

1) Padrões Web e xHTML

Quanto aos atributos de determinada tag, sempre que declarados deverão ter algum valor, valor este que estará entre aspas duplas. A seguir uma comparação de atributos de uma tag declarados em HTML e em XHTML:

Em HTML: `<TD COLSPAN=2 NOWRAP>`

Em XHTML: `<td colspan="2" nowrap="nowrap"></td>`

Ainda rumo à evolução, algumas tags e atributos foram removidos das especificações do W3C. Tais tags são marcadas como *deprecated* (arcaico) e podem não ser reconhecidas por navegadores mais recentes.

Estas são algumas pequenas mudanças que foram aplicadas para a montagem da especificação do XHTML. Outras normas e especificações serão abordadas nos tópicos a seguir.

2) Mark-up Semântico

Segundo definição do dicionário Aurélio, semântica é “o estudo da relação de significação nos signos e da representação do sentido dos enunciados”. A partir dessa definição, podemos entender que o termo “Mark-up Semântico” refere-se a codificação de páginas da web nas quais as marcações (tags) são utilizadas corretamente, dando mais sentido ao código, com o objetivo de facilitar sua leitura, entendimento e funcionalidades numa análise futura do programador.

Podemos dizer ainda que elementos específicos são utilizados para assuntos específicos, aspecto muito importante para o desenvolvimento web moderno. O XHTML traz uma gama de elementos, cada qual com uma finalidade definida, embora possa ter mais do que um único uso. Como as linguagens de mark-up são muito flexíveis, é possível que elementos sejam utilizados de forma incorreta ou incoerente, cabendo aos desenvolvedores estarem atentos na hora de programar e fazer o uso correto do elemento.

Dois exemplos clássicos de tags que vem sendo abandonadas e cujo uso não mais é recomendado pelo W3C, são as tags `<i>` e ``, que respectivamente, possuem a funcionalidade de deixar o texto em itálico (italic) e negrito (bold). Contudo, a utilização das tags `` (emphasis) e `` (forte) em substituição às primeiras, fazem maior sentido ao lermos o código fonte de uma página web.

Vamos agora fazer a análise do trecho de código HTML abaixo, onde os conceitos de mark-up semântico não foram aplicados:

```
<font size=7 color=red face=Helvetica>Cabeçalho do Artigo</font>
<br /><br />
<font size=4 color=black face=Arial>&nbsp;&nbsp;&nbsp;Lorem ipsum
dolor sit amet, consectetur adipiscing elit. Sed aliquet
elementum erat. Integer diam mi, venenatis non, cursus a,
hendrerit at, mi.
<br /><br />
&nbsp;&nbsp;&nbsp;Quisque faucibus lorem eget sapien. In urna sem,
vehicula ut, mattis et, venenatis at, velit. Ut sodales lacus
sed eros.</font>
```

Podemos observar que o trecho descreve um título ou cabeçalho de um pequeno texto composto por dois parágrafos. De imediato, podemos perceber que as tags não fazem qualquer sentido, se comparadas ao conteúdo, além da presença de elementos de estilização juntamente com os elementos de marcação do texto, temos ainda elementos para caracteres especiais a fim de realizar o espaçamento de um novo parágrafo, consistindo em novo uso inadequado de elementos para realizar a estilização. A análise de tal código não será consistente para todos os browsers das diversas plataformas disponíveis hoje no mercado, além da dificuldade de identificar precisamente “o quê é o quê” neste código. O mark-up mais adequado para o trecho acima, seria o seguinte:

Cabeçalho do Artigo

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

```
<p> Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Sed aliquet elementum erat. Integer diam mi, venenatis non,  
cursus a, hendrerit at, mi.</p>  
<p> Quisque faucibus lorem eget sapien. In urna sem, vehicula  
ut, mattis et, venenatis at, velit. Ut sodales lacus sed  
eros.</p>
```

Neste caso, o cabeçalho assim como os parágrafos estão com as tags corretas, dispensando o uso de quebras de linhas e caracteres especiais para marcar um novo parágrafo. Isto representa a integridade estrutural da página, garantia de exibição concisa nos diversos browsers e plataformas, além da lógica e semântica na codificação e leitura posterior. Quanto à formatação e estilização, serão tratados por estilos CSS, que irão suprir perfeitamente todas as necessidades de espaçamentos, recuos, cores, fontes, etc.

Utilizando as tags adequadas para cada situação, teremos ainda grande facilidade e versatilidade quando formos estilizar a página com estilos CSS, pois cada elemento poderá ser estilizado individualmente, sem comprometer ou dificultar a estilização do restante da página.

Além dos benefícios já citados, algo muito importante deve ser considerado, que já algum tempo vem ganhando destaque: a análise dos motores de busca. Grandes sites de pesquisa como Google, Yahoo! e Bing, possuem algoritmos de busca muitíssimo sofisticados, que realizam toda a análise não só do conteúdo de um site, mas também da estrutura lógica deste site. Esta estrutura lógica está intimamente ligada com o mark-up semântico. Considerando que temos tags com objetivos específicos e utilizações pré-determinadas, suponhamos que procuro por citações de autores brasileiros famosos. Um site cuja citação está entre as tags <blockquote></blockquote> terá maior relevância do que uma citação que estiver entre simples tags <p></p>. Há alguns anos começaram os estudos de SEO – *Search Engine Optimization* ou Otimização para Motores de Busca, a fim de adequar o conteúdo de páginas da web para que estas tenham maior relevância quando algum de seus conteúdos for pesquisado em um motor de busca. Hoje, empresas já demandam

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

profissionais com conhecimentos específicos nesta área para prestarem serviços a seus contratantes.

Para concluir, a tabela 1 a seguir traz um resumo de tags e seus usos-padrão numa página da web.

Tabela 1 – Resumo de Tags e Utilização Padrão

Tag	Utilização
<h1>, <h2> ... <h6>	Níveis de títulos/cabeçalhos. Não devem ser considerados apenas pelo tamanho padrão do texto exibido, que pode ser alterado e sim pela relevância do título/cabeçalho.
	Enfatizar um trecho de texto.
	Enfatizar fortemente um trecho de texto.
<big> e <small> <big> e <small>	Aumentar e diminuir o tamanho de um texto interno. (O resultado pode variar um pouco para cada browser).
<code>	Denota um exemplo de código. (Normalmente exibido em fonte simples)
<kbd>	Denota um texto inserido pelo usuário. (Normalmente exibido em fonte simples)
<samp>	Indica um exemplo de programação. (Normalmente exibido em fonte simples)
<blockquote>	Utilizado para longas citações. Bastante útil em trabalhos acadêmicos.
<q>	Utilizado em citações breves.
<abbr title="Cascading Style Sheets">CSS</abbr>	Utilizado para abreviações de termos. (Ao colocar o mouse sobre a abreviação, aparece o seu significado).
<acronym title="Programa de Educação Profissional">PEP</acronym>	Utilizada para acrônimos. (Ao colocar o mouse sobre o acrônimo, aparece o seu significado).
	Algo que foi deletado ou não mais é utilizado. O navegador exibe um risco sobre o termo.
<ins>	Algo que foi inserido. O navegador exibe um risco sob o termo.

3) Padrões de Documentos

Apesar de as diferenças entre o XHTML e o HTML serem tênues, um simples documento em branco possui a declaração um pouco mais completa. Embora possam haver variações, será algo muito parecido com o trecho a seguir:

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html;
charset=utf-8" />
    <title></title>
  </head>
  <body>
  </body>
</html>
```

Apesar de esta declaração ser a equivalente de um documento HTML mínimo, há algumas diferenças importantes. A declaração DOCTYPE determina qual o DTD (document type definition) está sendo seguido. Deve ser sempre referenciado em letras maiúsculas, conforme recomendação do W3C. É esta declaração que define qual tipo de documento estará sendo exibido, informando ao browser e ao serviço de validação.

São três as opções de declaração DOCTYPE: XHTML Strict, XHTML Transitional e XHTML Frameset, cada uma com suas peculiaridades. Vejamos os detalhes:

- **xHTML Strict**: Força uma codificação mais rigorosa, já que não permite o uso de elementos de estilização no mark-up e elementos ultrapassados. Contudo, tende a garantir maior compatibilidade com os browsers.

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11-strict.dtd">
```

- xHTML Transitional: De uso bastante comum, permite a utilização de elementos *deprecated*, sendo muitíssimo útil em ocasiões onde há dificuldades de contornar um determinado problema.

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11-transitional.dtd">
```

- xHTML Frameset: GRANNEL (2009) define os frames como uma relíquia, já que raramente são utilizados no desenvolvimento web atual. Mesmo assim, com o intuito de garantir o máximo de compatibilidade para os designers que ainda o utilizam, o W3C especificou um DTD para ele.

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11-frameset.dtd">
```

A tag de início `<htmlxmlns="http://www.w3.org/1999/xhtml">` destina-se a reduzir a ambiguidade dos elementos definidos dentro da página web, onde a declaração de linguagem indica a linguagem padrão usada no conteúdo do documento.

Temos ainda a declaração do tipo de conteúdo, que faz uso de uma tag meta. É aqui também que definimos o conjunto de caracteres utilizado, normalmente utilizando o UTF-8 (Unicode), que é uma codificação padrão e recomendada, que suporta várias linguagens e caracteres. Para nós, brasileiros, uma alternativa é o ISO-8859-1 (Latin 1), cuja declaração é simplesmente "iso-8859-1".

Ainda que pareça complexo, os softwares voltados para desenvolvimento web como Dreamweaver, Aptana Studio/Eclipse, Netbeans, entre outros, já fazem estas declarações iniciais automaticamente ao se criar um novo documento. Portanto, cabe a nós apenas entendê-las e saber a finalidade de cada uma. Alguns desses programas pode inclusive incluir antes do DOCTYPE a declaração do XML, que para fins de desenvolvimento de páginas pode ser excluída, já que browsers mais antigos podem interpretá-la de forma inadequada.

4) Títulos de Páginas

Muitas das vezes tratados em segundo plano, os títulos são elementos importantes em um website. Normalmente aparecem na parte superior da janela do browser e/ou da aba em que está. Por padrão, algumas ferramentas de desenvolvimento adicional na tag destinada ao título “Untitled Document” ou simplesmente a deixam em branco, isto quando a adicionam automaticamente.

Isto poderá acarretar em um documento xHTML inválido, página sem nome ou com nome genérico, além de deixar uma má primeira impressão de uma página sem identidade.

A recomendação é que cada página tenha seu respectivo título, sempre relacionando a empresa ou pessoa proprietária do site, mas um título único ainda que não seja o mais indicado, é aceitável.

A tag de título deve estar presente dentro da tag <head></head>, conforme o exemplo a seguir:

```
<!DOCTYPE html PUBLIC "-//W3C/DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html;
charset=utf-8" />
    <title>Título da Minha Página</title>
  </head>
  <body>
  </body>
</html>
```

5) Meta Tags

As meta-tags são utilizadas pelos motores de busca para indexar informações sobre o documento em questão e categorizar páginas da web. Com a utilização do mark-up semântico, este papel das deixou de ser exclusivo das meta-tags e agora, toda a página é analisada. Contudo, seu uso não deve ser descartado. As meta-tags são declaradas entre tags <head></head>, tal como a tag <title></title>.

Abaixo, as meta-tags mais comumente utilizadas e relevantes serão explicadas.

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

- `<meta name="keywords" content="palavras-chave, separadas, por, vírgula" />`
 - Responsável por conter uma lista de palavras-chave que se relacionem com a página em questão e possivelmente sejam digitas por um usuário num site de buscas.
- `<meta name="description" content="Curta descrição do site" />`
 - Nesta tag, será incluída a descrição do site como um todo, de forma resumida.
- `<meta name="robots" content="all, index" />`
 - Podendo ainda variar o parâmetro 'content' com os valores 'noindex', 'none', 'follow' ou 'nofollow', indicam ações que o motor de busca irá tomar em relação ao seu site.
- `<meta name="Revisit-After" content="30 Days" />`
 - Muito válida para sites de conteúdo dinâmico, indica o período no qual o motor de busca deve retornar para avaliar seu conteúdo.
- `<meta name="Author" content="Seu nome - Sua empresa - etc" />`
 - Não relevante para máquinas de busca, serve para que o autor do site, "assine" seu projeto e deixa uma URL ou e-mail de contato.

6) Favicons

Os "favicons" são aqueles pequenos ícones que vemos na barra de endereços do browser. São incluídos por meio da tag `<link />`, utilizando os atributos `rel`, `href` e `type`. O valor do atributo `type` irá mudar de acordo com o tipo de arquivo utilizado para a imagem. Arquivos de ícone (extensão *.ico), funcionam em todos os browsers, mas nos browsers mais recentes, arquivos PNG (extensão *.png) também são aceitos. A tag deve ser incluída dentro de uma tag `<head></head>`, assim como o título do site e as meta-tags.

Veja abaixo dois exemplos de inclusão de favicon:

```
<link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />  
<link rel="shortcut icon" href="imagens/favicon.png" type="image/png" />
```

Vale lembrar que o parâmetro href deverá apontar para a localização exata de seu favicon, bem como o nome completo do arquivo incluindo a extensão.

UNIDADE II – FUNDAMENTOS DO CSS

1) Introdução

O CSS é o padrão W3C de definição da apresentação visual para as páginas WEB. Surgiram a partir da crescente complexidade e “sujeira” que as tags de estilização agregaram ao HTML. Além disso, os problemas de apresentação e inconsistência com a grande variedade de browsers que surgiam no mercado, fizeram com que essas tags se tornassem indesejáveis e ultrapassadas.

A idéia inicial do CSS era simples: separar o conteúdo da estilização e design. No princípio, houveram grandes dificuldades, primeiramente pela falta de suporte dos browsers e resistência dos designers para esta mudança, já que teriam que trocar métodos já consolidados (apesar de ultrapassados) por uma nova tecnologia. Porém, o W3C investiu nos incentivos aos desenvolvedores, procurando mostrar-lhes os grandes benefícios que a utilização do CSS poderiam proporcionar. Atualmente, não se pode falar de desenvolvimento de websites sem associar a CSS, hoje em sua versão 2.1 e com a versão 3 em desenvolvimento, que promete grandes inovações e sofisticções que irão facilitar ainda mais a vida dos desenvolvedores.

Utilizar Folhas de Estilo Cascadeado (CSS) vai muito além da idéia de “limpar” o código HTML. Envolve também as questões do mark-up semântico, deixar de utilizar tabelas para

organizar o conteúdo, facilitar em futuras manutenção e alteração do layout, lógica do código HTML e indexação do website.

As CSS podem a princípio parecer complicadas nos primeiros contatos, mas após um pouco de prática, pode-se perceber o quão simples são de serem utilizadas, provendo uma maneira fácil de estilizar o conteúdo de páginas da Web. Um documento CSS ou um script CSS é composto de dois elementos principais: os seletores e as propriedades. Nesta Unidade II, iremos estudar os seletores, que representam qual a porção da página da web será estilizada.

2) Adicionando Estilos a uma Página da Web

O método mais usual e recomendado de aplicação de regras CSS para uma página web, é utilizar documentos externos. Neste caso, as regras são definidas num arquivo *.css, que será anexado a um documento HTML. Isto pode ser feito de duas formas:

- Utilizando a tag <link />:
`<link rel="stylesheet" href="meusestilos.css" type="text/css" media="screen" />`

- Utilizar a importação para um elemento style:
`<style type="text/css" media="screen">
/* */
@import url(meusestilos.css);
/*]]> */
</style></code></div><div data-bbox="138 694 862 735" data-label="Text"><p>Ambas incorrerão no mesmo resultado, porém a segunda era mais utilizada devido a alguns problemas de compatibilidade do Internet Explorer.</p></div><div data-bbox="138 742 862 783" data-label="Text"><p>A tag <style></style> também é utilizada para descrição das propriedades no próprio documento HTML, conforme exemplo a seguir:</p></div><div data-bbox="198 790 503 901" data-label="Text"><pre><style type="text/css" media="screen">
/* <![CDATA[*/
seletor {
 propriedade: valor;
}</pre></div><div data-bbox="831 921 862 939" data-label="Page-Footer"><p>11</p></div>`

```
/* ]]> */  
</style>
```

O problema de utilizar desta forma, é que os estilos apenas serão válidos para o documento no qual estão inseridos. Assim sendo, o cascadeamento dos estilos fica mais restrito.

A terceira forma é a aplicação de estilos inline, ou seja, diretamente numa tag xHTML:

```
<p style="propriedade: valor;">Parágrafo estilizado inline.</p>
```

Esta terceira forma, no entanto, é muito restritiva além de manter a poluição do código xHTML, além de ser uma forma arcaica que tende a ser eliminada.

3) Regras e Seletores CSS

As CSS consistem num conjunto de regras que definem como os elementos de uma página serão exibidos. As regras consistem em um seletor e uma declaração, onde o seletor inicia uma regra, especificando qual parte do documento xHTML será estilizado e as propriedades determinam como o elemento selecionado será apresentado. Em um seletor, podemos incluir uma ou mais propriedades e seus respectivos valores.

A sintaxe padrão do CSS é composta por seletor, abertura de chaves, propriedades e respectivos valores e fechamento de chaves. É importante lembrar que antes do valor da propriedade, colocamos dois pontos ":" e ao final de cada linha de comando, fechamos com um ponto-e-vírgula ";", conforme mostra o exemplo a seguir:

```
seletor {  
    propriedade: valor;  
    propriedade2: valor;  
}
```

As quebras de linha não são obrigatórias, mas deixa o código mais legível.

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

Em Folhas de Estilo podemos usar diversos tipos de seletores, variando de acordo com sua aplicação. Nos itens a seguir iremos explicar cada um deles e explicar suas diferenças e características principais.

3.1) Seletores de ID

De acordo com as especificações do W3C, seletores de ID podem ser usados uma única vez no contexto de uma página. No código HTML, você aplica este seletor utilizando o atributo id de um elemento:

```
<div id="rodape"></div>
```

Na sua Folha de Estilos, um seletor de ID será sempre precedido de tralha (#), conforme os exemplos a seguir:

Exemplo 1:	Exemplo 2:
<pre>#meuSeletor{ margin-top: 15px; }</pre>	<pre>div#rodape{ width: 760px; }</pre>

Nos exemplos acima, foram utilizados seletores de id. A diferença entre eles é que, no exemplo 1, para qualquer elemento HTML cujo atributo id for "meu Seletor", a propriedade será aplicada. Já no exemplo 2, apenas elementos <div></div> com o atributo id de valor "rodape" serão estilizados.

Em síntese, declarar o nome da tag antes do seletor, seja ele de ID ou Classe, restringirá a estilização àquela tag.

3.2) Seletores de Classe

Os seletores de Classe nos permitem que um grupo de elementos seja estilizado, já que diferentemente de elementos cujo determinado ID pode ser atribuído a um e somente um elemento por página, vários elementos numa mesma página podem possuir a mesma classe. Para tal, será utilizado o atributo "class" de um elemento HTML. No CSS, utilizaremos um ponto (.) antes do nome do seletor para indicar que refere-se a uma classe.

Veja os exemplos a seguir:

xHTML:

```
<div id="rodape">  
    <p class="destaqueAzul"> Parágrafo com destaque azul. </p>  
    <p class="destaqueVerde"> Parágrafo com destaque verde.</p>  
    <p class="destaqueAzul"> Outro parágrafo com destaque azul. </p>  
    <p class="destaqueVerde"> Outro parágrafo com destaque verde.</p>  
</div>
```

CSS:

```
.destaqueAzul{  
    font-weight: bold;
```

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

```
        color: #00F;  
    }  
  
    .destaqueVerde{  
        font-weight: bold;  
        color: #0F0;  
    }
```

Simples, não é verdade? Contudo, reparem que a propriedade “font-weight” é comum para ambas as classes. Podemos então melhorar nosso código para deixá-lo mais “enxuto”. No atributo “class” de um elemento HTML temos a possibilidade de atribuir duas classes de uma só vez, separando-as por um espaço. Isto é algo perfeitamente aceito pelos Padrões Web do W3C, além de funcional e compatível com os diversos browsers. Assim sendo, poderíamos fazer da seguinte maneira:

xHTML:

```
<div id="rodape">  
    <p class="destaque azul"> Parágrafo com destaque azul. </p>  
    <p class="destaque verde"> Parágrafo com destaque verde.</p>  
    <p class="destaque azul"> Outro parágrafo com destaque azul.  
</p>  
    <p class="destaque verde"> Outro parágrafo com destaque  
verde.</p>  
</div>
```

CSS:

```
.destaque{  
    font-weight: bold;  
}  
  
.azul{  
    color: #00F;  
}  
  
.verde{  
    color: #0F0;  
}
```

Percebam que apesar da necessidade de criar uma nova classe, eliminamos a redundância da propriedade “font-weight” em nosso arquivo CSS. Para este exemplo, no qual utilizamos a criação de nova classe para eliminar a redundância de uma única propriedade, pode parecer desnecessário. Mas considere um arquivo CSS de um site inteiro e 4, 5, 6 ou mais propriedades repetidas. Certamente algumas linhas de código poderão ser eliminadas, deixando o arquivo CSS mais limpo, leve e, conseqüentemente, o carregamento de seu site mais rápido.

A especificação de uma determinada classe para determinado elemento HTML pode ser feito assim como nos seletores de ID, bastando para isso colocar o nome da tag antes do (.) que determina o seletor de classe, como no exemplo abaixo:

```
p.verde{  
    color: #0F0;  
}
```

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

No exemplo acima, a classe “verde” somente irá estilizar elementos de parágrafo, ou seja, a tag <p></p>.

3.3) Seletores de Tags

Seletores de tags são os mais genéricos possíveis. No arquivo CSS é simplesmente escrito o nome da tag e automaticamente, todas as ocorrências daquela tag em suas páginas, terão aquele estilo, a não ser que seja atribuído a ele uma Classe ou ID, que terá precedência sobre o seletor de tag.

Vamos observar o exemplo a seguir:

xHTML:

```
<p class="vermelho"> Este meu parágrafo será escrito na cor vermelha.</p>
```

CSS:

```
p{
    font-weight: bold;
    text-indent: 20px;
    color: #222;
}

.vermelho{
    color: #F00;
}
```

No exemplo acima, o texto de meu parágrafo aparecerá em negrito, com 20 pixels de espaçamento na primeira linha, porém a cor apresentada será vermelha. Isto porque o atributo “color” descrito na classe “.vermelho” com o hexadecimal da cor vermelho puro (#f00), sobreporá o cinza escuro descrito no seletor de tag “p”. E tal procedimento será válido para quaisquer outros parágrafos ou elementos que em seu atributo “class” for colocado o valor “vermelho”.

Em síntese, propriedade de seletores de maior restrição sobrescrevem as propriedades de menor restrição. Assim sendo, propriedades descritas num seletor de ID (#), terão preferência sobre os de classe (.) que por sua vez terão preferência sobre os seletores de tag. Mas observe que isso somente é válido para propriedades CSS idênticas, do contrário, uma mistura de tudo será exibida. Veja no exemplo a seguir:

xHTML:

```
<p id="primeiroParagrafo" class="vermelho"> Meu parágraf.</p>
```

CSS:

```
p{
    font-size: 15px;
    font-family: Arial;
    text-indent: 15px;
}
#primeiroParagrafo{
    margin-top: 15px;
    text-indent: 20px;
}
.vermelho{
    color: #f00;
}
```

A exibição do trecho acima, teria um parágrafo com espaçamento de 20 pixels da margem (text-indent do ID sobrepõe o da tag), com fonte de 15 pixels, margem superior de 15 pixels, da cor vermelha e fonte Arial.

3.4) Seletores Mistos e Seletores Agrupados

A versatilidade do CSS nos permite vários tipos de seleções. De acordo com a forma que são feitas as declarações dos seletores, podemos restringir implicitamente a estilização de alguns elementos de nossa página. Podemos também atribuir determinado estilo para vários elementos de uma só vez, através de agrupamentos.

Vejamos primeiramente um exemplo de seleção mista, onde restringiremos que apenas os parágrafos de um determinado bloco <div></div> seja da cor azul.

xHTML:

```
<div id="cabecalho">
    <p> O texto de meu cabeçalho é azul.</p>
</div>
<div id="corpo">
    <p> O texto do corpo do meu site é cinza escuro.</p>
```


</div>

CSS:

```
#cabecalho{
    width: 760px;
    margin-bottom: 20px;
}
#cabecalho p{
    color: #00f;
}
#corpo{
}
#corpo p{
    color: #222;
}
```

Percebam que bastou que o seletor de tag “p” fosse aplicado imediatamente após o seletor de ID “#cabecalho” para que o conteúdo de todas as tags “p” contidas no meu bloco de cabeçalho fossem exibidos da cor azul. O mesmo raciocínio serve para a análise da exibição em cinza escuro dos parágrafos contidos no bloco cujo atributo id for “corpo”.

Os agrupamentos de seletores, como dito anteriormente, servirão para aplicar determinada propriedade CSS para vários elementos de uma só vez. O exemplo abaixo ilustra o procedimento:

CSS:

```
h1, h2, h3, h4, h5, h6{
    color: #00f;
}
```

Acima, atribuímos a cor azul para todos os parágrafos das páginas onde a folha de estilo estiver anexada. A não ser que uma classe ou id específico cuja propriedade “color” for diferente de azul seja atribuído a um elemento h1, h2, h3, h4, h5 ou h6. As regras de preferência também se aplicam para os casos de agrupamento.

Por fim, podemos juntar os conceitos de mistura e agrupamento de seletores:

xHTML:

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

```
<div id="cabecalho">
    <p> O texto de meu cabeçalho é azul.</p>
</div>
<div id="corpo">
    <p> O texto do corpo do meu site é azul.</p>
</div>
<div id="rodape">
    <p> Texto do rodapé na cor cinza escuro. </p>
```

CSS:

```
#cabecalho{
    width: 760px;
    margin-bottom: 20px;
}
#corpo{
}
#cabecalho p, #corpo p{
    color: #00f;
}
#rodape{
}
#rodape p{
    color: #222;
}
```

No exemplo acima, teremos os parágrafos das <div></div> “cabecalho” e “corpo” na cor azul e os parágrafos da <div></div> “rodape” na cor cinza escuro.

Na próxima unidade, conheceremos as propriedades básicas do CSS, para então podermos começar a montar leiautes completos de websites.

UNIDADE III – Propriedades Básicas do CSS

Conforme visto na última unidade, os seletores determinam quais os elementos serão estilizados. Nesta 3ª unidade, conheceremos as principais propriedades do CSS e como elas funcionam. São as propriedades em si que irão determinar como os elementos serão modificados, tais como sua cor, espaçamento, margens, bordas, etc., todas elas especificadas pelos Padrões do W3C.

Para os valores da propriedade podemos atribuir valores em pixels (px), em, porcentagens (%) ou pontos (pt). Os pontos são utilizados apenas para impressão, enquanto os demais são para exibição em tela. Dentre os três, o mais comum é utilizar pixels (px). As porcentagens normalmente servem para referenciar um valor fixo

anteriormente expresos em px, pt ou em, enquanto os em, são muito pouco utilizados devido a sua complexidade de mensuração.

1) Margin

Conforme o próprio nome sugere, indica o afastamento ou espaçamento de um determinado elemento em relação a seu elemento “pai” ou “irmão”, ou seja, ao elemento no qual estiver contido ou lhe for de mesmo nível hierárquico. As margens podem ser aplicadas juntamente ou separadamente para superior (propriedade margin-top), direita (propriedade margin-right), inferior (propriedade margin-bottom) ou esquerda (propriedade margin-left).

Ao aplicar margens de uma só vez com a propriedade “margin” unicamente, a ordem será superior, direita, inferior e esquerda, onde os valores serão declarados separados por espaço. Podemos também declarar as margens superior/inferior e direita/esquerda, nesta ordem, separando os dois valores com espaços, conforme exemplo abaixo:

```
/* Declaração de todas as margens*/
#cabecalho{
    margin: 15px 5px 15px 5px; /* superior, direita, inferior,
    esquerda */
}
/* Declaração de margens superior/inferior e direita/esquerda*/
#conteudo{
    margin: 15px 5px; /* Mesmo resultado do seletor #cabecalho */
}
/* Declarações separadas*/
#rodape{
    margin-top: 15px;
    margin-right: 5px;
    margin-bottom: 15px;
    margin-left: 5px;
}
```

Dica: Para centralizar um determinado elemento, utilizamos a propriedade “margin” com os valores “0” e “auto”, ou seja, teremos valores nulos para superior/inferior e automáticos (que serão iguais de acordo com a largura do elemento pai) para as laterais, centralizando o elemento em questão. Isto é muitíssimo útil para imagens e tabelas, por exemplo.

2) Padding

A propriedade “padding” é utilizada para preenchimento interno do elemento com espaço vazio. Se temos um elemento cuja sua largura for 150px e aplicarmos a propriedade “padding” para lateral direita ou esquerda no valor de 20px, o elemento passará a ter 170px, onde 20px serão de “enchimento vazio”.

Assim como a propriedade “margin”, a propriedade “padding” pode ser aplicada para todos os lados de uma só vez, para cada lado separadamente (padding-top, padding-right, padding-bottom, padding-left) ou para superior/inferior e esquerda/direita.

O exemplo a seguir ilustra a propriedade “padding”.

```
/* Declaração de preenchimento inferior*/  
div.caixaEsquerda{  
    height: 120px; /* Especificação da altura */  
    padding-bottom: 30px; /*Preenchimento inferior. Altura total:  
150px */  
}
```

3) Border

Declarando a propriedade “border”, criaremos um contorno, ou seja, uma simples borda em nosso elemento. Esta borda, poderá ter determinada espessura (border-width), determinada cor (border-color) e ser de determinado tipo (solid, dashed, dotted, double, groove, ridge, inset ou outset). Pode ainda ser declarada para o componente como um todo ou para cada um de seus lados, tal como as propriedades “margin” e “padding”, além de permitir o tipo, cor e espessura de cada lado separadamente.

Vejamos a seguir dois exemplos de códigos onde as figuras nos auxiliam no entendimento:

```
.caixa{  
    border: 2px #00f solid; /* Uma borda fina, azul e sólida*/  
}
```

```
.caixaDupla{
```

```
Border: 5px #f00 double; /*Razoavelmente grossa, vermelha e  
dupla */  
}
```

Os outros tipos de borda: “dashed” define linha formada por pequenos traços, “dotted” define linha formada por pontos, e groove, ridge, inset ou outset definem estilos de borda 3D que irão variar de acordo com a cor aplicada à borda.

4) Background

Com a propriedade “background”, poderemos adicionar um fundo para qualquer elemento de nossas páginas HTML. O “background” é muito utilizado para colocarmos efeitos em menus ou quando algum elemento é sobreposto com a seta do mouse.

A propriedade “background” é bastante versátil, permitindo que determinemos a cor, imagem, eixo de repetição da imagem (horizontal, vertical, ambos ou nenhum), fixação da imagem e posicionamento da imagem. Isto pode ser feito de uma só vez utilizando a propriedade “background” ou separadamente, utilizando “background-color”, “background-image”, “background-repeat”, “background-attachment” e “background-position”.

Vejamos alguns exemplos do comportamento da propriedade “background” e suas derivadas:

```
/* Fundo de todo meu site */  
body{  
    background: #222 url(imagens/meufundo.png) no-repeat fixed top  
center;  
}  
/* Fundo da minha div de conteúdo */  
div#conteudo{  
    background-image: url(imagens/degradePrincipal.png);  
    background-repeat: repeat-x;  
    background-position: top left;  
}
```

Vamos agora entender cada um dos casos.

Para o fundo de meu site, aplicaremos o “background” na tag <body></body>, conforme o seletor de tags “body” fez. Neste caso, meu fundo terá como cor o cinza escuro, uma determinada imagem, que não se repetirá, será fixa, colada ao topo e centralizada.

Quanto à <div id=”conteudo”></div>, terá uma imagem de fundo que se repetirá infinitas vezes (até atingir o fim da div) no eixo “x” (horizontalmente), colada ao topo e colada à esquerda.

Dica:Caso esteja utilizando um fundo para seu website e deseja que este fundo não se movimento quando rolar a página do site e esteja sempre presente, deverá declarar o valor “fixed” para a propriedade “background” (caso esteja utilizando declaração única) ou para a propriedade “background-attachment”.

5) Color

Uma das mais simples, a propriedade color simplesmente determina a cor do texto que estiver contido em determinada tag, tal como ilustra o próximo exemplo:

```
p{
color: #f00;
}
```

No exemplo acima, os textos dos parágrafos de meu site serão exibidos na cor vermelha.

6) Text e derivadas

As propriedades “text” são utilizadas para formatação dos textos de uma página na web. Poderão ser estilizados o alinhamento, decoração e indentação (espaçamento do parágrafo) com estas propriedades.

Para o alinhamento de nosso texto, utilizaremos a propriedade “text-align”, que por padrão vem com o valor “left” (alinhado à esquerda), o qual nem precisa ser especificado. Para centralizar, justificar e alinhar à direita, utilizaremos, respectivamente, “center”, “justify” e “right”, conforme no exemplo a seguir:

```
p{
text-align: justify;
}
p.centralizado{
text-align: center;
```

```
}  
p.direita{  
text-align: right;  
}
```

A decoração envolve efeitos básicos como “underline” (linha acima do texto), “line-through” (linha cortando o texto), “underline” (linha sob o texto) e “blink” (texto piscando), sendo que este último funciona apenas nos navegadores Firefox e Opera.

Interessante que apenas com esta propriedade podemos criar um simples efeito para que links (<a>) não mostrem o sublinhado quando o mouse estiver sobre ele, conforme o exemplo abaixo:

```
/* Declaração dos estilos para link */  
a{  
color: #0f0;  
text-decoration: underline;  
}  
a:hover{  
text-decoration: none;  
}
```

Neste último exemplo, fizemos uso da pseudo-classe “:hover”, que indica a aplicação das determinadas propriedades quando o cursor do mouse estiver sobre o elemento. Concluímos então que, quando o cursor do mouse estiver sobre um link, a propriedade “text-decoration” receberá o valor “none”, ou seja, nenhum, que fará a linha sob o texto do link desaparecer.

Por fim, a propriedade “text-indent”, que indica qual o tamanho do espaço deixado em branco pela primeira linha de um parágrafo, para que a exibição em tela seja algo similar à deste mesmo parágrafo que você está lendo. Para esta propriedade é especificado um determinado valor que corresponderá ao espaço.

```
p{  
text-indent: 20px;  
}
```

Lembrete: Não existe a propriedade “text-color”. Lembre-se que para modificar a cor da escrita de determinado elemento, é utilizada a propriedade “color”, abordada no item 5 desta unidade.

7) Font e derivadas

A propriedade “font” e suas derivadas estão relacionadas com atributos próprios da escrita, tais como: família da letra e letra específica, tamanho, estilo, espessura e variantes.

Dentre elas, a que merece algum cuidado é a propriedade “font-family”, responsável por especificar a fonte utilizada ou a família de fontes. Devemos nos atentar pois se utilizarmos uma fonte que não esteja presente em todos os computadores, o trecho escrito com ela poderá não aparecer adequadamente para todos os usuários. Por isso, é importante utilizarmos fontes comuns a todos os computadores.

A Tabela 2 a seguir, nos mostra a família genérica e fontes específicas com as quais não teremos problemas para trabalhar nos diversos browsers e plataformas.

Tabela 2. Famílias Genéricas de Fontes e Fontes Específicas	
Família Genérica	Fonte Específica
Serif	Times New Roman
	Georgia
	Arial
Sans-serif	Verdana
	Tahoma
Monospace	Courier New
	Lucida Console

Fonte: W3C <http://www.w3schools.com/css/css_font.asp>

Baseando-se na Tabela 2, podemos exemplificar a declaração da propriedade “font-family” da seguinte maneira:

```
p{
font-family: Serif, "Georgia", "Times New Roman";
}
```

Desta forma, estaremos garantindo que uma fonte com serifa (aquele pequeno acabamento nas extremidades de cada letra) seja exibida para o usuário, pois mesmo que o sistema não possua a fonte Times New Roman ou Georgia, a fonte padrão com serifa será utilizada para exibir o texto.

Para alterarmos o tamanho do texto, a propriedade utilizada é “font-size”. Para esta propriedade, podemos atribuir um valor inteiro de px, em, % ou pt, além de

valores literais como “small”, “large”, “medium”, “larger”, entre outros. Entretanto, a utilização dos valores em pixels (px) ou pontos (pt) é mais comum.

```
p{
font-family: Serif, "Georgia", "Times New Roman";
font-size: 12px;
}
```

Quanto aos estilos da fonte, cuja propriedade CSS é “font-style”, podemos atribuir os valores normal (padrão, que não precisa ser declarado inicialmente), “italic” ou “oblique”. O valor “italic” coloca a fonte em itálico convencional, enquanto “oblique” reproduz um itálico mais acentuado.

```
p{
font-family: Serif, "Georgia", "Times New Roman";
font-size: 12px;
font-style: oblique;
}
```

Para a espessura da fonte, é utilizada a propriedade “font-weight”. Em outras palavras, definirá o quão negrito ficará a escrita. Os valores mais comuns são “bold”, “bolder” e “lighter”, mas valores numéricos como 100, 200, 300 ... 900 também são aceitos. Por padrão, o valor desta propriedade é “normal”.

```
p{
font-family: Serif, "Georgia", "Times New Roman";
font-size: 12px;
font-style: oblique;
font-weight: bolder;
}
```

A última derivada da propriedade font, a “font-variant” está relacionada com a exibição do texto com maiúsculas e minúsculas - valor “normal” - ou com maiúsculas e pequenas maiúsculas – valor “small-caps”. Um texto em “small-caps” fica algo como MEU TEXTO EM SMALL-CAPS.

```
p{
font-family: Serif, "Georgia", "Times New Roman";
font-size: 12px;
font-style: oblique;
font-weight: bolder;
font-variant: small-caps;
}
```

}

Lembrete: Todos as propriedades derivadas de “font” podem ser declaradas de uma só vez na própria propriedade “font”.

8) Float

Propriedade bastante utilizada para posicionamento de várias imagens ou na confecção de leiautes, o “float” (flutuação) irá posicionar o elemento horizontalmente na extrema esquerda ou extrema direita, permitindo que outros elementos fiquem a seu redor. Para eliminar a flutuação dos elementos vindouros, utilizamos a propriedade “clear” com o valor “both”.

```
img{  
    float: left;  
}
```

O estilo acima define que todas as imagens irão ser posicionadas na extrema esquerda de seus contêineres. Caso várias imagens sejam colocadas em seguida, irão ser posicionadas lado-a-lado.

Ao desenvolver o leiaute de um site, muitas vezes as flutuações poderão ter comportamentos inesperados pelos mais diversos motivos. Uma forma de corrigir é modificando a forma de exibição, com a propriedade “display” que será mostrada a seguir.

9) Display

A propriedade display é muito versátil e pode ter inúmeras aplicações, desde a “eliminação” de um elemento da tela, até a exibição em linha reta ou como tabela de grupos de elementos, sendo o comportamento determinado apenas por seu valor.

Podemos inclusive determinar que determinados elementos se comportem como blocos. Um exemplo clássico é o da tag . Por padrão, a tag span possui suas medidas de largura e altura nativamente automáticos, variando de acordo com seu conteúdo. Porém, atribuindo o valor “block” para a propriedade “display” de um , faremos com que ele se comporte como um elemento de bloco, tal

como um <div></div>, podendo inclusive determinar seu tamanho. Veja no exemplo a seguir.

```
span.bloco{
    display: block;
    width: 200px;
    height: 200px;
    border: 1px solid #000;
}
```

Quanto a ocultação de elementos, a propriedade “display” deverá receber o valor “none”. Porém, o espaço outrora ocupado por este elemento que não mais é exibido, será ocupado pelo próximo elemento, como se uma força física arrastasse o elemento a seguir para ocupar o espaço deixado por aquele que não mais é exibido.

Dica: Para ocultar um elemento preservando o espaço que ele ocupa, devemos utilizar a propriedade “visibility”, atribuindo a ela o “hidden”.

Um outro artifício onde a propriedade “display” é muito utilizada, é na montagem de menus de navegação. É muito comum a utilização de listas não-ordenadas () para a estilização de menus. Todavia, estas listas normalmente se comportam da seguinte maneira:

Código xHTML:	Exibição padrão:
<pre> Opção 1 Opção 2 Opção 3 </pre>	<ul style="list-style-type: none">• Opção 1• Opção 2• Opção 3

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

Desta maneira, além de aparecerem os marcadores da lista (●), a lista ainda aparece na vertical. Se eu quiser um menu horizontal, não ficarei satisfeito. Assim sendo, podemos fazer o seguinte:

CSS:	xHTML:	Exibição Estilizada:
<pre>/* Retiramos os marcadores*/ ul{ list-style: none; } /* Exibimos em linha. */ ul li{ display: inline; margin-right: 3px; }</pre>	<pre> Opção 1 Opção 2 Opção 3 </pre>	Opção 1Opção 2Opção 3

Dependendo de sua necessidade, um dos valores para a propriedade “display” irá se encaixar e resolver algum comportamento inesperado de seu leiaute. Você deverá então testar para verificar aquele que melhor se adequar. Normalmente, para problemas com leiaute, os valores “inline-block” ou “inline-table” resolvem grande parte das incompatibilidades.

No site do W3C Schools encontra-se uma relação de todos os possíveis valores para a propriedade display no link http://www.w3schools.com/css/pr_class_display.asp.

UNIDADE IV – FUNDAMENTOS DO JAVASCRIPT

1) Introdução

O Javascript é uma linguagem de scripting desenvolvida por Brendan Eich na Netscape em 1995, sendo inicialmente embarcado no Netscape Navigator 2.0. Seu objetivo é dar vida à páginas HTML que outrora eram estáticas, permitindo maior interatividade com o usuário e resposta a eventos.

É uma linguagem *cliente-side*, ou seja, desenvolvida para ser executada no navegador do cliente e não em um servidor. Sua sintaxe é similar ao C, Perl e Java, sendo ainda uma linguagem interpretada (não necessita de compilação).

Nos últimos tempos vem se tornando uma das linguagens mais populares e utilizadas no desenvolvimento web, com a diferença de que não mais é utilizada para prover simples interações com o usuário, mas também para auxiliar no desenvolvimento de sistemas que rodam na plataforma web.

É importante ressaltar que Javascript não é Java e são linguagens totalmente distintas. Aplicações Java são compiladas, scripts Javascript são interpretados por um browser; Java é fortemente tipado e restrito, Javascript é fracamente tipado e flexível; tipos de variáveis necessitam ser explicitamente declarados em Java, o que não ocorre no Javascript.

Ainda citando importantes detalhes do Javascript, lembramos que Javascript não é HTML, apesar de poder ser embutido em uma página HTML; não é utilizado para ler ou gravar arquivos na máquina do usuário (com exceção de Cookies); modificadores de acesso – bastante utilizados em orientação a objetos – não se aplicam para a sintaxe do Javascript.

Passando agora para a sintaxe de código Javascript, alguns aspectos devem ser reforçados:

- Javascript é *case-sensitive*, ou seja, “minhacor” é diferente de “minhaCor”;
- Todas as linhas de código devem ser finalizadas com ponto-e-vírgula (;);

- Comentários pode ser feitos em linha utilizando “//” ou em bloco (múltiplas linhas) utilizando “/*” para marcar o início e “*/” para marcar o fim do bloco;
- Indentação do código deve sempre ser aplicada;
- Funções, laços de repetição e estruturas condicionais normalmente terão seus blocos de código demarcados com “{ }”;

A codificação Javascript é bem simples e podemos escrever “Olá Mundo” facilmente da seguinte maneira:

```
document.write(“Olá Mundo!”);  
alert(“Olá Mundo”);
```

No primeiro caso, a mensagem irá aparecer no contexto da página HTML. No segundo caso, uma janela irá aparecer contendo a mensagem.

Nos tópicos a seguir, iremos abordar as funcionalidades básicas da programação com Javascript e conhecer alguns de seus recursos.

2) Arquivos Javascript

Arquivos Javascript são arquivos comuns de texto com extensão “*.js”. Neste arquivo, podemos colocar todo nosso código, contendo declaração de variáveis, funções, resposta a eventos, etc. Quando utilizamos um Ambiente de Desenvolvimento Integrado ou IDE, tal como o Netbeans, ao criarmos um arquivo Javascript, o Ambiente nos fornece alguns recursos que facilitam o trabalho de codificação, onde podemos destacar:

- Colorização e destaque de palavras: palavras reservadas, funções, variáveis, laços de repetição e etc, possuem cores e estilos diferenciados, que facilitam a leitura de nosso código;
- Marcação de erros de sintaxe;
- Auxílio de codificação: ao começarmos a digitar o nome de um comando, uma lista de possíveis complementos para o que está sendo digitado, bem como informações sobre a funcionalidade de determinada função são exibidos

Após criarmos, editarmos e salvarmos um arquivo Javascript, este deverá ser anexado às páginas HTML, tal como com arquivos CSS. Para tal, utilizamos o seguinte código, preferencialmente dentro da tag <head></head>:

```
<script language="Javascript" src="pasta/arquivo.js"></script>
```

Determinando a linguagem (language) e a localização de nosso arquivo (src), todo nosso código já foi adicionado à página e poderemos agora utilizar as funções e recursos que codificamos.

3) Tag <script></script>

A tag script além de servir para importarmos um arquivo Javascript para as páginas, pode também exercer a função de contêiner de código. Você pode criar funções dentro de uma tag <script></script> e fazer uso delas na página em que foram criadas – e somente nesta página.

Veja no exemplo abaixo:

```
<script language="Javascript">  
    document.write("Olá Mundo!");  
</script>
```

Isto é aconselhado apenas quando funções servirão apenas para determinada página, preferencialmente quando não forem muito extensas, afim de manter o código HTML limpo e puro.

Vale ressaltar que os benefícios da codificação dentro de tags <script></script> providos por um IDE podem variar entre um e outro software.

4) Variáveis

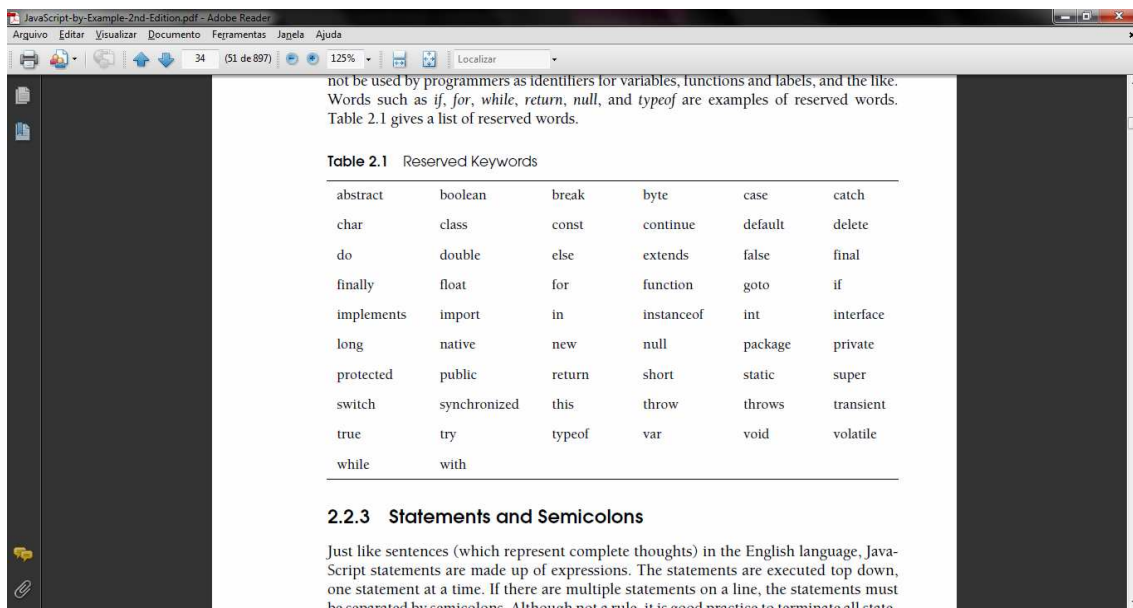
As variáveis são dados que podem ser alterados em tempo de execução. Para sua a declaração de seus nomes, algumas regras devem ser obedecidas:

- Todos identificadores devem começar com letra ou com o caracter (_);

- Todos os caracteres seguintes podem incluir também os dígitos de 0 a 9;
- As letras permitidas vão de A a Z, maiúsculas ou minúsculas;
- Nenhuma palavra chave reservada do Javascript pode ser usada;

A tabela 3, a seguir, traz as palavras-chave ou reservadas do Javascript:

Tabela 3 – Palavras-chave ou Reservadas do Javascript



The screenshot shows a PDF document titled 'JavaScript-by-Example-2nd-Edition.pdf' in Adobe Reader. The document contains a table of reserved keywords and a section on statements and semicolons.

not be used by programmers as identifiers for variables, functions and labels, and the like. Words such as *if*, *for*, *while*, *return*, *null*, and *typeof* are examples of reserved words. Table 2.1 gives a list of reserved words.

Table 2.1 Reserved Keywords

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	delete
do	double	else	extends	false	final
finally	float	for	function	goto	if
implements	import	in	instanceof	int	interface
long	native	new	null	package	private
protected	public	return	short	static	super
switch	synchronized	this	throw	throws	transient
true	try	typeof	var	void	volatile
while	with				

2.2.3 Statements and Semicolons

Just like sentences (which represent complete thoughts) in the English language, JavaScript statements are made up of expressions. The statements are executed top down, one statement at a time. If there are multiple statements on a line, the statements must be separated by semicolons. Although not a rule, it is good practice to terminate all state-

Como mencionado anteriormente, variáveis Javascript não necessitam ser explicitamente declaradas. Assim sendo, podemos declarar variáveis facilmente das seguintes maneiras:

```
var carro = "Civic"; // String ou caractere
var numero = 3; //Numérico
var administrador = true; //Booleano ou lógico
var diasFeira = ["Segunda", "Terça", "Quarta", "Quinta", "Sexta"];
//Array ou vetor
```

Dica:A contagem dos índices de um vetor sempre se inicia no índice 0. Desta forma, considerando o exemplo acima, para imprimirmos "Segunda", a sintaxe seria: `document.write(diasFeira[0]);`

5) Operadores

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

Os operadores Javascript, assim como em outras linguagens, são utilizados para comparar, atribuir ou concatenar valores. As tabelas 4, 5, 6 e 7 a seguir resumem suas sintaxes e aplicações.

Tabela 4 – Operadores Aritméticos

Operador	Descrição	Exemplo	Resultado
+	Adição	$x=y+2$	$x=7$
-	Subtração	$x=y-2$	$x=3$
*	Multiplicação	$x=y*2$	$x=10$
/	Divisão	$x=y/2$	$x=2.5$
%	Módulo da divisão (resto)	$x=y\%2$	$x=1$
++	Incremento simples	$x=++y$	$x=6$
--	Decremento simples	$x=--y$	$x=4$

* Dado $y=5$

** Pode ser utilizado para juntar strings: "a" + "b" resultará na exibição de "ab"

Tabela 5 – Operadores de Atribuição

Operador	Exemplo	É o mesmo que...	Resultado
=	$x=y$		$x=5$
+=	$x+=y$	$x=x+y$	$x=15$
-=	$x-=y$	$x=x-y$	$x=5$
=	$x=y$	$x=x*y$	$x=50$
/=	$x/=y$	$x=x/y$	$x=2$

* Dados $x=10$ e $y=5$

Tabela 6 – Operadores de Comparação

Operador	Descrição	Exemplo
==	é igual a	$x==8$ retorna false
===	é exatamente igual a (considera valor e tipo de variável)	$x===5$ retorna true $x===5$ retorna false
!=	não é igual a (diferente de)	$x!=8$ retorna true
>	é maior que	$x>8$ retorna false
<	é menor que	$x<8$ retorna true
>=	é maior que ou igual a	$x>=8$ retorna false
<=	é menor que ou igual a	$x<=8$ retorna true

* Dado $x=5$

Tabela 7 – Operadores Lógicos

Operador	Descrição	Exemplo
&&	E (and)	(x < 10 && y > 1) retorna true
	OU (or)	(x==5 y==5) retorna false
!	Não (not)	!(x==y) retorna true

* Dados x=6 e y=3

Estes operadores serão aplicados para as mais diversificadas situações em sua codificação. Se houverem dúvidas em sua aplicação ou funcionalidade, recorra às tabelas-resumo.

6) Estruturas Condicionais

Estruturas condicionais são utilizadas a todo o tempo para fazer as mais diversificadas verificações, sejam de ordem de grandeza ou lógica. O Javascript, faz uso das estruturas condicionais consagradas nas linguagens de programação: *if*, *else if* e *else*.

Vamos considerar a seguinte situação: você deseja saber se determinado número é maior que outro. Veja como fica:

```
var n1 = 20;
var n2 = 10;
if (n1 > n2){
    document.write("O número "+n1+" é maior que "+n2);
}
```

Para este exemplo, onde a variável n1 recebeu o valor 20 e n2 recebeu o valor 10, a afirmativa foi verdadeira; então o conteúdo da estrutura foi executado, que para esta situação, exibiu uma mensagem para o usuário.

Mas e se estes valores fossem trocados? O usuário também precisaria ser informado. Para isto, utilizaremos a seguinte estrutura:

```
var n1 = 10;
var n2 = 20;
if (n1 > n2){
    document.write("O número "+n1+" é maior que "+n2);
} else {
    document.write("O número "+n1+" é menor que "+n2);
}
```

```
}
```

Ou seja, se o valor da variável n1 for maior que n2, ele exibe a mensagem dizendo que o número é maior; senão, exibirá uma mensagem que diz que o número é menor.

Contudo, poderíamos antes de decretar o veredito final (se um número é maior ou menor que o outro), testar se um número é igual ao outro. Isto é feito da seguinte maneira:

```
var n1 = 10;
var n2 = 20;
if (n1 > n2){
    document.write("O número "+n1+" é maior que "+n2);
} else if (n1==n2){
    document.write("O número "+n1+" é igual a "+n2);
} else {
    document.write("O número "+n1+" é menor que "+n2);
}
```

Agora sim: testamos se o número é maior que o outro; senão, se ele é igual ao outro e por fim, só resta ele ser menor, algo que é óbvio, pois se um número não é maior, nem é igual, ele somente pode ser menor que o outro.

Estas estruturas de condicionais poderão ser aplicadas utilizando todos os operadores das tabelas 6 e 7 desta apostila, com todos os tipos de variáveis.

Atenção:As estruturas condicionais não fazem comparativos entre um array (vetor) e outro. Apenas podem ser testados, ainda assim, separadamente, os **elementos** de um e outro array ou de um mesmo array.

7) Comando Switch

O switch é um comando de escolha que permite a execução de um determinado bloco de comandos de acordo o valor de uma determinada variável.

Veja no exemplo genérico a seguir:

```
switch (variavel){
    case valor1:
        //bloco de comandos...
        break;
    case valor2:
        //bloco de comandos...
        break;
    case valor-n:
```

```
        //bloco de comandos...
        break;
default:
    //bloco de comandos...
    break;
}
```

Podemos assumir que, para cada valor que a variável assumir, um diferente bloco de códigos será executado.

Vejamos no exemplo prático abaixo:

```
var diaSemana = 3;
switch (diaSemana){
    case 1:
        alert("Segunda-feira");
        break;
    case 2:
        alert("Terça-feira");
        break;
    case 3:
        alert("Quarta-feira");
        break;
    case 4:
        alert("Quinta-feira");
        break;
    case 5:
        alert("Sexta-feira");
        break;
    default:
        alert("Final de semana!");
        break;
}
```

Neste caso, o código executado será o do “case 3”, pois 3 é o valor da variável. Caso o valor fosse qualquer um diferente de 1, 2, 3, 4 ou 5, o bloco executado seria o “default”. Este é executado sempre que nenhum dos valores testados for igual ao da variável em questão. Variáveis de conteúdo tipo string também podem ser testadas.

Atenção: Jamais se esqueça de terminar o bloco de códigos de um “case” com um “break;”. Outrossim, o bloco de códigos do “case” seguinte será executado.

8) Laços de Repetição

Os laços de repetição são estruturas de códigos utilizadas para executar determinado bloco de códigos “n” vezes até que determinada condição de parada alcançada. Em Javascript três laços essenciais devem ser destacados: *while*, *for* e *do..while*.

Vejamos as particularidades de cada um deles separadamente.

8.1) While

O comando *while* executa um bloco de códigos enquanto sua condição for verdadeira (*true*). A partir do momento que a condição for *false*, o laço para, conforme mostra o exemplo a seguir:

```
var cont=0;
while (cont<3){
    alert("Cont vale "+cont);
    // Outros códigos...
    cont++; //Incremento da variável. MUITO importante!
}
```

No exemplo acima, enquanto (*while*) a variável *cont* for menor que três, será exibido uma janela pop-up com o valor da variável “*cont*”. Em seguida, a variável *cont* receberá um incremento simples. Este passo é imprescindível para o correto funcionamento do laço, pois sem ele, o código entraria em loop infinito.

Vale mencionar que a variável de controle (a que é colocada entre parênteses e irá determinar o fim do laço), deve ser sempre um número inteiro, visto que uma string ou boolean não poderá ser incrementada.

8.2) For

O laço de repetição *for* é bastante útil quando você sabe exatamente ou aproximadamente quantas vezes aquele código deverá ser executado.

Vamos analisar o exemplo a seguir:

```
var i=0;
var cont=0;
for (i=0; i<=5; i++){
    if (i%2==0){
        alert("O número "+i+" é par.");
        cont++;
    }
}
```

```
}  
alert (cont);
```

O exemplo a seguir testa se todos os números entre 0 e 5, incluindo o próprio 5 são pares, e a cada número par encontrado, a variável “cont” é incrementada, realizando a contagem dos números. Ao final do laço, é exibida a contagem dos números pares.

Outro uso bastante comum, é para preencher arrays:

```
var vetor=new Array(3);  
var i=0;  
for (i=0; i<3; i++){  
    vetor[i]=prompt("Digite um número: ");  
}
```

Perceba que o código contido no laço será executado 3 vezes: vez 0, vez 1 e vez 2. Lembrando: arrays em Javascript iniciam a contagem dos índices no número 0. Assim, se incluíssemos o 3 na contagem, estaríamos buscando por um quarto índice, de valor 3 que não existe.

8.3) Do..while

O laço do..while, executará determinado bloco de código enquanto o teste da variável de controle retornar verdadeiro, tal como no laço while. A diferença, é que este teste será executado apenas no final do laço, o que garante que o laço será executado pelo menos uma vez.

Observe o exemplo:

```
var numero=3;  
do{  
    alert (numero+" é maior que 2, mas mesmo assim o laço foi executado!");  
    numero++; // Não se esqueça do incremento!!!  
}while (numero<2);
```

9) Eventos

Conhecer os eventos é imprescindível no aprendizado de Javascript. Serão eles os grandes responsáveis pela interação com os usuários, pois determinam ações, em sua maioria, realizadas pelo usuário em um determinado elemento do código HTML.

A idéia é simples: ao acontecer determinado evento, uma função Javascript é chamada (mais comum) ou determinado código Javascript é executado.

O exemplo a seguir nos mostra como funciona:

```
<p onmouseover="alert('Mouse em cima!');"> Passe o mouse! </p>
```

Neste caso, quando o cursor do mouse for posicionado em cima deste parágrafo, surgirá uma caixa de alerta com a mensagem “Mouse em cima”.

São vários os eventos para diversas necessidades. A Tabela 8 abaixo resume os eventos e seus disparadores.

Tabela 8 – Eventos e Disparadores

Evento	Disparo
onAbort	Interrupção do carregamento de imagem
onBlur	Elemento perde o foco
onChange	O valor do elemento de um form foi alterado
onClick	O usuário clicou com o botão do mouse
onError	Houve um erro quando uma imagem era carregada
onFocus	Elemento recebeu foco
onLoad	O documento terminou o carregamento
onMouseOut	O mouse foi retirado de cima do elemento
onMouseOver	O mouse foi posicionado em cima do elemento
onSubmit	O usuário enviou um formulário
onUnLoad	O usuário saiu da janela

10) Objetos de Núcleo do Javascript

O Javascript possui alguns objetos pré-definidos em seu núcleo, objetos estes que contém funcionalidades muito úteis em nosso dia-a-dia. Os mais comumente utilizados são Date() e Math().

Vejamo-los separadamente.

10.1) Date

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

O objeto `Date()` nos permite criar as mais diversificadas formas de datas que puder imaginar, bastando que informemos o formato desejado, conforme os exemplos a seguir:

```
var minhaData = new Date("Month dd, yyyy hh:mm:ss");  
var minhaData2 = new Date("Month dd, yyyy");  
var minhaData3 = new Date(yy, mm, dd, hh, mm, ss);  
var minhaData4 = new Date(yy, mm, dd);  
var minhaData5 = new Date(milliseconds);
```

Perceba que cada formato passado, representa uma forma na qual a data atual será atribuída a uma variável. Utilizando apenas `Date()`, o formato completo será aplicado.

10.2) Math

O objeto `Math()` nos traz várias constantes matemáticas métodos específicos muito úteis em nosso dia-a-dia.

Iremos apresenta-las nas tabelas 9 e 10 a seguir:

Tabela 9 – Constantes

Constante Javascript	Significado
Math.E	Constante matemática E
Math.PI	Constante matemática pi
Math.SQRT2	Valor da raiz quadrada de 2
Math.SQRT1_2	Valor da raiz quadrada de ½
Math.LN2	Logaritmo natural de 2
Math.LN10	Logaritmo natural de 10
Math.LOG2E	Logartimo de E na base 2
Math.LOG10E	Logartimo de E na base 10

Tabela 10 – Métodos

Método	Funcionalidade	Exemplo
Math.round()	Arredondamento de número decimal	Math.random(4.8), retorna 5
Math.random()	Geração de número aleatório (randômico)	Alert(Math.random()), retorna um número qualquer entre 0 e 1
Math.floor()	Utilizado em conjunto com Math.random() para gerar números aleatórios	Math.floor(Math.random()*11), retorna um número aleatório entre 0 e 10.

11) Funções

Funções são blocos de códigos que executam determinada tarefa e devolvem algum retorno, ainda que este retorno seja nulo.

Um função é composta pela palavra-chave “function”, o seu nome e parâmetros (valores passados para ela).

No Javascript, uma declaração de função pode ser feita da seguinte maneira:

```
function nomeDaFuncao (parâmetros){  
    // códigos da minha função  
}
```

Assim sendo, podemos escrever uma função que exibe a soma entre dois números e os exibe em um alerta da seguinte maneira:

```
function realizaSoma (n1, n2){  
    alert(n1+n2);  
}
```

E para acessá-la, basta escrever seu nome e passar os devidos parâmetros:

```
var numero = 15;  
var numero2 = 20;  
realizaSoma (numero, numero2);
```

Podemos ainda simplesmente atribuir o valor desta soma, sem exibi-la para uma terceira variável:

```
function realizaSoma (n1, n2){  
    return n1+n2;  
}  
  
var numero = 15;  
var numero2 = 20;  
var soma = realizaSoma (numero, numero2);
```

A palavra chave “return”, devolve um determinado valor obtido por uma função para o código de onde ela foi chamada, podendo este valor ser utilizado posteriormente, se atribuído a uma variável ou a chamada função ter sido colocada estrategicamente dentro de outra função, algo perfeitamente possível e permitido.

COLÉGIO PIAU – Unidade de Leopoldina
Educação Profissional Técnica de Nível Médio

É claro que esta é uma função bastante simples, que serve apenas para exemplificar o funcionamento. Funções muito mais elaboradas serão desenvolvidas por você para fins muito mais específicos. Treine sua lógica de programação fazendo exercícios para exercitá-la.