual do elemento sem o **CSS Reset** aplicado (**Figura 1**) e com esse recurso (**Figura 2**). Na **Figura 1**, nota-se, com muita clareza, que existe uma certa distância do texto para a margem nos três navegadores (**Google Chrome, Mozilla Firefox e Microsoft Edge**).

MÓDULO X

Isso é muito prejudicial ao desenvolvimento responsivo, principalmente por um motivo: o desenvolvimento responsivo, em geral, baseia-se em porcentagens. Imagine um layout com quatro colunas, cada uma com **25**% da largura da página. Caso não haja o reset, esses **100**% seriam somados aos pixels de **margem**/***padding*** do navegador, quebrando o layout e fazendo com que a quarta coluna fique na linha de baixo.

Figura 1. HTML exemplo sem CSS reset.

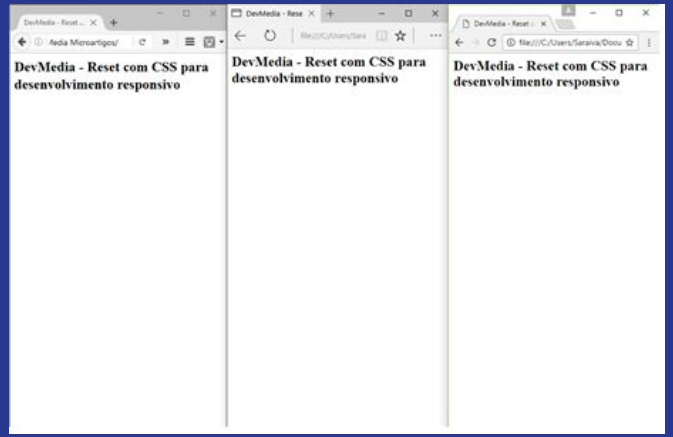
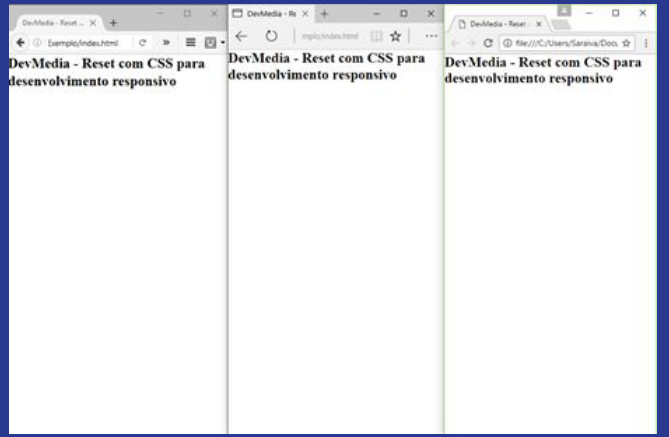


Figura 2. HTML exemplo com CSS reset.



MÓDULO X

Seção 2: Reset específico para alguns elementos

O **reset do CSS dos elementos é uma operação bastante simples**, porém, muitas vezes acabamos esquecendo desse passo quando começamos a desenvolver de forma responsiva. No geral, isso leva a designs “quebrados” em alguns navegadores, mas que potencialmente funcionam bem em outros.

No caso de nosso exemplo, precisamos resetar apenas os elementos <**html**>, <**body**> e <**h2**>, uma vez que estamos utilizando somente eles. Dessa forma, um **CSS** como o da **Listagem 1** seria suficiente.

```
1  html, body, h2
2  {
3      margin: 0;
4      padding: 0;
5  }
```

Listagem 1. CSS para resetar somente os elementos necessários

MÓDULO X

Linha 01: definição dos elementos que serão resetados: **<html>**, **<body>** e **<h2>**;

Linha 03: zeramos a propriedade CSS “margin”, que elimina a margem padrão que os navegadores definem para os elementos;

Linha 04: zeramos a propriedade CSS “padding”, que elimina a “margem interna” que os navegadores definem para os elementos.

Uma ação simples como essa é suficiente para termos o layout mostrado na **Figura 2**, possibilitando um design idêntico em todos os navegadores. Também é importante ressaltar que, caso tivéssemos mais elementos no HTML, bastaria zerá-los também. Por exemplo, se tivéssemos um elemento <**div**>, teríamos que zerá-lo da mesma maneira que a ensinada na **Listagem 1**.

MÓDULO X

Seção 3: Reset para todos os elementos

Caso a complexidade do **HTML** aumente, podemos concluir que a abordagem anterior pode prejudicar a construção da interface. Seria necessário controlar o **reset** de muitos elementos. Diante disso, o CSS traz uma outra possibilidade de reset, mais simples, e que permite que tenhamos todos os elementos do **DOM** como alvo: através do operador *. Vejamos um exemplo no código abaixo:

```
1 | * { margin: 0; padding: 0; }
```

Com isso, ao invés de informarmos cada elemento do **DOM** que será resetado, com o operador * determinamos que todos eles o serão. Assim, é garantido que, independentemente do que for inserido no **HTML**, teremos um design responsivo sem diferenças entre os navegadores.

MÓDULO X

Pseudo Classes e pseudo elementos das CSS

> *prefixo: pseudo*
>
> 1. De teor falso; cujo conteúdo não corresponde à realidade.

Pseudoclasses e **pseudoelementos** representam um espécie de estado "**fantasma**" ou uma caracterização específica de um elemento que pode ser estilizado. Algumas **pseudoclasses** mais conhecidas são:

:link*, *:visited*, *:hover*, *:active*, *:first-child* e *:nth-child.

Existem outras e nós iremos estudá-las a seguir.

Pseudoclasses têm sua sintaxe iniciada pelo sinal de dois pontos (**:**) seguida pelo nome da **pseudoclasse** e em alguns casos por um valor escrito entre parênteses.

Pseudoelementos são como elementos virtuais que podem ser tratados como elementos regulares da marcação **HTML**. Contudo, eles não existem na árvore do documento (no DOM) pois são criados com uso de **CSS**.

Alguns pseudoelementos mais conhecidos são: ***:after*, *:before* e *:first-letter***.

MÓDULO X

Dois pontos simples ou dobrados?

A resposta curta é: na maioria dos casos é indiferente.

A sintaxe com dois pontos dobrados (**::**) foi criada pelas **CSS3** para estabelecer uma diferenciação entre pseudoelementos, tais como, ***::before*** **e** ***::after*** e pseudoclasses, tais como, ***:hover*** e ***:active***. Todos os navegadores, exceto os **IE8** e anteriores, oferecem suporte para dois pontos dobrados na sintaxe para pseudoelementos.

Alguns pseudoelementos, tal como, ***::backdrop*** admitem apenas a sintaxe com dois pontos dobrados.

Eu prefiro a sintaxe com dois pontos simples, para fazer com que meu **CSS** seja compatível com navegadores antigos. Uso sintaxe com dois pontos dobrados para os pseudoelementos que exigem esta sintaxe e não admitem sintaxe com dois pontos simples.

Você pode usar qualquer uma das sintaxes, pois, neste casos, não existe uma maneira certa ou errada.

MÓDULO X

É bom ressaltar que a especificação, na data em que este artigo foi escrito, recomenda a sintaxe com dois pontos simples pela razão já citada de compatibilidade com navegadores antigos (**retrocompatibilidade**):

> Notar que a nova maneira CSS3 de escrever CSS preconiza o uso de dois pontos dobrados na sintaxe para pseudoelementos, por exemplo: `a::after { … }`, cuja finalidade é a de estabelecer diferenciação com as pseudoclasses. Você vai encontrar esta sintaxe nas CSS. Contudo as CSS3 preveem também a sintaxe com uso de dois pontos simples para os pseudoelementos, por razões de retrocompatibilidade. Assim, nós recomendamos que você, pelo menos por enquanto, adote a sintaxe com dois pontos simples.

MÓDULO X

Quando (não) usar conteúdo gerado

Gerar, ou criar, conteúdo com **CSS** é possível com a declaração da propriedade ***content*** para os pseudoelementos ***:before*** ou ***:after***.

O valor de "**content**" pode ser tanto texto puro como um container a ser manipulado com **CSS** para mostrar um gráfico, uma forma ou um elemento decorativo.
Neste exemplo o conteúdo criado foi do tipo texto.

Não se deve criar conteúdos textuais com **CSS** que sejam relevantes para o entendimento do documento pelas seguintes razões:

- **Eles não são acessíveis a todos os leitores de tela.**
- **Eles não são selecionáveis.**
- **Se o conteúdo criado for irrelevante, apenas com fins decorativos, os leitores de tela que os suportam, lerão o conteúdo, interferindo de forma inconveniente na experiência do usuário.**

MÓDULO X

Pseudo Classes e pseudo elementos experimentais

Pseudoclasses e pseudoelementos experimentais são aqueles previstos nas especificações, mas que ainda não atingiram o estágio de estáveis ou finalizados. A sintaxe e comportamento de tais funcionalidades podem ser alteradas ao longo da evolução das especificações.

Mas, já é possível fazer experiências com as pseudoclasses e pseudoelementos experimentais valendo-se do uso de prefixos proprietários. Para informações a respeito consulte **Can I Use** e também algumas ferramenta para auto prefixação, tais como, **-prefix-free** or **Autoprefixer**

Foi colocado o rótulo "**experimental**" ao lado do nome das pseudoclasses e pseudoelementos que se encontram nesta condição.

MÓDULO X

Índice dos conteúdos (em ordem alfabética)

MÓDULO X

- :lang
- :last-child
- :last-of-type
- :link
- :not
- :nth-child
- :nth-last-child
- :nth-last-of-type
- :nth-of-type
- :only-child

- :nth-of-type
- :only-child
- :only-of-type
- :optional
- :out-of-range
- ::placeholder (experimental)
- :read-only
- :read-write
- :required
- :root

- ::selection
- :scope (experimental)
- :target
- :valid
- :visited
- Bonus content: A Sass mixin for links

MÓDULO X

Estados

Um estado da pseudoclasse acontece quando o usuário desencadeia uma ação.

Uma "ação" em **CSS** pode ser também uma "não ação" como no caso de um link que ainda não tenha sido visitado.

MÓDULO X

:link

A pseudoclasse ***:link*** o estado "normal" dos links que ainda não tenham sido visitados. Recomenda-se que a pseudoclasse ***:link*** seja a primeira a ser declarada entre todas aquelas destinadas a estilizar os estados dos links. A ordem de declaração dos quatro estados é: ***:link*, *:visited*, *:hover*, *:active***

```
a:link { color: orange; }
```

A pseudoclasse pode ser omitida como mostrado a seguir:

divh4 class="titulo-codigo">CSSdivh4>

```
a { color: orange; }
```

MÓDULO X

:visited

A pseudoclasse ***:visited*** é usada para estilizar links visitados e deve ser a segunda a ser declarada (depois da pseudoclasse ***:link***).

```
a:visited { color: blue; }
```

MÓDULO X

:hover

A pseudoclasse ***:hover*** é usada para estilizar links no momento em que o usuário coloca o dispositivo apontador sobre um elemento. Essa pseudoclasse não é exclusiva para links embora seja seu uso mais comum.

Deve ser a terceira a ser declarada (depois da pseudoclasse ***:visited***).

```
a:hover { color: orange; }
```

MÓDULO X

:active

A pseudoclasse ***:active*** é usada para estilizar um elemento no momento em que o usuário o torna "**ativo**" quer com uso de um dispositivo apontador que tocando-o em um dispositivo do tipo touchscreen.

Pode-se ativá-lo também com uso do teclado tal como se faz com a pseudoclasse ***:focus***.

Seu comportamento e semelhante ao de ***:focus***, com a diferença que ***:active*** é um evento que ocorre entre o apertar e soltar no clique do mouse.

Deve ser a quarta a ser declarada (depois da pseudoclasse ***:hover***).

```
a:active { color: rebeccapurple; }
```

MÓDULO X

:focus

A pseudoclasse:***focus*** é usada para estilizar um elemento que ganha o foco quer com uso de um dispositivo apontador que tocando-o em um dispositivo do tipo touchscreen, que pelo teclado.

É muito usado em campos de formulário.

```
a:focus { color: green; }
```

Ou

```
input:focus { background: #eee; }
```

MÓDULO X

Bônus: Sass Mixin para links

Se você usa pré-processadores **CSS**, tal como Sass, este bônus poderá ser do seu interesse.

(Se você não usa pré-processadores — e não há nada contra isso — sinta-se à vontade para ir direto a pseudoclasses estruturais

O **mixin Sass** tem por finalidade otimizar o fluxo de trabalho criando um conjunto de regras **CSS** para os links.

A ideia ao criar o mixin foi a de não declarar argumentos default. Assim, seremos "**forçados**" a declarar explicitamente os quatro estados dos links.

As pseudoclasses ***:focus*** e ***:active*** em geral são declaradas em conjunto. Covém notar que o mixin sugerido poderá ser aplicado a qualquer elemento da marcação **HTML** e não somente aos links.

MÓDULO X

Pseudoclasses estruturais

Pseudoclasses estruturais casam com porções da árvore do documento (ou **DOM**) que não podem ser casadas com uso de outros tipos de seletores ou elementos de combinação de seletores.

:first-child

A pseudoclasse ***:first-child*** casa com o elemento primeiro filho de seu elemento pai.

:first-of-type

A pseudoclasse ***:first-of-type*** casa com o primeiro elemento de um determinado tipo que seja filho de seu elemento pai.

MÓDULO X

:last-child

A pseudoclasse ***:last-child*** casa com o elemento último filho de seu elemento pai.

:last-of-type

A pseudoclasse ***:last-of-type*** casa com o último elemento de um determinado tipo que seja filho de seu elemento pai.

:not

A pseudoclasse ***:not*** é também conhecida como pseudoclasse de negação.

Ela aceita um argumento — basicamente um outro "**selector**" — escrito entre sinais de parênteses.

O argumento poderá ser, também, uma pseudoclasse.

O argumento pode ser encadeado, mas não pode conter outro seletor ***:not***.

MÓDULO X

:nth-child

A pseudoclasse ***:nth-child*** casa com um ou mais elementos dependendo da ordem em eles aparecem na marcação **HTML**.

Esta pseudoclasse é uma das mais versáteis e robustas das CSS.

A pseudoclasse *:nth* admite um argumento que consiste em uma fórmula a ser inserida entre parênteses.

A fórmula poderá ser um simples número inteiro, algo do tipo ***an+b*** ou as palavras chaves *odd* ou *even*.

A fórmula ***an+b*:**

- a letra ***a*** é um número inteiro;
- a letra ***n*** é o literal *n* (ou seja, a própria letra *n*);
- o sinal + é o operador matemático que pode ser soma (+) ou subtração (-);
- a letra ***b*** é um número inteiro que deverá estar presente na fórmula somente se for usado o operador matemático.

MÓDULO X

:nth-last-child

A pseudoclasse ***:nth-last-child*** é idêntica à pseudoclasse ***:nth-child*** exceto que o casamento se dá contando-se os elementos filhos de trás para frente.

:nth-of-type

A pseudoclasse ***:nth-of-type*** é idêntica à pseudoclasse ***:nth-child***, a principal diferença é que se trata de um seletor mais específico, pois casa com elementos filhos de determinado tipo.

:nth-last-of-type

A pseudoclasse ***:nth-last-of-type*** é idêntica à pseudoclasse ***:nth-of-type*** exceto que o casamento se dá contando-se os elementos filhos de trás para frente.

MÓDULO X

:only-child

A pseudoclasse ***:only-child*** casa com o filho único de um elemento.

:only-of-type

A pseudoclasse ***:only-of-type*** casa com elementos que não têm irmãos do seu tipo.

Ele é idêntico ao elemento ***:only-child*** exceto pelo fato de que casa com um determinado tipo de elemento o que faz com que o seletor seja mais específico.

:target

A pseudoclasse ***:target*** casa com o elemento ao qual foi definido um atributo **ID** (único).

MÓDULO X

Validação

Formulários são uma funcionalidade do desenvolvimento web que normalmente requerem cuidados especiais com relação a experiência do usuário. O uso de pseudoclasses poderá ser um poderoso aliado do desenvolvedor na validação, fazendo o processo de preenchimento de campos uma experiência bem melhor para o usuário.

Convém notar que a maioria das pseudoclasses que estudaremos a seguir além de serem úteis para validação de campos de formulários podem também serem usadas com outros elementos da marcação **HTML**.

MÓDULO X

:checked

A pseudoclasse *:checked* casa com radio buttons, checkboxes e elementos option que tenham sido marcados ou selecionados.

:default

A pseudoclasse ***:default*** casa com o elemento default de um formulário entre um grupo de elementos similares.

Se você precisar estilizar um botão default de um formulário ao qual não tenha sido definido um atributo classe use a pseudoclasse ***:default***.

Notar que o uso de botão do tipo "**Reset**" ou "**Clear**" em um formulário pode acarretar sérios **problemas de usabilidade**. Evite usá-lo a menos que seja absolutamente necessário.

MÓDULO X

:disabled

A pseudoclasse ***:disabled*** casa com elementos de formulário que tenham sido desabilitados.

Elementos desabilitados não podem ser selecionados, marcados, ativados e nem ganhar foco.

:enabled

A pseudoclasse ***:enabled*** casa com elementos que estão habilitados. Todos os elementos de formulário estão habilitados por padrão — exceto se definirmos, para o elemento o atributo ***disabled*** na marcação **HTML**.

Pode-se usar uma combinação de ***:enabled*** e ***:disabled*** para fornecer **feedback**, melhorando assim a experiência do usuário.

MÓDULO X

:empty

A pseudoclasse *:empty* casa com elementos que não têm qualquer tipo de conteúdo, ou seja, vazios.

Se um elemento contiver uma simples letra, outro elemento HTML ou até mesmo um espaço em branco ele não será um elemento empty (vazio).

- **Empty (vazio)**

Nenhum conteúdo ou caractere deve existir dentro do elemento.

- Um comentário HTML não é considerado conteúdo nestes casos.

- **Not empty (não vazio)**

Existe conteúdo ou caracteres dentro do elemento.

- Até mesmo espaços em branco são considerados conteúdos.

MÓDULO X

:valid

A pseudoclasse ***:valid*** casa com campos de formulário cujos dados de entrada estejam de acordo

com o formato requerido para o campo.

:invalid

A pseudoclasse *:invalid* casa com campos de formulário cujos dados de entrada estejam em desacordo com o formato requerido para o campo.

:required

A pseudoclasse ***:required*** casa com elementos input aos quais foi definido o atributo ***required*** na marcação **HTML**.

A prática de colocar um sinal asterísco **(*)** ao lado de um campo de preenchimento obrigatório pode ser complementada com estilização diferenciada do campo com uso desta pseudoclasse.

Estamos assim, aproveitando o máximo dos dois mundos, **HTML** e **CSS**.

MÓDULO X

:lang

A pseudoclasse ***:lang*** casa elementos aos quais tenha sido definido o atributo ***lang***="" da **HTML**,

com elementos cujo idioma tenha sido definido com uso de *meta* elementos ou com protocolos, tal como, cabeçalhos **HTTP**.

o atributo ***lang***="" da **HTML** em geral é definido para a tag ***html***, mas pode ser definido para qualquer elemento da marcação, conforme necessário.

Convém notar que é prática comum usar sinais de aspas (**quotation marks**) para marcar idiomas, definindo-se a propriedade **CSS*quotes***. Contudo a maioria dos agentes de usuário (UA), incluídos aí os **IE 9+**, são capazes de adicionar nativamente as aspas, caso elas não tenham sido declaradas nas **CSS**.

Dependendo das circunstâncias isso pode ou não ser conveniente, pois existem diferenças entre as aspas inseridas nativamente pelo navegador e aquelas que você é capaz de criar com uso de **CSS**.

MÓDULO X

:root

A pseudoclasse ***:root*** casa com o elemento pai mais alto (raiz) em um documento.

Virtualmente em todos os casos a pseudoclasse ***:root*** casa com o elemento ***html*** quando se trata de um documento **HTML**. Em outras linguagens de marcação de documentos, tais como **XML** e **SVG**, esta classe casa com outros elementos .

MÓDULO X

Pseudoelementos

Conforme foi dito no início deste artigo pseudoelementos são uma espécie de elementos virtuais que podem ser tratados como elemento HTML regular. Eles não existem na árvore do documento (no DOM), pois não são inseridos na marcação HTML, eles são criados com uso de CSS.

Também foi dito que dois pontos dobrados **(::)** e dois pontos simples **(:)** se destinam meramente a diferenciar visualmente as sintaxes **CSS 2.1** e **CSS3** para os pseudoelementos.

::before/:before

O pseudoelemento ***:before*** e seu vizinho ***:after*** cria conteúdo (textual ou de forma) em um elemento **HTML** regular. Ressaltamos mais uma vez que tais conteúdos não estão presentes no DOM embora possam ser manipulados como se estivessem. A propriedade ***content*** é uma funcionalidade das CSS.

Lembre-se que texto criado com uso deste pseudoelemento não pode ser selecionado.

MÓDULO X

::after/:after

O pseudoelemento ***:after*** é usado para criar conteúdo (seja texto ou formas) em um elemento HTML.

Tais conteúdos não serão incluídos no **DOM**, mas podem ser manipulados com **CSS** como se ali estivessem. Para que isso seja possível é necessário que a propriedade ***content*** seja declarada nas **CSS**.

Convém notar que conteúdos textuais assim criados não podem ser selecionados.

::selection

O pseudoelemento ***::selection*** casa com a porção do documento que tenha sido selecionada.

Até segunda ordem os navegadores com engine baseadas em Gecko requerem o prefixo proprietário ***::-moz-selection.***

Nota: Combinar as versões com prefixo e sem prefixo em uma simples regra **CSS** não funcionará.

É preciso declarar regras separadas para as duas versões.

MÓDULO X

Pseudoclasses e pseudoelementos são bastante úteis, não é mesmo?

Estas funcionalidades oferecem tantas possibilidades que podemos até nos sentir intimidados com elas. Mas, é assim a vida do web designer e do desenvolvedor.

Certifique-se de testar exaustivamente.

Pseudoclasses e pseudoelementos bem implementados oferecem muitas possibilidades.

MÓDULO X

EXERCÍCIOS...

A Educação é o primeiro passo para um futuro melhor...

MÓDULO X

FIM...

A Educação é o primeiro passo para um futuro melhor...