**AULA 8.1 – PSEUDOCLASSES do navegador**

Na aula passada vimos alguns seletores novos, o "~", o "+" e o ">". Apesar de muitas vezes nos ajudar, eles possuem alguns problemas como:

* Legibilidade: é difícil agrupar mais de um símbolo e deixar claro o que queremos com eles;
* Desempenho: o navegador terá muito trabalho para analisar toda a estrutura do HTML;
* Fragilidade: a chance de errarmos no código na hora de implementá-lo é grande.

Por essas e outras razões, vamos utilizar uma outra ferramenta que facilitará nossa vida e terá resultados parecidos com os dos seletores.

Imaginemos que vamos montar esta tabela zebrada, com as linhas de cores diferentes:



Seu HTML seria neste formato:

<table>

<tr>...</tr>

<tr>...</tr>

<tr>...</tr>

<tr>...</tr>

<tr>...</tr>

</table>

Queremos dar um estilo para as linhas pares (*even*) e outro para as ímpares (*odds*). Podemos fazer isso criando classes:

<table>

<tr class="odd">...</tr>

<tr class="even">...</tr>

<tr class="odd">...</tr>

<tr class="even">...</tr>

<tr class="odd">...</tr>

</table>

Essa formatação é muito comum, então não vamos inserir as classes dentro do HTML. Vamos, por sua vez, criar as ***pseudoclasses***.

As *pseudoclasses* são chamadas com os ":" (dois pontos):

:odd

...

:even

Na linguagem que estamos usando, generalizamos essas *pseudoclasses* com "*:nth-child()*". Dessa maneira, podemos estilizar não só as linhas pares ou ímpares:

* **:nth-child(odd)** - linhas ímpares;
* **:nth-child(even)** - linhas pares;
* **:nth-child(3)** - terceira linha;
* **:first-child** - primeira linha;
* **:last-child** - última linha.

Também podemos inserir fórmulas dentro do *:nth-child()* como *nth-child(2n)* que nada mais é do que as linhas pares. Ou *nth-child(2n+1)*, que são as ímpares.

Generalizando: **:nth-child(xn + c)**

Em que:

* **x** é a periodicidade, de quantos em quantos elementos será aplicado o estilo;
* **c** é o ponto de partida;
* **n** é a variável, que começa em 0 (zero).

É importante lembrar que não temos apenas a tabela na nossa página. Podemos ter parágrafos, por exemplo. Se aplicarmos **:nth-child(odd)** em um HTML desse tipo:

<p>...</p>

<p>...</p>

<table>

<tr>...</tr>

<tr>...</tr>

<tr>...</tr>

<tr>...</tr>

<tr>...</tr>

</table>

<p>...</p>

<p>...</p>

O estilo será aplicado em todos os elementos ímpares, ou seja, no primeiro parágrafo, na tabela e no quinto parágrafo, além dos elementos filhos ímpares dentro da tabela. Então é comum especificarmos esse seletor agrupando-o com outro, a *tag*:

tr:nth-child(odd)

ou

table:nth-child(odd)

Neste caso usamos as *tags* tr e table para especificar o seletor na tabela, só aplicando nas linhas ímpares.

Não confunda o agrupamento de seletores, que é específico para o mesmo elemento e são escritos juntos:

table:first-child (a tabela que é o primeiro filho)

Com os seletores funcionando em nível hierárquico (elementos diferentes), escritos separadamente:

tr :first-child (a tabela é pai e "first-child" é seu primeiro filho)

Vamos pegar outro caso para resolvermos. Na página de biografia do João da Silva havia os *block-quotes* organizados dessa maneira:

1 <p>

2 <p>

3 <h2>

4 <blockquote>

5 <p>

6 <p>

7 <h2>

8 <blockquote>

9 <p>

10 <p>

11 <h2>

12 <blockquote>

Tínhamos que inserir classes específicas para cada um. Vamos limpar o código utilizando o *nth-child* dessa maneira:

blockquote:nth-child(even)

Pode parecer que agora todos os *block-quotes* de número par terão o mesmo estilo. Porém a contagem da pseudoclasse vale para todo mundo, ou seja, ela pegará todos os *block-quotes* que estão em posições pares, nesse caso englobaria os três. Todos são filhos pares do *body*. Para resolver por fim esse problema, podemos pedir que conte apenas os *block-quotes* utilizando

blockquote:nth-of-type(even)

Dessa maneira a contagem da pseudoclasse fica

<p>

<p>

<h2>

1 <blockquote>

<p>

<p>

<h2>

2 <blockquote>

<p>

<p>

<h2>

3 <blockquote>

Revendo, vimos nesta aula quatro pseudoclasses:

* **:first-child**
* **:last-child**
* **:nth-child**
* **:nth-of-type**

Com o uso das pseudoclasses, mantemos o poder que os seletores avançados possuíam, porém com elas temos uma melhor legibilidade e, por sua vez, um melhor desempenho.

Um outro recurso das pseudoclasses é que algumas delas são dinâmicas, o que nos permite fazer efeitos de interatividade. Imagine um exemplo onde temos um botão que muda de cor quando o ponteiro do *mouse* fica sobre ele:



Uma forma de fazer isso é via *javascript*:

<a href="...">Compre já</a>

<a href="..." class="destaque">

Este é outro efeito muito comum nas páginas da web, então o próprio navegador pode inserir essa classe para nós:

<a href="..." class="hover">

A pseudoclasse *hover* (passar por cima) que nos permite esse efeito. O código para fazer funcionar recebe a classe ".botao" com a formatação que quisermos:

.botao {

border: 8xp solid purple;

}



Quando o *hover* estiver no mesmo elemento, ou seja, o seletor será aplicado na classe *.botão* e na pseudoclasse *:hover*, o estilo muda:

.botao {

border: 8px solid purple;

}

.botao:hover {

background-color: purple;

}



Com isso temos o efeito que queríamos sem precisar do *javascript*.

Então, quando estamos falando de *pseudoclasses* não nos limitamos às *estruturais*, temos uma gama de *dinâmicas* também:

<table>

<thead>

<tr>

<th style="text-align:center">Estruturais</th>

<th style="text-align:center">Dinâmicas</th>

</tr>

</thead>

<tbody>

<tr>

<td style="text-align:center">:first-child</td>

<td style="text-align:center">:hover</td>

</tr>

<tr>

<td style="text-align:center">:last-child</td>

<td style="text-align:center">:focus</td>

</tr>

<tr>

<td style="text-align:center">:nth-child</td>

<td style="text-align:center">:active</td>

</tr>

<tr>

<td style="text-align:center">:nth-of-type</td>

<td style="text-align:center">:checked</td>

</tr>

</tbody>

</table>