

1- SOBRE A WEB

História

Formatar informações dos sites não é algo novo. Por volta de 1970, no começo da trajetória do Padrão Generalizado de Linguagens de Marcação, SGML, vários browsers já personalizavam as aparências dos documentos, cada um com seu estilo próprio.

O estudante Hakon Wium Lie percebe as dificuldades de formatação ao desenvolver um site e resolve criar uma maneira fácil para formatar a informação do HTML. Foi aí que ele propôs a criação do CSS ou Cascading Style Sheets em 1994.

Como a web não havia sido projetada para desenvolver os criativos ambientes gráficos que temos atualmente os recursos de desenvolvimento eram limitados e os criadores faziam das tripas coração para criar seus sites. Entre as muitas ideias que surgiram para ultrapassar ao ambiente de "apenas texto" da internet, estava aquela de utilizar tabelas de HTML para posicionar os elementos no layout, utilizando slices de imagem, gifs transparentes e a técnica de aninhamento de tabelas para contornar os problemas que os padrões proprietários traziam. A esse tipo de técnica, que foi usada pela maior parte dos websites, chamamos de layout com tabelas.

W3C – World Wide Web Consortium

Em agosto de 1994 foi fundada a W3C, um consórcio formado por instituições comerciais e educacionais, com o objetivo de definir padrões para as respectivas áreas relacionadas à Web. Em dezembro de 1997 a W3C publicou o HTML 4 e após dois meses publicou o XML 1. A W3C reformulou o grupo responsável pelo HTML para criar uma "suíte de tags XML". O primeiro passo foi dado em dezembro de 1998 quando o grupo reescreveu o HTML em XML sem adicionar elementos ou atributos novos. Essa especificação foi chamada de XHTML1.

O objetivo seguinte da W3C foi a reestruturação dos formulários Web. Em agosto de 1999 o mesmo grupo responsável pelo HTML publicou o primeiro rascunho da extensão dos formulários para XHTML. Alguns meses depois, essa "extensão dos formulários para XHTML" foi rebatizada de "XForms". Foi criado um grupo específico, responsável pelo XForms. Esse grupo trabalhou em paralelo com o grupo de desenvolvimento do HTML com a finalidade de publicar a primeira versão do XForms em outubro de 2003.

Enquanto isso, com a transição do XML completa, os membros do grupo do HTML criaram "a nova geração do HTML". Em maio de 2001 eles publicaram a primeira versão do XHTML1.1.

Em julho de 2004 a W3C organizou um workshop onde estavam presentes membros da W3C e companhias desenvolvedoras de navegadores como a fundação Mozilla e a Opera. Neste workshop foi apresentada uma visão do futuro da web com uma evolução do padrão HTML4 incluindo novas características para aplicações modernas.

No workshop a W3C deixou claro que não seriam desenvolvidas novas tecnologias de extensões do HTML e CSS que não fossem criadas pelo atual grupo de trabalho da W3C. Diante desse impasse, o grupo que desenvolvia o HTML e os formulários HTML tinha duas escolhas: ou se uniam à W3C, ou se separavam. Dessa forma, em julho de 2004 surgiu o WHATWG.

De acordo com o W3C a Web é baseada em três pilares:

1. Um esquema de nomes para localização de fontes de informação na Web, a URI.
2. Um Protocolo de acesso para acessar estas fontes, o HTTP.

3. Uma linguagem de Hipertexto, para a fácil navegação entre as fontes de informação, o HTML.

WHATWG – Web Hypertext Application Technology Working Group

Enquanto o W3C focava suas atenções na criação da segunda versão do XHTML (Linguagem reformulada do HTML, baseada em XML, com o intuito de melhorar a acessibilidade) um grupo chamado Web Hypertext Application Technology Working Group ou WHATWG trabalhava em uma versão do HTML que trazia mais flexibilidade para a produção de websites e sistemas baseados na Web, o que seria chamado hoje de HTML5.

O WHATWG (<http://www.whatwg.org/>) foi fundado por desenvolvedores de empresas como Mozilla, Apple e Opera em 2004. Eles não estavam felizes com o caminho que a Web tomava e nem com o rumo dado pelo W3C ao XHTML. Entre outros assuntos que o WHATWG se focava era Web Forms 2.0 que foi incluído no HTML5 e o Web Controls 1.0 que foi abandonado, por enquanto.

Por volta de 2006, o trabalho do WHATWG passou a ser conhecido pelo mundo e principalmente pelo W3C – que até então trabalhavam separadamente – que reconheceu todo o trabalho do grupo. Em Outubro de 2006, Tim Berners-Lee anunciou que trabalharia juntamente com o WHATWG na produção do HTML5 em detrimento do XHTML 2. Contudo o XHTML continuaria sendo mantido paralelamente de acordo com as mudanças causadas no HTML. O grupo que estava desenvolvendo especificamente o XHTML 2 foi descontinuado em 2009.

HTML5

O HTML5 é a versão mais recente do HTML. Com HTML5 a reprodução de vídeo e áudio ficou mais fácil e também é possível desenhar gráficos com `<canvas>`, SVG e CSS3 2D/3D. Temos o suporte a API's facilitado e algumas aplicações já disponíveis como Geolocation e FileAPI.

O HTML5 trouxe novos elementos semânticos para cabeçalhos, rodapés, menus, seções e artigos. Novos elementos para formulários também foram criados assim como novos atributos, novos tipos de entradas e validação automática.

Um dos principais objetivos do HTML5 é padronizar a semântica na Web, facilitando a integração do código entre os navegadores. Ao contrário das versões anteriores, o HTML5 fornece ferramentas para CSS e Javascript desempenhar seu papel de melhor forma possível. O HTML5 permite, por meio de suas APIs, a manipulação das características desses elementos, de forma que o website ou a aplicação continue leve e funcional.

O HTML5 também cria novas tags e modifica a função de outras. As versões antigas do HTML não continham um padrão universal para a criação de seções comuns e específicas como rodapé, cabeçalho, menus etc. Não havia um padrão de nomenclatura de IDs, Classes ou tags. Não havia um método de capturar de maneira automática as informações localizadas nos rodapés dos websites.

Há outros elementos e atributos que suas funções e significados foram modificados e que agora podem ser reutilizados de forma mais eficaz. Por exemplo, elementos como B ou I que foram descontinuados em versões anteriores do HTML agora assumem funções diferentes e integram mais significado para os usuários.

O HTML5 modifica a forma de como escrevemos documentos e como organizamos a informação na página de maneira semântica e com menos código.

O WHATWG tem mantido o foco para manter a compatibilidade. Nenhum site deverá ser refeito totalmente para se adequar aos novos conceitos e regras. O HTML5 está sendo criado para que seja compatível com os browsers recentes, possibilitando a utilização das novas características imediatamente.

Os motores de renderização dos browsers são os softwares responsáveis por transformar linguagem de marcação, scripts e informações de formatação em um conteúdo compreensível para ser exibido em tela. Estes mecanismos precisam ser atualizados constantemente para dar suporte a evolução da HTML.

| MOTOR | BROWSER |
|----------|-----------------------|
| Webkit | Safari, Chrome, Opera |
| Gecko | Firefox |
| Trident | Explorer |
| EdgeHTML | Edge |

PARA SABER MAIS

- [WHATWG community](#)
- github.com/whatwg/
- [\(X\)HTML5 Validator](#)

EXERCÍCIOS

<!-- Obtenha as versões atualizadas dos principais Browsers em seu sistema operacional ou dispositivo móvel. Opera, Chrome, Edge, Explorer, Firefox, Safari. -->

2- CRIANDO UM DOCUMENTO HTML

Tags são estruturas da linguagem de marcação que contém instruções para que os navegadores (e alguns dispositivos) possam renderizar uma página, elas normalmente vêm em pares, como `<p>` e `</p>`, e geralmente são palavras em inglês ou simplesmente abreviações de palavras da língua norte-americana. Em HTML é possível incorporar elementos dentro de elementos e os tags containers são chamadas "nós". Algumas tags não possuem fechamento em "par" (como a tag `<p>` `</p>`), então podemos utilizar a barra em sua própria declaração.

A primeira é a tag de início, a segunda é a tag final escrita exatamente como a tag inicial, mas com uma barra antes do nome, a barra representa o fechamento da tag e o encerramento daquela determinada marcação. O início e fim das tags também são chamados de tags de abertura e fechamento.

Para que uma página seja identificada e renderizada como uma página HTML, é necessário, antes de tudo, definirmos a estrutura padrão para que o navegador compreenda o tipo de conteúdo que está recebendo.

A estrutura básica em uma página HTML é a seguinte:

| HTML |
|--|
| <pre><!doctype html> <html> <head> <!-- Aqui ficam todos os metadados como indicado no curso Online--> <meta charset="utf-8"> <title>B E C O D E</title> </head> <body> </body> </html></pre> |

Toda página HTML5 deve iniciar com `<!DOCTYPE html>`. A declaração do `<!DOCTYPE>` não é uma tag HTML, é uma instrução para o navegador sobre qual versão do HTML a página é escrita. No HTML4 a declaração do `<!DOCTYPE>` refere-se a um DTD, porque o HTML4 foi baseado em SGML. O DTD especifica as regras da linguagem de marcação para que o navegador processar o conteúdo corretamente. HTML5 não se baseia em SGML, portanto não necessita de uma referência a uma DTD.

Após a declaração do tipo de documento a tag `<html>` abre o documento sendo seguida por `</html>` que encerra a declaração. O código HTML pode ser entendido como uma série de elementos em árvore onde alguns elementos são filhos de outros e assim por diante. O elemento principal dessa grande árvore é sempre a tag HTML.

Dentro da tag `<head>` `</head>` meta tags contem conjunto de informações a respeito da página e do conteúdo nela publicado como título e codificação, também links de arquivos externos CSS e JS. Podemos dizer que é dentro das tags `<head>` `</head>` que está toda a parte inteligente da página. Informações como Title e Description, por exemplo, são usadas pelos mecanismos de busca para montar uma SERP (Search Engine Results Page). Há também uma metatag responsável por chavear qual tabela de caracteres a página está utilizando. `<meta charset="utf-8">`

UTF-8 (8-bit Unicode Transformation Format) é um tipo de codificação Unicode de comprimento variável que pode representar qualquer carácter universal padrão do Unicode, sendo também compatível com o ASCII. Por esta razão, está a ser adaptado como tipo de codificação padrão para email, páginas web, e outros locais onde os caracteres são armazenados. Atualmente a maioria dos sistemas e browsers utilizados por usuários suportam plenamente Unicode. Por isso, fazendo seu sistema Unicode você garante que ele será bem visualizado aqui, na China ou em qualquer outro lugar do mundo. O que o Unicode faz é fornecer um único número para cada caractere, não importa a plataforma, nem o programa, nem a língua.

A tag LINK é uma ligação para fontes externas que serão usadas no documento. No nosso exemplo há uma tag LINK que importa o CSS para nossa página:

HTML

```
<head>
<!-- Aqui ficam todos os metadados como indicado no curso Online-->
<meta charset="utf-8">
<link rel="stylesheet" type="text/css" href="IndexCSS.css">

<title>B E C O D E</title>

</head>
```

O atributo `rel="stylesheet"` indica que aquele link é relativo a importação de um arquivo referente a folhas de estilo. No HTML5 há outros links relativos que você pode inserir como o `rel="archives"` que indica uma referência a uma coleção de material histórico da página.

A tag `<script>` `</script>` é usada para definir que as sequencias de comandos seguintes serão interpretadas pelo motor de Scripts do Browser, No HTML5, o atributo `type` usado no HTML4 passa a ser opcional, pois se entende que o Javascript é a linguagem padrão de scripts usados na Web.

HTML

```
<head>
<!-- Aqui ficam todos os metadados como indicado no curso Online-->
<meta charset="utf-8">
<link rel="stylesheet" type="text/css" href="IndexCSS.css">

<title>B E C O D E</title>

</head>
```

Dentro da tag <body></body> é disposto o conteúdo da nossa página, através de tags semanticamente criadas para o que desejamos exibir para o usuário. Por exemplo, um bloco de texto pode ser definido com as tags <p></p> que representam um parágrafo, enquanto uma imagem pode ser definida com a tag .

Exemplo de documento HTML simples

HTML

```
<!doctype html>
<html>
<head>
<!-- Aqui ficam todos os metadados como indicado no curso Online-->
<meta charset="utf-8">
<link rel="stylesheet" type="text/css" href="IndexCSS.css">
<title>B E C O D E</title>
</head>
<body>
<h1>Iniciando CSS</h1>
<p>Formatar informação dos sites não é algo novo. Por volta de 1970, no começo da trajetória do SGML, já se falava em algo parecido.</p>
<p>Quando o HTML foi criado, a intenção não era de forma alguma, formatar informação. A medida que o HTML foi se popularizando e evoluindo, foram incluídas em suas qualidades, o domínio de controlar algumas aparências para o documento. Isso fez com que a linguagem ficasse muito complexa, mais difícil para entender e manter.<br>
Outro problema era que os browsers tinham diferenças de implementações, o que dificultava a visualização dos sites, trazendo menos controle na navegação pela web.</p>
</body>
</html>
```

Os elementos da categoria Heading definem uma seção de cabeçalhos, que podem estar contidos ou não em um elemento na categoria Sectioning.

- h1
- h2
- h3
- h4
- h5
- h6

HTML

```
<!doctype html>
<html>
<head>
<!-- Aqui ficam todos os metadados como indicado no curso Online-->
<meta charset="utf-8">
<link rel="stylesheet" type="text/css" href="IndexCSS.css">
<title>B E C O D E</title>
</head>
<body>
<h1> Eu sou um título! </h1>
  <h2> Eu sou um subtítulo! </h2>
  <h3> Eu sou um subtítulo! </h3>
  <h4> Eu sou um subtítulo! </h4>
  <h5> Eu sou um subtítulo! </h5>
  <h6> Eu sou um subtítulo! </h6>
</body>
</html>
```

Eu sou um título!

Eu sou um subtítulo!

Eu sou um subtítulo!

Eu sou um subtítulo!

Eu sou um subtítulo!

Eu sou um subtítulo!

Eu sou um subtítulo!

Eu sou um subtítulo! **No navegador**

Eu sou um subtítulo!

Onde a <h1> é a mais importante e tipicamente maior em relação as outras e a <h6> é a menos importante na hierarquia. Então temos:

- ✓ A declaração DOCTYPE define o tipo de documento;
- ✓ As informações entre <html> e </html> vão gerar uma página web ;
- ✓ As informações entre <head> e </head> formam o cabeçalho da página;
- ✓ As tag <meta> são informações mais técnicas;
- ✓ O texto entre <title> e </title> é exibido como título da página;
- ✓ As informações entre <body> e </body> formam o conteúdo visível da página;
- ✓ O texto entre <h1> e </h1> é exibido como um título de conteúdo;
- ✓ O texto entre <p> e </p> é exibido como um parágrafo.

PARA SABER MAIS

- [Useful HTML Meta Tags](#)
- [MDN HTML element reference](#)
- [Utilizando as metatags do Facebook](#)

EXERCÍCIOS

<!--

Escreva o código do primeiro exemplo de HTML em um documento de texto e salve com o nome de index.htm na área de trabalho.

Abra o documento em diferentes Browsers.

-->

3- FORMATANDO UM DOCUMENTO

A SEMÂNTICA DAS NOVAS MARCAÇÕES

Em todas as páginas da Web existem divisões básicas referentes aos tipos de conteúdo que são colocados em cada parte do layout, como cabeçalho, rodapé ou menu de navegação.

Nas versões anteriores do HTML não havia tags com uma semântica apropriada para cada uma dessas divisões. Dessa forma, os desenvolvedores acabavam usando a tag `<div>` para todas as situações, e criando seus próprios padrões de nomenclaturas através dos atributos `id` ou `class`.

No HTML5 foram criadas diversas tags semânticas para indicar aos “user-agents” quais conteúdos estão sendo inseridos em cada uma das divisões da página, organizando e padronizando o desenvolvimento.



O HTML5 trouxe uma série de elementos que nos ajudam a definir setores principais no documento HTML. Com a ajuda destes elementos, podemos, por exemplo, diferenciar diretamente pelo código HTML5 áreas importantes do site com sidebar, rodapé e cabeçalho. Conseguimos seccionar a área de conteúdo indicando onde exatamente é o texto do artigo.

Estas mudanças simplificam o trabalho dos buscadores. Com o HTML5 os buscadores conseguem vasculhar o código de maneira mais eficaz, procurando e guardando informações mais exatas e levando menos tempo para estocar essa informação.

Até a especificação quatro da HTML usamos a tag Bold ` ` para exibir um texto em negrito em uma página Web ou a tag Italic `<i> </i>` para exibir um texto em itálico. As especificações para a HTML5 redefinem os elementos `b` e `i` retirando-lhes o caráter puramente de apresentação e a eles atribuindo uma função semântica. De acordo com a especificação o ` ` por exemplo, deve ser usado como último recurso, quando nenhuma outra tag for apropriada. As diretrizes de uso desses elementos são as seguintes:

- O elemento `<i> </i>` destina-se a marcar trechos de textos que mereçam um destaque de entonação de voz ou modo quando lidos, ou seja, textos fora do fluxo normal de leitura do texto como um todo. Por exemplo: designações taxonômicas, termos técnicos, frases em outro idioma, um pensamento, um nome de navio ou outro trecho de texto qualquer cuja apresentação tipográfica seja tipicamente em itálico.
- O elemento ` ` destina-se a representar um trecho de texto estilisticamente destacado do fluxo de prosa normal sem a ele atribuir nenhuma importância extra. Por exemplo: palavras-chave de um documento, nomes de produtos em uma resenha ou

outro trecho de texto qualquer cuja apresentação tipográfica seja tipicamente em negrito.

ÊNFASE E FORÇA

A especificação HTML5 determina que texto enfático deva ser indicado com a tag `` ``, textos de maior importância devem ser indicado com a tag `` `` e textos marcados ou destacados devem usar a tag `<mark>` `</mark>`. Você deve usar a propriedade CSS "font-weight" para definir o texto em negrito quando a única função é realizar uma marcação visual.

PARÁGRAFOS

As tags `<p>` `</p>` que usamos existem para agrupar as linhas de texto em uma página. O Navegador exibirá o texto em um único parágrafo ainda que seja digitados vários "ENTER" em um bloco de texto.

QUEBRA DE LINHA

Se precisar exibir uma parte do texto em uma nova linha(SHIFT + ENTER), você pode inserir uma quebra de linha utilizando a tag Break, ou `
`.

LINHAS HORIZONTAIS

Linhas horizontais são usadas para dividir a informação em diferentes blocos temáticos em uma mesma seção de texto. Para fazer uso de uma linha horizontal (Horizontal Rule) usa-se a tag `<hr/>`.

COMENTÁRIOS

Você pode adicionar comentários para incluir uma nota, explicações, sugestões que não fazem parte visual do documento. Usa-se a tag comment `<!-- -->` para adicionar trechos de código que serão ignorados pelo Browser durante a leitura do documento HTML. Normalmente você usa a tag `<!-- -->` para fornecer informações sobre o autor do documento ou para adicionar uma mensagem de copyright.

PRÉ-FORMATADO

Como vimos ao usar a tag `<p></p>`, por padrão, os browsers não exibem os espaços extras e linhas em branco de um documento HTML na página Web correspondente. A tag Pre-formatted `<pre></pre>` conserva espaços múltiplos e linhas em branco mantendo a forma em que o texto foi escrito

HTML

```
<!DOCTYPE html>
<html>
  <!--Inicia a tag de marcação do cabeçalho-->
  <header>
    <meta charset="utf-8"><!--Aceita acentuação na página-->
    <meta charset="pt-br"><!--Página em Português Brasil-->
    <!--mostra o título da página-->
    <title> Tags de Parágrafo</title>
  </header>
  <!--Inicia o corpo da página-->
  <body>
    <section id="Conteúdo">
      <p> Iniciando um <strong>parágrafo</strong> usando a
      tag p.</p>
    </hr>
    <pre> Já usando o pré de <mark>pré formatado</mark>, temos
    o mesmo parágrafo, porém com
    <i>visualização diferenciada</i>.</pre>
    <p>usando a quebra <br>de linha.</p>
  </section>
</body>
</html>
```

IMAGENS

Usamos imagens para melhorar a comunicação dos conceitos explicados em uma página Web. Imagens melhoram, compõe a aparência do documento e destacam as ideias. Tipicamente utilizamos imagens para incluir títulos e logos em uma página, também como links para outras páginas. Primeiramente quando adicionamos uma imagem a uma página, é preciso manter em mente o tamanho da imagem, formato e posição desta imagem. O tamanho da imagem reflete diretamente o tamanho do arquivo da imagem que por sua vez reflete no tempo de carregamento deste arquivo em uma rede. Quanto menor a velocidade de conexão do usuário, maior o tempo de espera para a carga completa da imagem. Você pode arrumar as imagens de várias maneiras e posições em sua página, entretanto, o posicionamento de diversas imagens na mesma página deverá ser consistente. Neste ponto Vamos trocar o título de nossa página por uma imagem. O atributo `src = ""` indica o caminho do arquivo que será apresentado pela página do browser neste ponto do documento HTML.

HTML

```
  
</a><!--o </a> indica fechamento do link-->
```

A sintaxe é a seguinte:

- img= tag utilizada para inserir uma imagem na página web.
- src= atributo usado para especificar o arquivo da imagem a ser inserida.
- alt= texto alternativo à imagem, considerado atributo de acessibilidade é bastante valioso para deficientes visuais
- width = largura em pixels.
- height = altura em pixels.

As primeiras imagens da Web eram formatadas utilizando entre 16 e 256 cores, já que este era o limite máximo de exibição dos monitores. Além disso, o uso de um número grande de cores numa imagem elevava muito o tamanho do arquivo. Atualmente não existe uma preocupação muito grande com a quantidade de cores, mas sim com o tamanho do arquivo.

A Internet suporta quase todos os tipos de arquivos de imagens. Tipicamente são utilizados em documentos HTML:

GIF - Graphics Interchange Format suporta oito bits por pixel, permitindo o uso de uma paleta de até 256 cores. Utiliza o tipo de compressão LZW e é mais indicado para gráficos, ícones e imagens que não necessitam muitas cores. Permite imagens com fundo transparente.

JPG - Joint Photographic Experts Group é o método mais utilizado para comprimir imagens fotográficas. Foi introduzido em 1983 e suporta até 16,8 milhões de cores. Não permite imagens com fundos transparentes.

PNG - Portable Networks Graphics é o substituto do formato GIF, que assim como JPG permite exibição de até 16,8 milhões de cores e dá suporte a canal alpha e fundos transparentes.

LISTAS

Uma lista é uma coleção de itens relacionados, são usadas para organizar dados como uma sequência de passos ou itens em um grupo.

Em HTML existem quatro tipos de listas:

1. Numeradas - usada para exibir um conjunto de itens depois de um número, de forma ordenada.
2. Não numeradas - usada para exibir um conjunto de itens depois de uma marcação, é chamada também de lista de marcação, ou ainda não ordenada.
3. Intercaladas - combinação de duas ou mais listas. Listas intercaladas estão sendo descontinuadas não sendo mais utilizadas.
4. Por definição - para exibir um conjunto de itens e suas descrições

Observe o código para uma lista tanto ordenada quanto desordenadas como vimos o curso Online como criar cada uma delas, além claro de como criá-las intercaladas.

HTML

```
<body>
<p>A BeCode oferece o curso <strong><i>Desenvolvedor Primeiros
Passos</i></strong> onde você irá aprender
<mark>os elementos básicos das linguagens de:</mark>
  <ol id="ListaOrdenada">
    <li>HTML</li><!--linha da lista-->
      <ul id="ListaDesordenada">
        <li>Tags de marcação</li><!--linha da lista-->
      </ul>
    <li>CSS</li>
      <ul id="ListaDesordenada">
        <li>Seletores básicos de formatação</li><!--linha da lista-->
      </ul>
    <li>JS</li>
      <ul id="ListaDesordenada">
        <li>Scripts de otimização para formulários</li><!--linha da lista-->
      </ul>
    </ol>
  </p>
</body>
</html>
```

LINKS

Conexões pelas quais você conecta páginas que podem ser do mesmo website, externas, de qualquer outro site, ou ainda conectar páginas a outros documentos. Os links podem ser criados no texto e também em outros elementos do website, como em imagens e também itens de um menu. A tag Anchor <a> (âncora) é utilizada para criar links no hipertexto. A sintaxe desta tag é a seguinte:

HTML

```
<!--Usando link externo-->
<a href="http://www.becode.com.br/">
```

O browser exibe o hiperlink em um formato diferente do texto normal, normalmente sublinhado e na cor azul. Ao especificar uma URL apropriada para um serviço de Internet, você também pode interligar sua página Web a esse serviço. Por exemplo, você pode criar um link entre sua página e o seu e-mail.

TABELAS

Tabelas devem ser usadas exclusivamente na listagem de dados tabulares. Uma tabela é definida pela tag `<table>`, seguida de suas marcações `<thead>`, `<tbody>`, `<tfoot>` e `<caption>`, são as definições para cabeçalho, rodapé e corpo da tabela respectivamente. O elemento `<caption>` é responsável pelo título da tabela. É esta tag que vai apresentar o que a tabela tem que transmitir para o usuário, deve ser inserido logo após a tag `<table>` e só pode haver um elemento `<caption>` por Tabela.

As tabelas são divididas em linhas chamadas de table rows com a tag `<tr>`, as quais são divididas por colunas com a tag `<td>`. Linhas que pertencem ao cabeçalho da tabela `<thead>` tem a declaração de suas colunas com a tag `<th>`.

HTML

```
<body>
<table>
  <caption>TABELA JOGO DA VELHA</caption>
  <thead>
    <tr><th colspan="3">Cabeçalho Tabela</th></tr>
    <tr><td colspan="3">   </td></tr>
  </thead>
  <tbody>
    <tr><td colspan="3">Inicio do Corpo </td></tr>
    <tr><td>X</td><td>O</td><td>X</td></tr>
    <tr><td>O</td><td>X</td><td>O</td></tr>
    <tr><td>X</td><td>O</td><td>X</td></tr>
    <tr><td colspan="3"> Fim do Corpo </td></tr>
  </tbody>
  <tfoot>
    <tr><td colspan="3">   </td></tr>
    <tr><td colspan="3">Rodapés Tabela</td></tr>
  </tfoot>
</table>
</body>
</html>
```

O atributo "colspan" determina a quantidade de colunas que a célula da tabela deverá ocupar, "col" vem de column (coluna) e span quer dizer expandir, abranger. Note que as células de cabeçalho e rodapé estão sozinhas em suas linhas e queremos fazer uma célula ocupar o espaço de três colunas, então na tag da célula em questão atribuímos colspan="3".

A mesma lógica pode ser aplicada ao atributo rowspan onde "row" tem o significado de "linha",

TAG SPAN

O elemento HTML ` ` é um container generico em linha para texto, que não representa nada por natureza. Ele pode ser usado para agrupar elementos para fins de estilo (usando os atributos `class` ou `id`), ou para compartilhar valores de atributos como `lang`. Ele deve ser usado somente quando nenhum outro elemento semântico for apropriado. `` é muito parecido com o elemento `<div> </div>`, entretando `<div> </div>`, é um elemento de nível de bloco enquanto ` ` é um elemento em linha.

HTML

```
<ul>
  <li><a href="#">A <span class="subTitle">Subtitle</span></a></li>
  <li><a href="#">B <span class="subTitle">Subtitle</span></a></li>
  <li><a href="#">C <span class="subTitle">Subtitle</span></a></li>
  <li><a href="#">D <span class="subTitle">Subtitle</span></a></li>
  <li><a href="#">E <span class="subTitle">Subtitle</span></a></li>
</ul>
```

TAG DIV

O elemento `<div> </div>` é um elemento genérico que serve para criar divisões que não tem significado semântico nenhum, formatações tipicamente visuais não carregam nenhum tipo de informação extra. Geralmente são livres agrupados de conteúdo que devem se distinguir por CLASSES e IDs.

HTML

```
<section id="noticias">
  <div> ... </div>
  <div> ... </div>
  <div> ... </div>
</section>
```

TAG SECTION

A tag `<section> </section>` define uma nova seção genérica no documento, um bloco de conteúdo relacionado. Por exemplo, a home de um site pode ser dividida em diversas seções: introdução, novidades ou informações de contato.

O conteúdo é relacionado e não faria sentido por si próprio? Utilize `<section> </section>`

HTML

```
<!doctype html>
<html>
<head>
<!-- Aqui ficam todos os metadados como indicado no curso Online-->
<meta charset="utf-8">
<link rel="stylesheet" type="text/css" href="IndexCSS.css">
<title>B E C O D E</title>
</head>
<body>
<section id="Listas">
<h1> Eu sou um título! </h1>
    <h2> Eu sou um subtítulo! </h2>
    <h3> Eu sou um subtítulo! </h3>
    <h4> Eu sou um subtítulo! </h4>
    <h5> Eu sou um subtítulo! </h5>
    <h6> Eu sou um subtítulo! </h6>
</section>
</body>
</html>
```

TAG ARTICLE

Define um post, um artigo, um bloco de texto ou ainda um mini aplicativo embutido no conteúdo (widget). O elemento article é um elemento que forma uma parte independente ou reutilizável do documento, sendo mais específico que o section e que o div. Todo o conteúdo do article pode ser utilizado em feeds ou outros meio de "sindicação". O <article> </article> é um tipo especializado de <section> </section>, tem um significado mais específico do que semântico.

O conteúdo faria sentido por si próprio em um leitor de feed? Utilize <article> </article>

HTML

```
<!doctype html>
<html>
<head>
<!-- Aqui ficam todos os metadados como indicado no curso Online-->
<meta charset="utf-8">
<link rel="stylesheet" type="text/css" href="IndexCSS.css">
<title>B E C O D E</title>
</head>
<body>
<section id="Listas">
  <article id="complementoListas">
    <h1>Tralho com Títulos</h1>
    <p>Agora vamos ver exemplos de como usar uma tag de <strong>título</strong>.</p>
  </article>
  <h1>Eu sou um título! </h1>
  <h2>Eu sou um subtítulo! </h2>
  <h3>Eu sou um subtítulo! </h3>
  <h4>Eu sou um subtítulo! </h4>
  <h5>Eu sou um subtítulo! </h5>
  <h6>Eu sou um subtítulo! </h6>
</section>
</body>
</html>
```

TAG NAV

A tag <nav></nav> representa uma seção da página que contém links para outras partes do Website. Nem todos os grupos de links devem ser elementos <nav></nav>, apenas aqueles grupos que contém links importantes. O elemento <nav></nav> não deve ser usado em links mencionados em um parágrafo de conteúdo e também em links no rodapé do tipo "site criado por". Essa tag pode ser aplicado naqueles blocos de links que geralmente são colocados no Rodapé e também para compor o menu principal do site.

HTML

```
<nav id='cssmenu'>
  <ul class="itens">
    <li><a href="Index_2.html">Home</a></li>
    <li><a href='#'>Cursos</a></li>
    <li><a href="Formulario.html">Contato</a></li>
  </ul>
</nav>
```

TAG FOOTER

Marca a área inferior, normalmente conhecida como rodapé, do documento ou do conteúdo de uma seção específica a qual está subordinado.

Utilizações de <footer> </footer> como último elemento do documento, na função de rodapé global e contendo um bloco de navegação global do documento:

HTML

```
<footer>
  <p>© 2015 BeCode | You are simply </p>
</footer>
```

TAG HEADER

Representa um grupo de introdução ou elementos de navegação. O elemento header pode ser utilizado para agrupar índices de conteúdos, campos de busca ou até mesmo logos. Dentro do header é possível inserir um ou mais elementos h1 até h6, organizando-os em uma hierarquia de títulos e subtítulos.

Como cabeçalho principal do documento.

HTML

```
<!-- CABECALHO -->
<header>
<div class="titulo">
<!--inline-->

</div>

<nav id='cssmenu'>
  <ul class="itens">
    <li><a href="Index_2.html">Home</a></li>
    <li><a href='#'>Cursos</a></li>
    <li><a href="Formulario.html">Contato</a></li>
  </ul>
</nav>











</header>
<!-- /CABECALHO -->
```

PARA SABER MAIS

- [HTML YouTube Videos](#)
- [HTML5 Video](#)
- [HTML5 Audio](#)
- [HTML Tag List](#)

EXERCÍCIOS

<!--Construa o seguinte documento usando HTML semântico. Aqui use tabelas.

| Место | Имя | Состояние \$ млн | В прошлом году \$ млн | Источник состояния |
|-------|---|---------------------|--------------------------|---|
| 1 |  Алишер Усманов МеталлИнвест | 18 100 | 17 700 | металлургия, интернет, телекоммуникации |
| 2 |  Владимир Лисин НЛМК | 15 900 | 24 000 | металлургия, транспорт |
| 3 |  Алексей Мордашов Северсталь | 15 300 | 18 500 | металлургия, машиностроение |
| 4 |  Владимир Потанин Норильский никель | 14 500 | 17 800 | металлургия, медь |
| 5 |  Вагит Алекперов Лукойл | 13 500 | 13 900 | нефть, инвестиции |
| 6 |  Михаил Фридман Альфа-Групп | 13 400 | 15 100 | нефть, финансы, телекоммуникации, розничная торговля |
| 7 |  Михаил Прохоров Группа Онексим | 13 200 | 18 000 | металлургия, инвестиции |
| 8 |  Виктор Вексельберг Ренова | 12 400 | 13 000 | нефть, металлургия, инвестиции |
| 9 |  Роман Абрамович Millhouse Capital | 12 100 | 13 400 | металлургия, инвестиции |
| 10 |  Леонид Михельсон Новатэк | 11 900 | 9 100 | газ, нефтехимия |

-->

4- FORMULÁRIOS HTML

Formulários HTML são usados para transmitir dados a um servidor. Um formulário HTML pode conter elementos de entrada, como campos de texto, caixas de seleção, botões de rádio, botões de envio e demais elementos como: listas de seleção, textarea, fieldset, legendas e elementos de rotulagem. Para se criar um form usamos as tags `<form>` `</form>`.

TAG INPUT

O elemento mais importante de um `<form>` `</form>` é o elemento `<input>` `</input>`, ele é usado para dar entrada nas informações do usuário. Um elemento `<input>` `</input>` pode variar de muitas formas, dependendo do "atributo type", inputs podem ser do tipo campo de texto, caixa de verificação, senha, botão de rádio, botão entre outros.

HTML

```
<!DOCTYPE html>

<html>

<header>
  <meta charset="utf-8">
  <meta charset="pt-br">
  <title> Exemplo Completo</title>
</header>
<body>
  <form name="form1" action="enviar.php" method="post">
    <!--conteúdo do formulário-->
  </form>
```

Definindo um campo de senha que oculta os caracteres digitados

HTML

```
<form id="_Form1">
  Senha: <input type="password" id="_Senha">
</form>
```

Os tipos `<input type="radio">` definem botões que permitem que o usuário selecione apenas uma opção entre de um número de opções, botões que se auto excluem fazem parte de um grupo definido através do atributo `name`:

HTML

```
<!DOCTYPE html>

<html>

<header>
<meta charset="utf-8">
<meta charset="pt-br">
<title> Exemplo Completo</title>
</header>
<body>
<form name="form1" action="enviar.php" method="post">
  <!--conteúdo do formulário-->

  <h1>Dê sua opinião sobre curso</h1>
  <input type="radio" name="_opinioao" id="_opinioao" value="otimo" class="_op" />Ótimo<br/>
  <input type="radio" name="_opinioao" id="_opinioao" value="bom" class="_op" />Bom<br />
  <input type="radio" name="_opinioao" id="_opinioao" value="regular" class="_op" />Regular<br />
  <input type="radio" name="_opinioao" id="_opinioao" value="ruim" class="_op" />Ruim<br />
  <input type="radio" name="_opinioao" id="_opinioao" value="pessimo" class="_op" />Péssimo<br />
</form>
</body>
</html>
```

As caixas de seleção permitem que o usuário selecione nenhuma ou mais opções de um número limitado de opções.

HTML

```
Canhoto? <input type="checkbox" id="_Canhoto" />
```

Um botão de envio é usado para enviar dados de formulário para um servidor. Os dados são enviados para a página especificada no atributo `action` do formulário.

HTML

```
<!--usando botões-->
<input type="submit" class="btn" onclick="return validar()" value="Enviar Dados">
<input type="reset" class="btn" value="Limpar Dados">
```

O código cria um típico campo para pesquisas.

HTML

```
Busca: <input list="_cidades" autocomplete="on" name="_BuscarCidade" type="search"
placeholder="pesquisar...">
<datalist id="_cidades">
  <option value="Brasília">
  <option value="Florianópolis">
  <option value="Rio de Janeiro">
  <option value="Porto Alegre">
  <option value="Curitiba">
  <option value="São Paulo">
</datalist>
```

TAG SELECT

A tag `<select>` `</select>` define o início de uma lista de seleção. As listas de seleções permitem que o usuário selecione uma das opções de um número limitado de opções.

HTML

```
<select id="_Carro">
  <option value="Chevrolet">Chevrolet</option>
  <option value="Volvo">Volvo</option>
  <option value="Saab">Saab</option>
  <option value="VW" selected="selected">VolksWagem</option>
  <option value="Mercedes">Mercedes</option>
  <option value="Audi">Audi</option>
  <option value="Fiat">Fiat</option>
</select>
```

TAG TEXTAREA

A tag `<textarea>` `</textarea>` define o início de uma caixa de texto que permite ao usuário escrever um texto com múltiplas linhas.

HTML

```
<textarea id="_TextoMultiLine"></textarea>
```

O ATRIBUTO PATTERN

Uma das tarefas mais enfadonhas de se fazer é validar formulários. Infelizmente, é também uma das mais comuns. O HTML5 facilita muito ao validar formulários, tornando automática boa parte do processo. O atributo "pattern" permite definir expressões regulares de validação. Exemplo de como validar CEP:

HTML

```
<form>
  <label for="CEP">
    CEP: <input name="_CEP" id="_CEP" required pattern="\d{5}-?\d{3}" />
  </label>
  <input type="submit" value="Enviar" />
</form>
```

PLACEHOLDER

Atributo usado para auxiliar o usuário no preenchimento do campo, dá uma dica do que deve ser inserido.

HTML

```
<input name="pesquisar" type="search" placeholder="Pesquisar" />
```

REQUIRED

Para tornar um campo de formulário obrigatório (seu valor precisa ser preenchido) basta, em HTML5, incluir o atributo "required".

HTML

```
<input name="login" required/>
```

PARA SABER MAIS

- [Expressões Regulares - Guia de Consulta Rápida](#)
- [HTML Forms](#)
- [HTML Input Types](#)

EXERCÍCIOS

<!--

Construa o seguinte formulário usando HTML semântico.

A Simple Form

Form Fundamentals

Customer Info

Name:

Telephone:

Email address:

Books

Quantity (Maximum 5):

© 2011 Acme United. All rights reserved.

-->

5- CSS

A CSS (Cascading Style Sheets) é uma linguagem de estilo que surgiu na época do HTML 4.0 e é utilizada para definir como os elementos de uma linguagem de marcação, como o HTML ou XML, serão apresentados visualmente pelos navegadores.

A vantagem é que o CSS é bem mais robusto que o HTML para estilização destes elementos e isto aumenta a possibilidade de criar layouts mais atrativos visualmente e por consequência aumentando o poder comercial do site.

O HTML nunca foi destinado a conter tags para a formatação de documentos web, ele foi concebido para definir o conteúdo de um documento. Quando tags como , e atributos de cor foram adicionados no HTML 3.2, começou um pesadelo para os desenvolvedores. Com a necessidade da criação de complexos aplicativos e sites, adicionar fontes e informações de cores a cada página individualmente tornou-se um processo longo e caro. Para resolver este problema, o World Wide Web Consortium (W3C) criou o CSS. Quando o HTML 4.0 surgiu, toda a formatação visual pode ser removida do HTML, sendo colocada em um arquivo separado, a folha de estilo.

Existem três maneiras de aplicarmos nossas formatações em CSS, duas delas não devem ser usadas por questões de boas práticas, mas é importante você ter conhecimento de todas, pois certamente você vai se deparar com elas durante em algum momento.

ATRIBUTO STYLE

A primeira maneira de se criar estilo em CSS é como um atributo style no próprio elemento. Uma das grandes vantagens do CSS é manter as regras de estilo fora do HTML e usando o atributo style não fazemos isso. Por este motivo não se recomenda esse tipo de uso na prática. Além do fato que cada vez que formos usar a tag <p> </p> temos que repetir o código de formatação dela, isso duplica o tempo de codificação de cada página e o nosso trabalho.

CSS

```
<p style="color: #000; font-size: 20px;">
  Olá! Sou um exemplo de texto. :))
</p>
```

TAG STYLE

A segunda maneira de se utilizar o CSS é declarando suas propriedades dentro de uma tag <style> </style>. Como estamos declarando as propriedades visuais de um elemento no header do nosso documento, precisamos indicar qual elemento nos referimos. Fazemos isso utilizando um seletor CSS que é basicamente uma forma de buscar certos elementos dentro da página que receberão os estilos visuais desejados.



No exemplo acima estamos dizendo que o texto contigo dentro da tag `<h1>` vai ficar na cor azul e o tamanho da fonte vai ser de **12px**.

O bloco de declaração contém uma ou mais declarações separadas por ponto e vírgula. Cada declaração inclui um nome de propriedade e um valor, separados por dois pontos.

No exemplo a forma de aplicar o bloco de seleção no código HTML

CSS

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <title>Custom validator</title>
  <style>
    h1 {
      color: blue;
      font-size: 12px;
    }
  </style>
</head>
<body>
  <h1>Meu primeiro título formatado por CSS</h1>
  <p>Meu primeiro parágrafo</p>
</body>
</html>
```

ARQUIVO EXTERNO

A terceira e última maneira de aplicarmos os estilos de CSS em uma página web, é através de um arquivo externo, com a extensão CSS. Este arquivo externo deve conter apenas estilos em CSS e nunca deve conter informações em HTML ou outras linguagens.

CSS

```
<head>
  <title> Target CodeShop( ); </title>
  .
  .
  .
  <link href="index.css" rel="stylesheet" />
</head>
```


UNIDADES DE MEDIDA

CSS oferece diferentes unidades para a expressão de comprimento. Algumas têm origem na tipografia, como o ponto (pt) e a paica (pc), outras unidades de medida são conhecidas devido a sua relação a aspectos de comuns do dia-a-dia, como centímetro (cm) e polegada (in). E uma unidade que foi inventada especificamente para CSS: px.

Não há qualquer restrição sobre qual unidade pode ser usada em qual lugar. Se uma propriedade aceita um valor em px ('margin: 5px') ela também aceita o valor em polegadas ou centímetros ('margin: 1.2in; margin: 0.5cm').

As unidades CSS estão agrupadas em dois tipos:

1. UNIDADES ABSOLUTAS - não estão referenciadas a qualquer outra unidade e não são herdadas. São unidades de medida de comprimento definidas nos sistemas de medidas pela física: centímetros, polegadas... São indicadas para serem usadas quando as mídias de exibição são perfeitamente conhecidas.
 - ✓ pt - point :1/72 in;
 - ✓ pc - pica :12 points ou 1/6 in;
 - ✓ mm - milímetro :1/10 cm;
 - ✓ cm - centímetro :1/100 m;
 - ✓ in - polegada :2,54 cm;
2. UNIDADES RELATIVAS - Tem relação a outra medida. Folhas de Estilo em Cascata que usam unidades de comprimento relativas são mais apropriadas para ajustes de uso em diferentes tipos de mídia.
 - ✓ px - pixel - Relativo ao dispositivo (mídia) de exibição;
 - ✓ em - Relativo ao tamanho da fonte ('font-size') herdada;
 - ✓ ex - Relativo a altura da letra x (xis) da fonte herdada;
 - ✓ % - percentagem - Relativo a uma medida previamente definida.

Pixel

A unidade de medida de comprimento pixel é relativa a resolução do dispositivo de exibição como a tela de um monitor. Pixel é o menor elemento em um dispositivo de exibição, ao qual é possível atribuir-se uma cor. Considere um dispositivo de exibição construído com uma densidade de 90 dpi (dpi = dots per inch = pontos por polegada). Por definição, a referência padrão para pixel é igual a um ponto no citado dispositivo. Daí pode-se concluir que 1 pixel naquele dispositivo de exibição é igual a $1/90 \text{ inch} = 0,28 \text{ mm}$. Para uma densidade de 300 dpi 1 pixel é igual a $1/300 \text{ inch} = 0,085 \text{ mm}$. Assim, pixel é uma medida relativa a resolução do dispositivo de exibição.

Em

A unidade de medida de comprimento em referencia-se ao tamanho da fonte (letra) do seletor onde for declarada. Quando em for declarada para a propriedade font-size referencia-se ao tamanho da fonte (letra) do elemento pai. Quando em for declarada para o elemento raiz do documento referencia-se ao valor inicial (default) do tamanho de fonte (letra).

Os exemplos esclarecem as definições:

CSS

```
h1 { line-height: 1.2em }
```

line-height de <h1> será 20% maior do que o tamanho das letras de <h1>

CSS

```
h1 { font-size: 1.2em }
```

font-size de <h1> será 20% maior do que o tamanho das letras herdado por <h1> p.ex.: se h1 estiver contido numa div com font-size=10px então font-size de h1 = 12px

Ex:

A unidade de medida de comprimento ex é igual a altura da letra x(xis) minúscula.

Porcentagem - %

Valores em porcentagem são relativos a outro valor anterior declarado. Este valor anterior há que estar definido e tipicamente esta definição está em uma determinada propriedade do mesmo elemento, na propriedade do elemento "pai" ou mesmo no contexto geral da formatação.

CSS

```
p { font-size: 10px }  
p { line-height: 120% } /*120% de 'font-size'=12px*/
```

Segue tabela de equivalência entre medidas e seus valores aproximados, valores dependem do Navegador utilizado e também do Sistema Operacional:

| PIXELS | EM'S | % |
|-------------|------------|-------------|
| 8px | 0.5em | 50% |
| 9px | 0.55em | 55% |
| 10px | 0.625em | 62.5% |
| 11px | 0.7em | 70% |
| 12px | 0.75em | 75% |
| 13px | 0.8em | 80% |
| 14px | 0.875em | 87.5% |
| 15px | 0.95em | 95% |
| 16px | 1em | 100% |
| 17px | 1.05em | 105% |
| 18px | 1.125em | 112.5% |
| 19px | 1.2em | 120% |
| 20px | 1.25em | 125% |
| 21px | 1.3em | 130% |
| 22px | 1.4em | 140% |
| 23px | 1.45em | 145% |
| 24px | 1.5em | 150% |
| 26px | 1.6em | 160% |
| 29px | 1.8em | 180% |
| 32px | 2em | 200% |
| 35px | 2.2em | 220% |
| 36px | 2.25em | 225% |

| | | |
|------|--------|------|
| 37px | 2.3em | 230% |
| 38px | 2.35em | 235% |
| 40px | 2.45em | 245% |
| 42px | 2.55em | 255% |
| 45px | 2.75em | 275% |
| 48px | 3em | 300% |

SELETOR POR ELEMENTO (TAG)

Seletores CSS permitem manipular elementos HTML, eles são utilizadas para "encontrar" (ou selecionar) elementos HTML com base no ID, classes, tipos, atributos, valores de atributos etc.

O seletor de elemento seleciona elementos com base no nome do elemento. Você pode selecionar todos os elementos `<p>` `</p>` em uma página (todos os elementos `<p>` `</p>` serão centro-alinhados, com a cor de texto em vermelho)

CSS

```
p {  
  text-align: center;  
  color: red;  
}
```

SELETOR POR ID

O seletor ID usa o atributo id de uma tag HTML para encontrar o elemento específico.

Um id deve ser único dentro de uma página, então só devemos usar o seletor de id quando desejamos encontrar um único elemento.

Para selecionar um elemento com um ID específico, devemos criar um ID usando o símbolo # mais o nome do id que desejamos. IMPORTANTE: Não inicie um nome de ID com um número!

CSS

```
#textoCentralizado {  
  text-align: center;  
  color: red;  
}
```

SELETOR POR CLASSE

O seletor "CLASSE" usa o atributo class de uma tag HTML para encontrar o elemento específico.

Uma classe poder não ser única dentro de uma página, então podemos usar o seletor de classe quando desejamos encontrar um ou mais elementos.

CSS

```
.textoCentralizado {  
  text-align: center;  
  color: red;  
}
```

```
}
```

Podemos também determinar quais elementos HTML específicos devem ser afetado por uma classe. No exemplo a seguir, todos os elementos p com a classe "textoCentralizado" serão centralizados.

CSS

```
p.textoCentralizado {  
  text-align: center;  
  color: red;  
}
```

COMBINANDO SELETORES

Podemos também combinar os seletores que aprendemos acima, para conseguir diferentes elementos e partes mais específicas de nossos sites. Alguns exemplos:

p.destaque seleciona apenas os parágrafos que possuem a classe "destaque".

div#cabecalho h1 seleciona tags h1 que estejam dentro da div com a id "cabecalho".

#conteudo ul li a seleciona links (tag a) dentro de itens de lista dentro de tags ul que estejam dentro de um elemento com a id "conteudo".

#conteudo p.destaque strong seleciona elementos strong dentro de parágrafos com a classe "destaque" que estejam dentro de um elemento com a id "conteudo".

.mensagem.destaque seleciona apenas elementos que tenham a classe "mensagem" e a classe "destaque".

Separando itens por vírgulas, como **p.destaque, h1, a** seleciona todos os respectivos elementos para as regras, útil para diminuir a repetição de regras no arquivo CSS.

::BEFORE

O pseudo-elemento ::before do CSS3 serve para adicionar alguma formatação ou conteúdo antes de um elemento HTML qualquer.

HTML

```
<ul>  
  <li>Design para a Internet: Projetando a Experiência Perfeita</li>  
  <li>Não Me faça Pensar</li>  
  <li>Construindo Sites com CSS e (X)HTML</li>  
  <li>jQuery: A Bíblia do Programador JavaScript</li>  
  <li>Use a Cabeça HTML com CSS e XHTML</li>  
</ul>
```

CSS

```
li {  
  font-family: Arial, sans-serif;  
  line-height: 200%;  
  color: #333;  
  list-style: none;  
}  
  
li:before {  
  content: "Livro: ";  
  color: #D35529;  
  font-weight: bold;  
}
```

::AFTER

O pseudo-elemento ::after do CSS3 serve para adicionar formatação ou conteúdo após um elemento HTML.

HTML

```
<ul>  
  <li rel="timao">Emerson Sheik</li>  
  <li rel="timao">Danilo</li>  
  <li rel="tricolor">Luis Fabiano</li>  
  <li rel="tricolor">Jadson</li>  
</ul>
```

CSS

```
li {  
  font-family: Arial, sans-serif;  
  line-height: 200%;  
  color: #333;  
  list-style: none;  
}  
  
li:after {  
  color: #D35529;  
  font-weight: bold;  
}  
  
li[rel="timao"]:after {  
  content: " - jogador do Corinthians.";  
}  
  
li[rel="tricolor"]:after {  
  content: " - jogador do Fluminense";  
}
```

::FIRST-CHILD E ::LAST-CHILD

Os seletores ::first-child e ::last-child do CSS3 nos permitem formatar apenas o primeiro ou o último elemento dentro de uma sequência.

HTML

```
<section>
  <div>&nbsp;</div>
  <div>&nbsp;</div>
  <div>&nbsp;</div>
  <div>&nbsp;</div>
  <div>&nbsp;</div>
  <div>&nbsp;</div>
</section>
```

CSS

```
section {
  width: 800px;
  overflow: auto;
  margin: 0 auto;
}

div {
  width: 100px;
  height: 100px;
  background-color: #333;
  margin: 20px;
  float: left;
}

div:first-child {
  background-color: #900;
  margin-left: 0px;
}

div:last-child {
  background-color: #006;
  margin-right: 0px;
}
```



PROPRIEDADE CONTENT

A propriedade content cria um conteúdo estático utilizando os pseudo-elementos ::after e ::before e só pode ser usada em conjunto com estes.

No CSS existe a possibilidade de você inserir um elemento de mentirinha (pseudo-elemento) em objetos no HTML. Estes elementos podem te auxiliar em diversos momentos do desenvolvimento, evitando a criação de elementos HTML vazios para produzir algum detalhe do layout que possa se misturar com o conteúdo real.

Seguidamente é preciso criar um elemento `` vazio no começo ou no final de uma marcação qualquer para produzir uma seta, um background decorativo ou qualquer outro detalhe, usar os pseudo-elementos é como obter um `` vazio, só que melhor, por que não é gerado lixo no HTML. A propriedade `content` pode ter os seguintes valores:

| VALOR | COMPORTAMENTO |
|--|---|
| <code>none, normal</code> | O conteúdo não vai ser gerado. |
| <code><string></code> | Uma string de texto normal. |
| <code>url()</code> | Permite inserir imagens de fontes externas. |
| <code>counter()</code> | Inserir contadores. |
| <code>attr(attribute)</code> | Obter valor de um determinado atributo do elemento. |
| <code>open-quote, close-quote</code> <code>no-open-quote, no-close-quote</code> | Gera marcações de aspas. |

Vamos analisar o exemplo: Imagine agora que você queira inserir em um título algum conteúdo no início ou no final dele. Imagine que você queira inserir no começo de todos os títulos, algum conteúdo... Por exemplo, todos os títulos devem começar com a palavra "Capítulo:". Em vez de colocar direto no HTML, podemos colocar via CSS.

HTML

```
<h1>:Times</h1>
<h1>:Times</h1>
<h1>:Times</h1>
<h1>:Times</h1>
```

Fizemos então alguns títulos em HTML. Mas teríamos que fazer um contador ou alguma outra técnica para poder contar cada capítulo, seria prático se pudéssemos por de uma maneira mais fácil e rápida. A propriedade `"content"` permite através de um `"counter-increment"`

CSS

```
h1 {
  counter-increment: numero-do-titulo;
}
h1::before {
  content: "Capítulo " counter(numero-do-titulo);
}
```

Colocamos a propriedade `counter-increment` com o valor `numero-do-titulo`. Esse valor é como se fosse uma variável em todos os elementos `h1`. Toda vez que o browser renderizar um `H1`, ele pegará essa variável (`numero-do-titulo`) que definimos e incrementará o valor dela.

No pseudo-elemento `::before` de cada título, exibimos o valor de contador daquela variável com o `counter(numero-do-titulo)`.

ATTRIBUTE SELECTOR

No CSS3 também é possível formatarmos elementos que possuam um atributo específico. Existem diferentes variações e combinações de seletores:

HTML

```
<section>
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div title="Seletores de atributos">4</div>
  <div>5</div>
  <div>6</div>
</section>
```

CSS

```
div {
  width: 100px;
  height: 100px;
  background-color: #333;
  margin: 10px;
  float: left;
  line-height: 100px;
  text-align: center;
  color: #fff;
  font-weight: bold;
  font-family: Verdana, Geneva, sans-serif;
  font-size: 16px;
}

div[title] {
  background-color: #900;
}
```

TARGET SELECTOR

O seletor target do CSS3 nos permite aplicar uma formatação ao elemento que é alvo ativo de um link. No exemplo existem três blocos diferentes, sobrepostos pelo posicionamento absoluto. Cada um dos links possui em seu atributo uma âncora para cada um dos blocos. Ao clicar em um dos links, o seletor aplica a propriedade ao bloco correspondente, fazendo com que ele se sobreponha aos demais.

HTML

```
<nav>
  <a href="#um" title="ir para o quadro 1">Um</a>
```



```
<a href="#dois" title="ir para o quadro 2">Dois</a>
<a href="#tres" title="ir para o quadro 3">Três</a>
</nav>

<div id="um">Quadro 1</div>
<div id="dois">Quadro 2</div>
<div id="tres">Quadro 3</div>
```

HTML

```
nav {
  margin: 10px;
}
nav a {
  padding: 10px;
  font-family: Arial, sans-serif;
}
div {
  width: 300px;
  height: 100px;
  line-height: 100px;
  text-align: center;
  padding: 10px;
  color: #fff;
  font-family: Arial, sans-serif;
  font-size: 16px;
  position: absolute;
  top: 100px;
  left: 20px;
  z-index: 1;
}
div#um {
  background-color: #009;
}
div#dois {
  background-color: #060;
}
div#tres {
  background-color: #900;
}
div:target {
  z-index: 1000;
}
```

NTH-CHILD SELECTOR

Com o seletor nth-child do CSS3 nós podemos aplicar uma formatação a diferentes elementos de acordo com um determinado intervalo.

CSS

```
div:nth-child(odd) {
```

```
background-color: #900;
}

div:nth-child(even) {
  background-color: #333;
}
```



CSS

```
div:nth-child(3n) {
  background-color: #900;
}
```



CSS

```
div:nth-child(3) {
  background-color: #900;
}
```



PARA SABER MAIS

- [Guia completo dos Seletores CSS](#)
- [10 Seletores de CSS Que Você Deveria Usar](#)
- [:nth Tester](#)

EXERCÍCIOS

<!--Crie uma tabela zebra com o seletor nth-child.-->

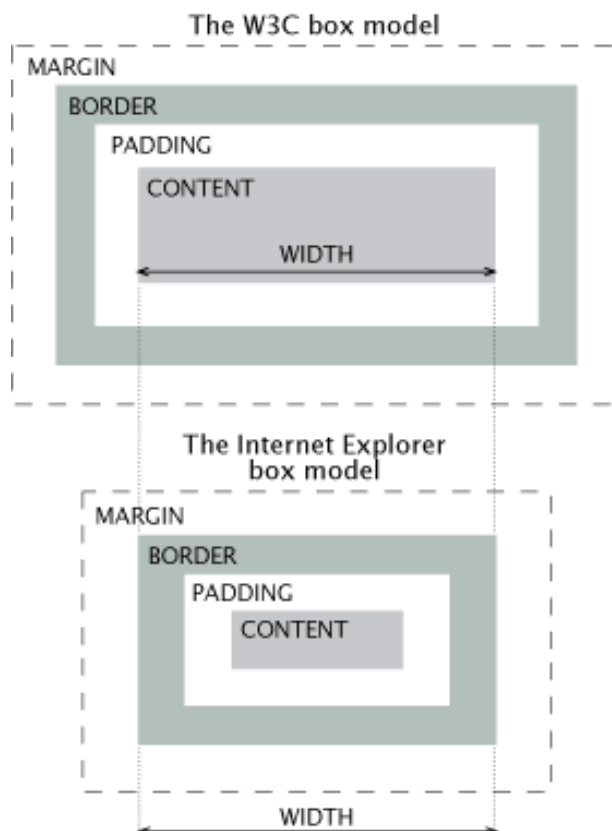
6- PROPRIEDADES CSS

CSS BOX MODEL

Todo elemento no HTML é uma caixa. Controlamos sua largura com width, sua altura com height, sua borda com border e ainda temos as margens externas e internas com margin e padding. Box model é como todas essas propriedades se relacionam pra determinar o tamanho final do elemento.

CSS

```
div {
  width: 200px;
  padding: 20px;
  border: 20px solid gray;
  margin: 10px;
}
```



POSICIONAMENTO CSS

Existem quatro tipos de posicionamento possíveis para as DIVs:

1. Estático (static)
2. Relativo (relative)
3. Absoluto (absolute)
4. Fixo (fixed)

O posicionamento estático corresponde ao fluir normal dos elementos num documento HTML, ou seja, as DIVs são posicionadas de cima para baixo, da esquerda para a direita. Se uma DIV for reposicionada através de css, a DIV imediatamente abaixo toma o seu lugar.

O posicionamento relativo é um reposicionamento de uma dada DIV relativamente à sua posição estática, ou seja, relativamente à sua posição estática a DIV está deslocada no eixo x tantos pixels e no eixo y outros tantos pixels.

O posicionamento absoluto é calculado tendo em conta a janela do browser. Neste caso as coordenadas x e y da DIV estão em perfeita correlação com as dimensões da janela do browser.

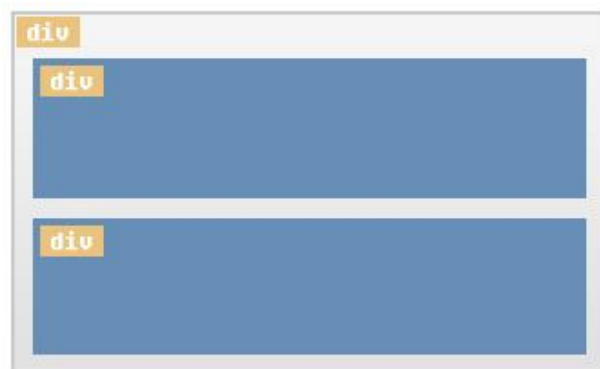
O posicionamento fixo permite manter uma DIV fixa na janela do browser, que não mexe com o movimento do Scroll. Permite-nos criar efeitos como os realizados com frames.

CSS FLOAT

O float é uma propriedade por vezes imprevisível no CSS. Observaremos passo-a-passo o funcionamento da propriedade em elementos de caixa. Inicialmente obtemos uma estrutura HTML simples com três tags Div:

HTML

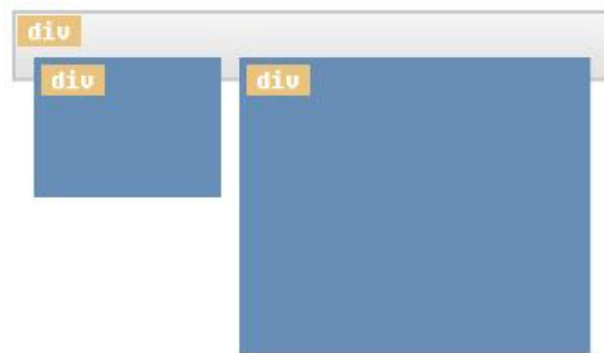
```
<div class="tudo">
  <div class="menu">Menu</div>
  <div class="conteudo">Conteúdo</div>
</div>
```



O Float é uma propriedade que faz o objeto flutuar à esquerda ou à direita do restante do conteúdo. Se quisermos o menu à esquerda e o conteúdo à direita, o que os desenvolvedores geralmente fazem é:

CSS

```
.menu { float: left; width: 30%; }
.conteudo { float: right; width: 60%; }
```



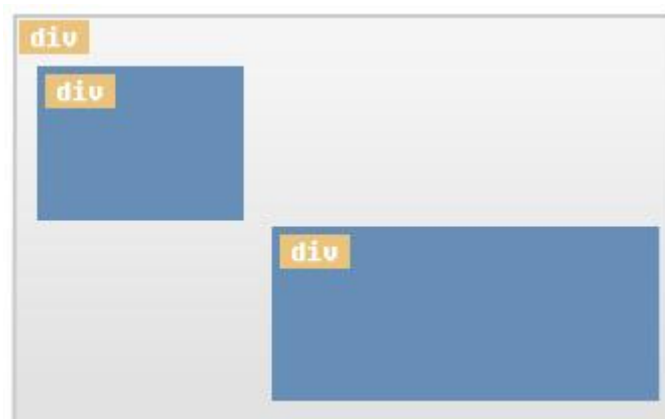
Mas isso pode gerar uma série de efeitos colaterais no desenvolvimento. Vamos compreender o que pode dar errado. Esta técnica gera, principalmente no Internet Explorer, alguns erros de renderização. Note que a borda do div pai que chamamos de "tudo" não está acompanhando os divs internos (.menu e .conteudo).

Para forçar o div "tudo" a circular também os objetos com float, um hack que pode funcionar é adicionar uma div com clear: both; depois de todos os divs flutuantes.

HTML

```
.clear { clear: both; }
<div class="clear"> </div>
```

Agora que o float está integrado com o restante do conteúdo é necessário alinhar "Float:right" com o "Float:left". Em alguns casos, as colunas não ficam uma do lado da outra por mais que você tente.



Para corrigir isso é necessário apenas flutuar a esquerda e permitir o da direita alinhar naturalmente, com um espaço de margem. Veja como é simples:

CSS

```
.menu { float: left; width: 30%; }
.conteudo { margin-left: 30%; }
```

Veja o código combinado de HTML e CSS:

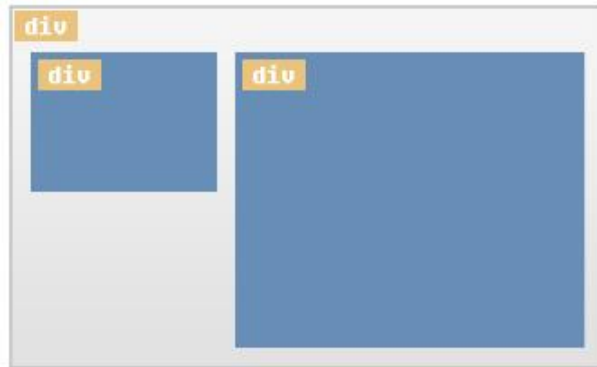
HTML

```
.clear { clear: both; }
```

```
<div class="clear"> </div>
```

CSS

```
.tudo { }
.menu { float: left; width: 30%; }
.conteudo { margin-left: 35%; }
.clear { clear: both; }
```



BACKGROUND-COLOR

A propriedade "background-color" especifica a cor de fundo de um elemento. No exemplo aplicamos uma cor de fundo no elemento body. Podemos então dizer que no exemplo a propriedade background-color determina a cor de renderização do seletor body que esta associado a marcação HTML <body> </body>.

CSS

```
body {
  background-color: #b0c4de;
}
```

BACKGROUND-IMAGE

A propriedade background-image especifica uma imagem para usar como pano de fundo de um elemento.

Por padrão, a imagem é repetida para que ele cobre todo o elemento, mas podemos definir como ela se repete ou mesmo fazer com que a imagem não se repita, usando a propriedade background-repeat, e os valores repeat-x, repeat-y ou no-repeat.

CSS

```
body {
  background-image: url("paper.gif");
  background-repeat: repeat-x;
}
```

Pode também ser utilizar uma codificação reduzida, sendo a ordem dos valores das propriedades o equivalente a:

- 1º background-color
- 2º background-image
- 3º background-repeat
- 4º background-attachment
- 5º background-position

Não sendo necessário preencher todos os valores, mas os valores inseridos vão ser interpretados pelo navegador nesta ordem.

CSS

```
body {  
  background: #ffffff url("paper.gif") no-repeat right top;  
}
```

COLOR

A propriedade color é utilizada para definir a cor do texto. No CSS as cores são mais frequentemente especificadas por:

- Valor HEX (hexadecimal) - como "# FF0000"
- Valor RGB (red green blue)- como "rgb (255.0.0)"
- Nome de uma cor - como "blue"

O CSS3 especifica novas formas de manipulação de cores:

- RGBA - Vermelho, Verde, Azul, Opacidade.
- HSL - Tom, Saturação, Luminosidade.
- HSLA - Tom, Saturação, Luminosidade, Opacidade.
- Opacity – Opacidade nos permite tornar o objeto CSS transparente.

Atualmente o RGBA, HSL, HSLA e Opacity são suportados apenas nas versões mais recentes do Firefox, Opera Safari e Chrome. Não existem soluções para o Internet Explorer ou versões antigas de outros navegadores.

CSS

```
body {  
  color: blue;  
}  
  
h1 {  
  color: #00ff00;  
}  
  
h2 {  
  color: rgb(255,0,0);  
}
```

O conjunto de cores RGB são três números que começam no 0 e vão até 255 (0% até 100%), onde o primeiro bloco define a quantidade de vermelho (Red), o segundo bloco a quantidade

de verde (Green) e o último bloco a quantidade de azul (Blue). A combinação destes números formam todas as cores que você pode imaginar.

No HTML o RGB pode ser usado em qualquer propriedade que tenha a necessidade de cor, como: color, background, border etc. Este código RGB define que o background o elemento P será amarelo.

CSS

```
p {  
  background: rgb(255,255,0);  
  padding: 10px;  
  font: 13px verdana;  
}
```

OPACITY

O CSS3 nos trouxe a possibilidade de modificar a opacidade dos elementos via propriedade opacity. Lembrando que quando modificamos a opacidade do elemento, tudo o que está contido nele também fica opaco e não apenas o background ou a cor dele. Perceba que o background e também a cor do texto ficaram transparentes.

CSS

```
p {  
  background: rgb(255,255,255);  
  padding: 10px;  
  font: 13px verdana;  
  opacity: 0.5;  
}
```



Útil mas dificulta muito quando queremos que apenas a cor de fundo de um determinado elemento tenha a opacidade modificada. É aí que entra o RGBA. O RGBA funciona da mesma forma que o RGB, ou seja, definindo uma cor para a propriedade. No caso do RGBA, além dos três canais RGB (Red, Green e Blue) há um quarto canal, A (Alpha) que controla a opacidade da cor. Nesse caso, podemos controlar a opacidade da cor do background sem afetar a opacidade dos outros elementos.

CSS

```
p {
```



```
background: rgba(255,255,255, 0.5);  
padding: 10px;  
font: 13px verdana;  
}
```



SHADOW

Uma das vantagens mais interessantes que o CSS3 nos dá é a possibilidade de cada vez menos abrimos o Photoshop. Agora temos a possibilidade de inserirmos sombras em textos e em elementos. As propriedades têm nomes diferentes, mas a mesma sintaxe.

CSS

```
p {  
  text-shadow: 5px 5px 5px rgba(0,0,0,0.5);  
  box-shadow: 5px 5px 5px rgba(0,0,0,0.5);  
}
```

Agora vamos entender o significado dos números: os dois primeiros números se referem a posição da sombra: o primeiro número é referente a posição vertical começando pelo topo e o segundo número é referente a posição horizontal, começando pela esquerda.

Utilizando números positivos os dois primeiros valores posicionam a sombra para a direita do texto, a coordenada X horizontalmente e a coordenada Y colocando a sombra abaixo do texto verticalmente.

O terceiro número se refere ao Blur. Sua sombra pode ser rígida ou "esfumaçada". Isso depende do design que você criou o pegou para programar. Você controla a rigidez da sombra por este número.

Você pode usar valores RGBA, em vez de cores código hexadecimal como o quarto valor. O último número representa a transparência e pode ser ajustado entre 0.0 e 1.0 para que você possa fazer o texto/elemento de sombra mais sutil e suave.

O uso de múltiplas sombras parece incrível, e alcançamos esse efeito apenas colocando novamente a propriedade text-shadow/box-shadow separadas por vírgulas:

CSS

```
p {  
  color: #000;
```

```
padding: 10px;
text-shadow: 0 0 4px #ccc,
            0 -5px 4px #ff3,
            2px -10px 6px #fd3,
            -2px -15px 11px #f80,
            2px -18px 18px #f20;
}
```

BORDER-IMAGE

Ainda está em fase de testes pelos browsers, a sintaxe do border-image se divide em três partes:

1. URL da imagem que será utilizada.
2. Tamanho do slice das bordas.
3. Como o browser irá aplicar a imagem na borda.

CSS

```
#box_image {
border-width: 10px 15px 12px 13px;
-webkit-border-image: url(border.png) 30% 35% 40% 30% round round;
}
```

A primeira propriedade, obviamente, descreve um elemento de borda, como de costume: a largura das quatro bordas é expressa em pixels, no sentido horário (top, border-right, border-bottom, border-left). A segunda propriedade é a que vai realmente fatiar a imagem. Normalmente existem três propriedades border-image.

Usaremos a imagem que será cortada para decorar o nosso elemento.



A segunda parte descreve a maneira como a imagem será cortada em nove partes (fatias). Esses valores podem ser expressos em percentuais, ou seja, sem qualquer unidade de largura em pixels.



A última parte é o valor de alongamento. Ele descreve como as fatias para os lados e na parte média são escalados e azulejos. Especificando somente um valor é o mesmo que especificar o dobro do mesmo lado.

Há três valores possíveis: "stretch", "repeat" and "round". O valor padrão é "stretch".



BORDER-RADIUS

Usando CSS3, criar uma borda arredondada é incrivelmente fácil. Ela pode ser aplicada a cada esquina ou cantos individuais, e a largura/cor é facilmente alterada. Podem-se definir valores variados dependendo do resultado desejado, é possível se arredondar as bordas até se formar um círculo, deixando a altura e largura com tamanhos iguais.

CSS

```
div {  
  border-radius: 20px;  
}
```

As bordas arredondadas não ficam restritas ao elemento elas podem ser usadas em botões, campos de formulário, parágrafos, caixas de destaque em uma página de textos, entre outros.

DEIXE UM COMENTÁRIO

Nome (obrigatório)

E-mail (obrigatório)

Site

Comentário

Enviar

GRADIENTES

Todos os browsers mais novos como Safari, Opera, Firefox e Chrome já aceitam essa feature e você pode utilizá-la hoje. O browser Internet Explorer não reconhece ainda, contudo você poderá utilizar imagens para estes browsers que não aceitam essa feature.

CSS

```
div {  
  width: 200px;  
  height: 200px;  
  background-color: #FFF;  
  background-image: -moz-linear-gradient(green, red);      /* Firefox 3.6+ */  
  background-image: -webkit-linear-gradient(green, red); /* Safari 5.1+, Chrome 10+ */  
  background-image: -o-linear-gradient(green, red);      /* Opera 11.10+ */  
}
```

O padrão é que o gradiente ocupe 100% do elemento como vimos no exemplo anterior, mas muitas vezes queremos fazer apenas um detalhe.

Nesse caso nós temos que definir um STOP, para que o browser saiba onde uma cor deve terminar para começar a outra. Perceba o 20% ao lado da cor vermelha. O browser calcula quanto é 20% da altura (ou largura dependendo do caso) do elemento, e começa o gradiente da cor exatamente ali. Você ainda pode criar códigos de gradientes online usando ferramentas grátis.

CSS

```
div {  
  width: 200px;  
  height: 200px;  
  background-color: #FFF;  
  background-image: -moz-linear-gradient(green, red 20%); /* Firefox 3.6+ */  
  background-image: -webkit-linear-gradient(green, red 20%); /* Safari 5.1+, Chrome 10+ */  
  background-image: -o-linear-gradient(green, red 20%); /* Opera 11.10+ */  
}
```

CURRENTCOLOR

O valor `currentColor` é muito simples de se entender e pode ser muito útil para diminuirmos o retrabalho em alguns momentos da produção. Imagine que o `currentColor` é uma variável cujo seu valor é definido pelo valor da propriedade `color` do seletor. Veja o código

CSS

```
p {  
  background: red;  
  padding: 10px;  
  font: 13px verdana;  
  color: green;  
  border: 1px solid green;  
}
```

Note que o valor da propriedade `color` é igual ao valor da cor da propriedade `border`.

Há uma redundância de código, que nesse caso é irrelevante, mas quando falamos de dezenas de arquivos de CSS modulados, com centenas de linhas cada, essa redundância pode incomodar a produtividade. A função do `currentColor` é simples: ele copia para outras propriedades do mesmo seletor o valor da propriedade `color`. Veja o código abaixo para entender melhor:

CSS

```
p {  
  background: red;  
  padding: 10px;  
  font: 13px verdana;  
  color: green;  
  border: 1px solid currentColor;  
}
```

Veja que aplicamos o valor `currentColor` onde deveria estar o valor de cor da propriedade `border`. Agora, toda vez que o valor da propriedade `color` for modificado, o `currentColor` aplicará o mesmo valor para a propriedade onde ela está sendo aplicada.

Isso funciona em qualquer propriedade que faça utilização de cor como `background`, `border`, etc. Obviamente não funciona para a propriedade `color`. O `currentColor` também não funciona em se-letores separados, ou seja, você não consegue atrelar o valor da propriedade `color` ao `currentColor` de outro seletor.

PARA SABER MAIS

- [CSS grid: isso muda tudo de novo](#)
- [CSS float: considerações, dicas e macetes para bons layouts na web](#)
- [Design Responsivo na prática: do rascunho ao digital](#)
- [CSS: Você deveria usar box-sizing: border-box em todas as suas páginas](#)
- <http://www.colorzilla.com/gradient-editor/>

EXERCÍCIOS

<!--Formate o formulário criado no exercício de HTML semântico. Usando CSS, aqui você pode pesquisar na web um modelo e tentar reproduzir.-->

7- FORMATANDO TEXTO CSS

LINKS

São quatro as pseudo classes para links:

1. **a:link**.....define o estilo do link no estado inicial;
2. **a:visited**...define o estilo do link visitado;
3. **a:hover**.....define o estilo do link quando passa-se o mouse sobre ele;
4. **a:active**.....define o estilo do link ativo (o que foi "clicado").

É importante a ordem de definição das regras para os estados dos links, quando duas ou mais regras são conflitantes aquela ocupando uma posição na folha de estilo é sobrescrita ("perde para") um regra posterior a ela na folha. Assim, se define a:hover ANTES de a:visited, esta prevalecerá sobre a:hover, em consequência o link visitado pela primeira vez assumirá a regra definida em a:visited e a partir de então a:hover não mais funcionará naquele link pois a:visited prevalecerá sobre a:hover.

CSS

```
a:link {  
  color: #FF0000;  
  text-decoration: none }  
  
a:visited { color: #00FF00; }  
  
a:hover {  
  color: #FF00FF;  
  text-decoration: none }  
  
a:active { color: #0000FF; }
```

TEXT-ALIGN

A formatação de texto em documentos HTML é semelhante a editoração eletrônica de um livro. A estruturação semântica do texto é criada primeiramente em HTML e então são aplicados os estilos em CSS.

A propriedade text-align é usada para definir o alinhamento horizontal de um texto. O texto pode ser centralizado ou alinhado à esquerda ou à direita ou justificado.

CSS

```
h1 { text-align: center; }  
  
p.data { text-align: right; }  
  
p.texto { text-align: justify; }
```

TEXT-DECORATION

A propriedade "text-decoration" é usada para definir ou remover decorações no texto.

CSS

```
a {  
  text-decoration: none;  
}  
  
h1 {  
  text-decoration: overline;  
}  
  
h2 {  
  text-decoration: line-through;  
}  
  
h3 {  
  text-decoration: underline;  
}
```

TEXT-TRANSFORMATION

Esta propriedade pode ser usada para transformar tudo em letras maiúsculas ou minúsculas, ou capitalizar a primeira letra de cada palavra.

CSS

```
p.maiusculo {  
  text-transform: uppercase;  
}  
  
p.minusculo {  
  text-transform: lowercase;  
}  
  
p.capitalizado {  
  text-transform: capitalize;  
}
```

FONT-FAMILY

No CSS a propriedade de fonte define a família das fontes, negritos, tamanho e estilo de um texto. Existem dois tipos de nomes para família de fontes:



1. Generic family - um grupo de famílias de fontes com um olhar similar (como "Serif" ou "Monospace")
2. Font family - uma família de fontes específicas (como "Times New Roman" ou "Arial")

A família da fonte de um texto é definida com a propriedade font-family. A propriedade font-family deve conter vários nomes de fontes como um sistema de "fallback". Se o navegador não suportar a primeira fonte, ele vai tentar a próxima fonte.

Sempre comece com a fonte que você quer, e termine com uma família genérica, assim você vai deixar o navegador escolher uma fonte semelhante na família genérica, parecida com as que você definiu primeiramente caso nenhuma destas fontes estejam disponíveis no dispositivo do usuário.

Se o nome de uma família de fonte é mais do que uma palavra, ele deve estar entre aspas, como: "Times New Roman" e quando há mais do que uma família de fontes, devemos especificar cada uma delas separada por vírgulas.

CSS

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

FONT-STYLE

A propriedade "font-style" é usada principalmente para especificar texto em itálico e possui três valores:

1. Normal - O texto é exibido normalmente
2. Itálico - O texto é mostrado em itálico
3. Oblíquo - O texto é "inclinar-se" (oblíqua é muito semelhante ao itálico, mas menos utilizado e suportado).

CSS

```
p.normal { font-style: normal; }  
  
p.italico { font-style: italic; }  
  
p.obliquo { font-style: oblique; }
```

FONT-SIZE

A propriedade "font-size" define o tamanho do texto. Ser capaz de gerenciar o tamanho do texto é importante no design para web. No entanto, você não deve usar essa propriedade para fazer parágrafos se parecerem como títulos, ou cabeçalhos se parecem com parágrafos. Neste caso, utilize sempre as tags HTML adequadas para isso, como o <h1></h1> até o <h6> </h6> para títulos e <p> </p> para parágrafos. O valor da font-size pode ser um tamanho absoluto ou relativo.

O tamanho absoluto define o texto para um tamanho especificado e não permite que o usuário altere o seu tamanho em todos os navegadores (isto se torna ruim por questões de

acessibilidade). Tamanho absoluto é útil quando o tamanho físico de impressão (na tela) da fonte é conhecido. Definir o tamanho com pixels lhe dá controle total sobre o tamanho do texto

CSS

```
h1 {  
  font-size: 40px;  
}
```

O tamanho relativo define o tamanho em relação a elementos e permite que o usuário altere o tamanho do texto nos navegadores. Se você não especificar um tamanho de fonte, o tamanho padrão para texto normal, como parágrafos, é 16px (16px = 1em).

Para permitir que os usuários possam redimensionar o texto (no menu do navegador), muitos desenvolvedores usam "em" em vez de pixels. 1em é igual ao tamanho da fonte de corrente. O tamanho padrão do texto nos navegadores é 16px. Assim, o tamanho padrão de 1em é 16px. O tamanho pode ser calculado a partir de pixels para "em" usando esta fórmula: pixels / 16 = EM.

CSS

```
h1 {  
  font-size: 2.5em; /* 40px/16=2.5em */  
}
```

@FONT-FACE

A regra @font-face é utilizada para que você utilize fontes fora do padrão do sistema em seus sites. Para que isso funcione, é necessário disponibilizar as fontes necessárias em seu servidor e linkar estas fontes no arquivo CSS. A sintaxe é bem simples e tem suporte a todos os navegadores, com algumas ressalvas.

CSS

```
@font-face {  
  font-family: bebasneue;  
  src: local(bebasNeue.otf),  
       url(bebasNeue.otf);  
}
```

Na primeira linha você customiza o nome da font que você usará durante todo o projeto. Na segunda linha definimos a URL onde o browser procurará o arquivo da font para baixar e aplicar no site. Suponha que o usuário tenha a font instalada, logo ele não precisa baixar a font, assim podemos indicar para que o browser possa utilizar o arquivo da própria máquina do usuário.

Formatos que podem ser usados e que os browsers podem adotar:

| STRING | FONT FORMAT | COMMON EXTENSIONS |
|-------------------|----------------------|-------------------|
| truetype | TrueType | .ttf |
| opentype | OpenType | .ttf, .otf |
| embedded-opentype | Embedded OpenType | .eot |
| woff | Web Open Font Format | .woff |
| svg | SVG Font | .svg, .svgz |

Você aplica a fonte como segue:

CSS

```
p {  
  font: 36px bebasneue, Arial, Tahoma, Sans-serif;  
}
```

TEXT-OVERFLOW

Alguns conteúdos em um elemento podem sair caixa de renderização do elemento por uma série de razões (margens negativas, o posicionamento absoluto, o índice superior a largura / altura definida por um elemento, etc). Nos casos em que isso ocorre, a propriedade text-overflow traz uma alternativa, o texto extra se mostra sem problemas. Os valores possíveis da propriedade são:

- text-overflow: ellipsis - O conteúdo do texto que transbordará, é mostrado com a sequência de "...".
- text-overflow: clip - O conteúdo visível para a área definida pela caixa de processamento, escondendo o restante que ficar fora dos limites da caixa, comportamento similar ao overflow: hidden.

Para que funcione em todos os browsers é necessário que o elemento tenha largura fixa, tenha a propriedade white-space:nowrap e overflow:hidden.



Esse é um exemplo de

CSS

```
text_overflow {  
  white-space: nowrap;  
  overflow: hidden;  
  text-overflow: ellipsis;  
  width: 200px;  
  height: 20px;  
  border: 1px solid #000;  
}
```

PARA SABER MAIS

- [Guia de Referência CSS](#)

EXERCÍCIOS

<!--

Construa o seguinte documento HTML semântico e formate visualmente com CSS.

Usuário: Admin













































25/01/2007 19:32:00 PM

XZ News

Sair

DESTINATÁRIOS | NEWSLETTER | NEWSLETTER URGENTE

Novo Usuário

| ID | Nome do Usuário | E-mail do Usuário | Data de Criação | |
|----|-----------------|--------------------|-----------------|--|
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |
| 1 | Pedro Bó | pedrobo@net.com.br | 29/09/2006 |   <input type="checkbox"/> |
| 2 | João Ninguém | johndoe@web.com | 22/08/2006 |   <input type="checkbox"/> |

Ativar Todos

Desativar Todos

Excluir Seleccionados

© 2006. Todos os direitos reservados.

-->

8- CSS TRANSFORM E TRANSITION

O módulo Transform permite que elementos renderizados em css sejam transformados em 2D ou 3D através de comandos específicos. A propriedade CSS3 Transform nos permite alterações lineares em elementos HTML, incluindo rotação, inclinações e alteração da escala.

Existem duas propriedades principais nas transformações CSS: transform, que define as alterações que serão feitas no elemento; transform-origin, que define a posição de origem do elemento, ou eixo de rotação.

TRANSLATE

A função translate é usada para movermos um elemento nos eixos X, Y, ou ambos. Caso seja passado apenas um parâmetro para a função, o elemento será movido em relação ao eixo X. Já com dois parâmetros, o primeiro será o valor usado para a movimentação no eixo X, e o segundo para o eixo Y.

CSS

```
.div-teste {  
  transform: translate(13px, 10px); /* move elemento 13px em X e 10px em Y */  
  transform: translate(13px); /* move elemento 13px no eixo X */  
  transform: translateX(13px); /* move elemento 13px no eixo X */  
  transform: translateY(10px); /* move elemento 10px no eixo Y */  
}
```

Além de px, também podemos passar valores expressos em %. Neste caso, 100% será equivalente a dimensão do elemento no eixo em que vamos efetuar o translate. Em outras palavras: se vamos movimentar o elemento no eixo X, 100% será igual a sua largura; se a movimentação ocorrer no eixo Y, 100% equivalerá a altura do elemento. No experimento abaixo vocês podem ver a função translate em ação de uma forma mais interativa e menos abstrata, basta mover os handlers dos inputs do Translate X e Translate Y.

Reparem que valores negativos também são aceitos. Valores negativos em X movimentam o elemento para a esquerda. Valores negativos em Y movimentam o elemento para cima. Um transform: translateX(-10px); moverá o elemento 10 pixels para a esquerda. Enquanto um transform: translateY(-5px); moverá o elemento 5 pixels para cima.

SKEW

A função skew irá "entortar" um elemento em relação a um dos eixos.

CSS

```
.div-teste {  
  transform: skewX(30deg); /* "entorta" o elemento horizontalmente */  
  transform: skewY(30deg); /* "entorta" o elemento verticalmente */  
}
```

As funções usadas para efetuarmos o skew são skewX e skewY. Cada uma recebe apenas um parâmetro, estes expressos em graus (deg).

ROTATE

A função rotate rotacional um elemento em seu próprio eixo. Recebe um único parâmetro que pode ser expresso em graus (deg), radianos (rad) ou voltas (turn).

CSS

```
.div-teste {  
  transform: rotate(30deg); /* rotaciona 30' no sentido horário */  
  transform: rotate(-45deg); /* rotaciona 45' no sentido anti-horário */  
  transform: rotate(2rad); /* rotaciona ~ 144,59' no sentido horário */  
  transform: rotate(0.5turn); /* rotaciona meia-volta no sentido horário */  
}
```

Valores positivos rotacionam o elemento no sentido horário, enquanto que valores negativos rotacionam o elemento no sentido anti-horário. O experimento abaixo mostra o rotate em ação.

SCALE

Altera o tamanho do elemento em ambos os eixos ou apenas em um eixo específico. Esta função aceita um ou dois parâmetros. Caso seja passado apenas um, a alteração irá ocorrer tanto na altura como na largura. Caso sejam passados dois, o primeiro argumento irá alterar a largura e o segundo irá alterar a altura.

Os valores passados como parâmetros não possuem unidades. Eles funcionam como “multiplicadores” do tamanho do elemento. Um transform: scale(2) irá duplicar o tamanho do elemento. Um transform: scale(0.5) irá reduzi-lo pela metade.

CSS

```
.div-teste {  
  transform: scale(2); /* dobra o tamanho do elemento em todas as direções */  
  transform: scale(2, 3); /* dobra a largura e triplica a altura */  
  transform: scaleX(0.5); /* reduz largura do elemento pela metade */  
  transform: scaleY(2); /* deixa o elemento duas vezes mais alto */  
}
```

Além da função scale, podemos usar as funções específicas de cada eixo: para o eixo X a scaleX e para o eixo Y a scaleY. Abaixo também podemos interagir com esta função e ver na prática como se comporta.

Observe que, ao passarmos valores negativos, nós invertemos a imagem. Se passarmos um valor de -1 para scaleX, é como se estivéssemos espelhando o elemento. -1 para scaleY e o elemento irá ficar de cabeça para baixo (invertido na vertical).

CSS TRANSITIONS

Mostrar um feedback ao usuário quando ele passar o mouse sobre um link (:hover) ou quando ele der foco em um campo input (:focus) são boas práticas. Há muitos jeitos de fazermos isso:

mudando a cor do texto, tirando o underline da palavra, alterando a borda do input que recebeu o foco ou até alterando a cor de background do elemento. Normalmente fazemos isso alterando o valor de uma propriedade CSS no estado :hover ou :focus do elemento. Quando fazemos isso, o resultado é instantâneo. Melhor dizendo, a alteração ocorre imediatamente ao usuário fazer a ação (seja passar o mouse sobre o elemento ou este ganhar foco). A alteração ocorre de forma brusca, do valor antigo da propriedade para o novo valor. Por exemplo, quando temos um elemento com borda colorida, e utilizamos o estado :hover para alterarmos a cor da borda, essa transição entre uma cor e outra acontece imediatamente assim que você passa o mouse em cima do elemento.

Como o CSS está implementado em todos os grandes navegadores e seu uso é comum podemos adicionar um efeito quando o navegador troca de um estilo para outro, sem usar animações em Flash ou JavaScript. É aí que entra a transition do CSS3. Ela analisa a mudança de valor entre a propriedade e faz com que essa transição, ao invés de ocorrer de forma brusca, ocorra suavemente em um tempo determinado. A propriedade transition possui quatro propriedades para você configurar:

1. **transition-property:** Nome da propriedade CSS sobre a qual o efeito da transição vai ser aplicado, caso seja omitida, não existirá uma propriedade para se aplicar o efeito da transição. É possível aplicar uma mesma transição para todas as propriedades CSS do elemento usando o valor: "all".
2. **transition-duration:** Duração do efeito em segundos (o padrão é 0). Obrigatória na declaração pois, se omitida, assume valor default zero e a transição não vai ter efeito.
3. **transition-timing-function:** Forma como a transição progride no tempo (o padrão é ease), em outras palavras, como se comporta o ritmo da transição durante o efeito.
4. **transition-delay:** Define a partir de quanto tempo (em segundos) o efeito da transição vai se iniciar (o padrão é 0).

CSS

```
.classname {  
  transform: rotate(4deg) scale(1) skew(1deg) translate(10px);  
  transition-delay: 0s;  
  transition-property: all;  
  transition-duration: 1s;  
  transition-timing-function: ease;  
}  
  
.classname:hover {  
  transform: rotate(0deg) scale(0.67) skew(1deg) translate(0px);  
}
```

A tabela relaciona propriedades que podem "sofrer" transforms e transitions.

| PROPERTY NAME | TYPE |
|---------------------|--|
| background-color | as color |
| background-position | as repeatable list of simple list of length, percentage, or calc |
| border-color | as color |
| border-width | as length |
| border-spacing | as simple list of length |
| bottom | as length, percentage, or calc |

| | |
|----------------|--------------------------------|
| clip | as rectangle |
| color | as color |
| font-size | as length |
| font-weight | as font weight |
| height | as length, percentage, or calc |
| left | as length, percentage, or calc |
| letter-spacing | as length |
| line-height | as either number or length |
| margin | as length |
| max-height | as length, percentage, or calc |
| max-width | as length, percentage, or calc |
| min-height | as length, percentage, or calc |
| min-width | as length, percentage, or calc |
| opacity | as number |
| outline-color | as color |
| outline-width | as length |
| padding | as length |
| right | as length, percentage, or calc |
| text-indent | as length, percentage, or calc |
| text-shadow | as shadow list |
| top | as length, percentage, or calc |
| vertical-align | as length |
| visibility | as visibility |
| width | as length, percentage, or calc |
| word-spacing | as length |
| z-index | as integer |

PARA SABER MAIS

- [Using CSS transitions](#)

EXERCÍCIOS

<!--

Implemente uma transição de tamanho e opacidade em uma imagem.

-->

9- CSS ANIMATION

Uma animação permite que um elemento mude seu estilo inicial para outro estilo. Você pode mudar muitas propriedades CSS que você queira e quantas vezes você quiser. É possível especificar quando a mudança irá acontecer em porcentagens, ou podemos usar as palavras chaves "from" (de) e "to" (para) que representam 0% e 100%. 0% representa o início da animação e 100% é quando a animação está completa.

Com o CSS3 podemos criar animações que podem substituir as animações em flash, imagens e JavaScript nas páginas existentes. Utilizamos inicialmente duas propriedades para criar uma animação:

1. **@keyframes** - local onde a animação é criada
2. **animation** - define o seletor da animação que será usada a qual foi criada pela regra keyframes

Quando uma animação é criada no keyframe, é necessário vincular isto em um seletor, caso contrário, a animação não terá efeito algum.

Para incular a animação de um seletor (elemento), especificamos pelo menos o nome da animação e a duração da animação.

A sintaxe para a criação de keyframes é a seguinte:

CSS

```
@keyframes nomedaanimacao {  
  seletores-keyframe {  
    estilo css para esse determinado keyframe;  
  }  
}
```

Existem duas forma de se criar KeyFrames, a forma mais básica, onde definimos um início e um fim para a animação. No exemplo **"from"** é equivalente ao início da animação (0%) e **"to"** é equivalente ao final da animação (100%).

CSS

```
@keyframes animacao {  
  from {  
    width: 100px;  
    background: black;  
  }  
  
  to {  
    background: yellow;  
    width: 200px;  
  }  
}
```

Desta forma há maior controle da animação. Para isso, utilizamos porcentagem para definir os keyframes. No código, a animação possui cinco passos e, a porcentagem é relativa à duração da animação que vai ser definida posteriormente.

Com a animação criada nos keyframes, precisamos vinculá-la a algum seletor, caso contrário a animação não terá nenhum efeito. Para fazer isso, temos que declarar pelo menos duas propriedades que são obrigatórias:

1. O nome da animação (igual ao especificado nos keyframes);
2. A duração da animação (se não for declarada, a animação não se inicia, pois o valor padrão é zero).

CSS

```
@keyframes animacaoBolada {  
  0% {  
    background: black;  
    width: 100px; }  
  
  25% { background: green; }  
  
  50% { background: blue; }  
  
  75% { background: red; }  
  
  100% {  
    background: yellow;  
    width: 200px; }  
}
```

PROPRIEDADES CSS ANIMATIONS

animation-name: Nome da animação especificada nos @keyframes;

CSS

```
animation-name: animacaoBolada;
```

animation-duration: Quanto tempo, em segundos ou milissegundos, durará um ciclo da animação (o padrão é 0).

CSS

```
animation-duration: 5s;
```

animation-timing-function: Forma como a animação progride no tempo (o padrão é ease). Do mesmo modo que a propriedade transition, pode ser usada de duas maneiras: utilizando alguns valores já pré-definidos: linear, ease, ease-in, ease-out e ease-in-out;

CSS

```
animation-timing-function: ease;
```

Definindo uma função customizada, especificando quatro coordenadas para definir a cubic bezier curve:

CSS

```
animation-timing-function: cubic-bezier(0.005, 0.0625, 0.365, 0.0840);
```

animation-delay: Define a partir de quanto tempo a animação vai se iniciar (o padrão é 0).

CSS

```
animation-delay: 0.2s;
```

animation-iteration-count: Determina o número de vezes que a animação vai se repetir (o padrão é 1). Podemos deixar a animação repetindo infinitamente, basta especificar o valor infinite.

CSS

```
animation-iteration-count: infinite;
```

animation-direction: Especifica se ao final da animação, ela deve reiniciar seu fluxo normalmente (normal), que é o padrão, ou voltar no sentido inverso (reverse).

CSS

```
animation-direction: reverse;
```

animation-play-state: Define se a animação está rodando (running), que é o padrão, ou pausada (paused).

CSS

```
animation-play-state: running;
```

Após definir as propriedades temos a animação montada:

CSS

```
div {  
  animation-name: animacaoBolada;  
  animation-duration: 5s;  
  animation-timing-function: ease;  
  animation-delay: 1s;  
  animation-iteration-count: infinite;  
  animation-direction: alternate;  
  animation-play-state: running;  
}
```

ANIMATIONS VS TRANSITIONS

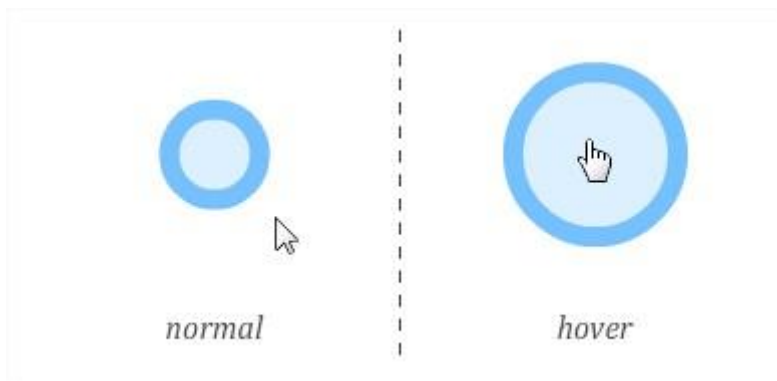
Em CSS3, nós temos duas técnicas para visualização de mudança que estão competindo por nossa atenção: Animações e Transições. De longe, as duas animações e transições são muito semelhantes. Ambos permitem que você:

- ✓ Especifique quais propriedades CSS serão alteradas
- ✓ Especifique a duração da animação ou transição.
- ✓ Especificar animação e transição específicas de eventos
- ✓ Visualizar as alterações das propriedades CSS

Porém, você vai perceber que as animações e transições divergem um pouco em sua singularidade. Animações e transições mostram suas diferenças quando se trata de acionar seus efeitos, como definir uma transição, um loop ou como ambas trabalham com JavaScript.

Uma das principais diferenças entre as animações e transições pode ser visto na forma como nós definimos o disparo do efeito.

Uma transição somente funciona como uma reação para uma propriedade CSS que foi alterada. Um cenário comum é aquele em que você usa a pseudo classe `:hover` para alterar o valor de uma propriedade CSS.



Para usar o exemplo visualizado aqui, se uma transição é definida, você seria capaz de ver o círculo crescer de seu tamanho normal para o seu tamanho hover. Outra maneira de disparar uma transição é usar JavaScript adicionando ou removendo classes CSS para simular uma mudança de propriedade CSS (simular a pseudo-classe `:hover`, por exemplo).

Animações, por outro lado, não necessitam de qualquer accionamento explícito. Depois de definir a animação, ela vai começar a tocar automaticamente.

Animações podem facilmente construir um loop configurando a propriedade `animation-iteration-count`. Você pode especificar o número de quantas vezes você quer que a sua animação se repita:

CSS

```
animation-iteration-count: 5;
```

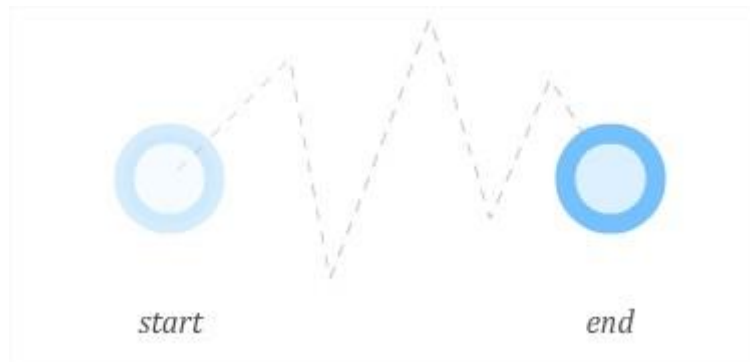
Se você quiser que sua animação tenha uma repetição infinita, você pode fazer isto desta maneira:

CSS

```
animation-iteration-count: infinite;
```

As transições não tem uma propriedade que especifica quantas vezes elas podem rodar. Quando disparada, uma transição roda somente uma vez. Você pode fazer um loop de transição, brincando com o `transitionEnd` event, mas isso não é particularmente simples, especialmente quando comparado com animações.

Com uma animação, você pode ter capacidade de definir keyframes que darão maior controle sobre suas propriedades CSS e seus valores além de apenas início e fim.



Você pode configurar quantos keyframes que você quiser, e quantas animações irá tocar, cada keyframe será o hit que especifica as mudanças que a propriedade irá refletir. Isso permite que você crie tipos de animações que ajudam o HTML5 a competir com as tecnologias de animação mais estabelecidos no mercado, como o Flash de forma mais eficaz. Com uma transição, você não tem muito controle sobre nada além do resultado final.



Uma transição simplesmente vai de um estado inicial para o estado final. Você não pode especificar todos os pontos no meio como você pode com uma animação, por isso, uma transição pode não ser uma boa escolha se você está tentando criar um efeito fadeIn ou uma animação complexa.

PARA SABER MAIS

- [CSS3 Animations](#)
- [CSS 3.0 Maker](#)
- [CSS Transition e CSS Animation](#)
- [Animate.css](#)

10- JAVASCRIPT

Javascript é uma linguagem de programação utilizada para criar pequenos programas encarregados de realizar ações no âmbito de uma página web. Com Javascript podemos criar efeitos especiais nas páginas e definir interatividades com o usuário. O navegador do cliente é o encarregado de interpretar as instruções Javascript e executá-las para realizar estes efeitos e interatividades, de modo que o maior recurso com que esta linguagem conta é o próprio navegador.

Javascript é o seguinte passo, depois de HTML, que pode dar um programador da web que decide melhorar suas páginas e a potência de seus projetos. É uma linguagem de programação bastante simples e pensada para fazer as coisas com rapidez, às vezes com leveza. Inclusive as pessoas que não tenham uma experiência prévia na programação poderão aprender esta linguagem com facilidade e utilizá-la com um pouco de prática.

Entre as ações típicas que se podem realizar em Javascript temos duas vertentes. Por um lado, os efeitos especiais sobre páginas web, para criar conteúdos dinâmicos e elementos da página que tenham movimento, mudam de cor ou qualquer outro dinamismo. Por outro, javascript nos permite executar instruções como resposta às ações do usuário, com o qual podemos criar páginas interativas com programas como calculadoras, agendas, ou tabelas de cálculo.

Javascript é uma linguagem com muitas possibilidades, permite a programação de pequenos scripts, e também de programas maiores, orientados a objetos, com funções, estruturas de dados complexas, etc. Toda esta potência de Javascript se coloca à disposição do programador, que se converte no verdadeiro dono e controlador de cada coisa que ocorre na página.

Que fique claro que Javascript não tem nada a ver com Java, salvo em suas origens. Atualmente são produtos totalmente distintos e não guardam entre si mais relação que a sintaxe idêntica e pouco mais. Algumas diferenças entre estas duas linguagens são:

- ✓ Java é compilado e JavaScript é interpretado
- ✓ JavaScript não é totalmente Orientado a objetos
- ✓ JavaScript não é usado tipicamente em servidores.
- ✓ JavaScript não uma linguagem fortemente tipada.

Para programar em Javascript necessitamos basicamente o mesmo que para programar páginas web com HTML. Um editor de textos e um navegador compatível com Javascript. Um usuário de Windows possui de saída todo o necessário para poder programar em Javascript, visto que dispõe dentro de sua instalação típica de sistema operativo, de um editor de textos, o Bloco de notas, e de um navegador: Google Chrome, Mozilla Firefox ou Internet Explorer.

Uma recomendação em relação ao editor de textos: Apesar de o Bloco de Notas editar os arquivos, é desejável contar com outros programas que nos oferecem melhores ferramentas na hora de escrever as linhas de código. Estes editores avançados têm algumas vantagens: colorem os códigos de nossos scripts, nos permitem trabalhar com vários documentos simultaneamente, etc.

FUNDAMENTOS

Existem duas maneiras de executar scripts na página. A primeira destas maneiras se trata de execução direta de scripts, a segunda é uma execução como resposta à ação de um usuário.

Execução direta - É o método mais básico de executar scripts. Neste caso se incluem as instruções dentro da etiqueta `<script></script>` tal como comentamos anteriormente. Quando o navegador lê a página e encontra um script vai interpretando as linhas de código e vai executando uma depois da outra. Chamamos a esta maneira execução direta, pois quando se lê a página se executam diretamente os scripts.

Resposta a um evento - É a outra maneira de executar scripts, mas antes de vê-la devemos falar sobre os eventos. Os eventos são ações que realiza o usuário. Os programas como Javascript estão preparados para apanhar determinadas ações realizadas, neste caso sobre a página, e realizar ações como resposta. Deste modo se podem realizar programas interativos, já que controlamos os movimentos do usuário e respondemos a eles. Existem muitos tipos de eventos distintos, por exemplo, o click do mouse, a seleção de texto da página, entre outros.

A linguagem Javascript tem uma sintaxe muito parecida a de Java por estar baseado nele. Também é muito parecida a da linguagem C, de modo que se o leitor conhece alguma destas duas linguagens poderá manejar com facilidade com o código.

Um comentário é uma parte de código que não é interpretada pelo navegador e cuja utilidade radica em facilitar a leitura ao programador. O programador, a medida que desenvolve o script, vai deixando frases ou palavras soltas, chamadas comentários, que ajudam a ele ou a qualquer outro a ler mais facilmente o script na hora de modificá-lo ou depurá-lo.

Existem dois tipos de comentários na linguagem. Um deles, a barra dupla, serve para comentar uma linha de código. O outro comentário pode ser utilizado para comentar várias linhas, é indicado com os signos `/*` para começar o comentário e `*/` para terminá-lo. Vejamos:

JavaScript

```
<script>
  //Comentário de uma linha

  /*
  Comentário de várias Linhas
  .
  .
  */
</script>
```

Em javascript se devem respeitar as maiúsculas e as minúsculas. Se nos equivocamos ao utilizá-las o navegador responderá com uma mensagem de erro de sintaxe. Por convenção os nomes das coisas se escrevem em minúsculas, salvo que se utilize um nome com mais de uma palavra, pois nesse caso se escreverão com maiúsculas as iniciais das palavras seguintes à primeira. Também se pode utilizar maiúsculas nas iniciais das primeiras palavras em alguns casos, como os nomes das classes, apesar de que já veremos mais adiante quais são estes casos e o que são as classes.

As distintas instruções que contém nossos scripts devem ser separadas convenientemente para que o navegador não indique os correspondentes erros de sintaxe. Javascript tem duas maneiras de separar instruções. A primeira é através do caractere ponto e vírgula (;) e a segunda é através de uma quebra de linha. Por esta razão, as sentenças Javascript não necessitam acabar em ponto e vírgula a não ser que coloquemos duas instruções na mesma linha.

De qualquer forma, não é uma idéia ruim se acostumar a utilizar o ponto e vírgula depois de cada instrução, pois outras linguagens como Java ou C obrigam a utilizá-las e estaremos nos acostumando a realizar uma sintaxe mais parecida à habitual em torno de programações avançadas.

- ✓ Semelhante a Java ou linguagem C.
- ✓ Case Sensitive.
- ✓ Separação das instruções com ";".

DECLARANDO VARIÁVEIS

Uma variável é um espaço em memória onde se armazena um dado, um espaço onde podemos salvar qualquer tipo de informação que necessitemos para realizar as ações de nossos programas. Por exemplo, se nosso programa realiza somas, será muito normal guardarmos em variáveis as distintas parcelas que participam na operação e o resultado da soma. O efeito seria algo parecido a isto.

JavaScript

```
parcela1 = 23;
parcela2 = 33;

soma = parcela1 + parcela2;
```

Neste exemplo temos três variáveis, parcela1, parcela2 e soma, onde guardamos o resultado. Vemos que seu uso para nós é como se tivéssemos um apartado onde salvar um dado e que se pode acessa-los colocando somente seu nome.

Os nomes das variáveis devem ser construídos com caracteres alfanuméricos e o caractere sublinhado (_). A parte disso há uma série de regras adicionais para construir nomes para variáveis. A mais importante é que tem de começar por um caractere alfabético ou sublinhado. Não podemos utilizar caracteres raros como o signo +, um espaço ou um \$. Nomes admitidos para as variáveis poderiam ser:

JavaScript

```
Idade;
PaisDeNascimento;
_nome;
```

Também há que evitar utilizar nomes reservados como variáveis, por exemplo, não poderemos chamar a nossas variáveis palavras como return ou for que já veremos que são utilizadas para estruturas da própria linguagem. Vejamos agora alguns nomes de variáveis que não está permitido utilizarem

JavaScript

```
12meses ;
seu nome ;
return ;
pe%pe ;
```

Declarar variáveis consiste em definir e de passo informar ao sistema de que se vai utilizar uma variável. É um costume habitual nas linguagens de programação definir as variáveis que serão

utilizadas nos programas e para isso, seguem-se algumas regras restritas. Porém, javascript ultrapassa muitas regras por ser uma linguagem mais livre na hora de programar e um dos casos no qual outorga um pouco de liberdade é na hora de declarar as variáveis, já que não estamos obrigados a fazê-lo, ao contrário do que acontece na maioria das linguagens de programação.

De qualquer forma, é aconselhável declarar as variáveis, além de um bom costume e para isso Javascript conta com a palavra var. Como é lógico, utiliza-se essa palavra para definir a variável antes de utilizá-la.

JavaScript

```
var operando1;  
var operando2;
```

Também se pode atribuir um valor à variável quando se está declarando

JavaScript

```
var operando1 = 23;  
var operando2 = 33;
```

Também se permite declarar várias variáveis na mesma linha, sempre que se separem por vírgulas.

JavaScript

```
var operando1, operando2;
```

Também poderemos declarar variáveis em lugares mais dimensionados, como por exemplo, uma função. A estas variáveis chamaremos de locais. Quando se declarem variáveis locais somente poderemos acessá-las dentro do lugar aonde tenham sido declaradas, ou seja, se havíamos declarado em uma função somente poderemos acessá-la quando estivermos nessa função.

As variáveis podem ser locais em uma função, mas também podem ser locais a outros âmbitos, como por exemplo, um loop. Em geral, são âmbitos locais qualquer lugar dimensionado por chaves.

JavaScript

```
<script>  
  function minhaFuncao() {  
    var variavelLocal;  
  }  
</script>
```

No script anterior declaramos uma variável dentro de uma função, pelo qual esta variável somente terá validade dentro da função. Podem-se ver as chaves para dimensionar o lugar onde está definida essa função ou seu âmbito.

Não há problema em declarar uma variável local com o mesmo nome que uma global, neste caso a variável será visível desde toda a página, exceto no âmbito onde está declarada a variável local já que neste lugar esse nome de variável está ocupado pela local e é ela quem tem validade. Resumindo, em qualquer lugar da página, a variável que terá validade é a global. Menos no âmbito onde está declarada a variável local, que será ela que vai ter validade.

JavaScript

```
<script>
  var numero = 2
  function minhaFuncao() {
    var numero = 19
    document.write(numero) //imprime 19
  }
  document.write(numero) //imprime 2
</script>
```

Um conselho para os principiantes poderia ser não declarar variáveis com os mesmos nomes, para que nunca tenha confusão sobre qual variável é a que tem validade em cada momento.

Em Javascript temos liberdade para declarar ou não as variáveis com a palavra var, mas os efeitos que conseguiremos em cada caso serão distintos. Na verdade, quando utilizamos var, estamos fazendo com que a variável que estamos declarando seja local ao âmbito onde se declara. Por outro lado, se não utilizamos a palavra var para declarar uma variável esta será global a toda a página, seja qual for o âmbito no qual tenha sido declarada.

No caso de uma variável declarada na página web, fora de uma função ou de qualquer outro âmbito mais reduzido, é indiferente se se declara ou não com var, desde um ponto de vista funcional. Isto é devido a que qualquer variável declarada fora do âmbito global a toda a página. A diferença pode ser apreciada em uma função, por exemplo, já que se utilizamos var a variável será local à função e se não o utilizamos, a variável será global à página. Esta diferença é fundamental na hora de controlar corretamente o uso das variáveis na página, já que se não o fazemos em uma função poderíamos sobrescrever o valor de uma variável, perdendo o dado que pudesse conter previamente.

JavaScript

```
<script>
  var numero = 2
  function minhaFuncao() {
    numero = 19
    document.write(numero) //imprime 19
  }
  document.write(numero) //imprime 2
  //chamamos a funcao
  minhaFuncao()
  document.write(numero) //imprime 19
</script>
```

Neste exemplo, temos uma variável global à página chamada numero, que contém um 2. Também temos uma função que utiliza a variável numero sem a ter declarado com var, pelo que a variável numero da funcao será mesma variável global numero declarada fora da função. Em uma situação com esta, ao executar a função se sobrescreverá a variável numero e o dado que havia antes de executar a função se perderá.

TIPOS DE DADOS

Em uma variável podemos introduzir vários tipos de informação, por exemplo, texto, números inteiros ou reais, etc. A estas distintas classes de informação conhecemos como tipos de dados. Cada um tem características e usos distintos, vejamos quais são os tipos de dados de Javascript.

Números - Para começar temos o tipo numérico, para salvar números como 9 ou 23.6

Strings - O tipo String salva um texto. Sempre que escrevemos uma cadeia de caracteres devemos utilizar as aspas (").

Booleanos - Também contamos com o tipo booleano, que salva uma informação que pode valer como sim (true) ou não (false).

Além disso, usamos também os operadores exatamente como os de Lógica de Programação.

| DESCRIÇÃO | SÍMBOLO |
|-------------------|---------|
| Negação unária | - |
| Incremento | ++ |
| Redução | -- |
| Multiplicação | * |
| Divisão | / |
| Módulo aritmético | % |
| Adição | + |
| Subtração | - |

Operadores lógicos

| DESCRIÇÃO | SÍMBOLO |
|------------------------|---------|
| NÃO lógico | ! |
| Menor que | < |
| Maior que | > |
| Menor ou igual a | <= |
| Maior ou igual a | >= |
| Igualdade | == |
| Desigualdade | != |
| Lógico AND | && |
| OU lógico | |
| Condicional (ternário) | ?: |
| Vírgula | , |
| Igualdade estrita | === |
| Desigualdade estrita | !== |

Operadores bit a bit

| DESCRIÇÃO | SÍMBOLO |
|-----------|---------|
|-----------|---------|

| | |
|--|-----|
| NOT bit a bit | ~ |
| Deslocamento para a esquerda bit a bit | << |
| Deslocamento para a direita bit a bit | >> |
| Deslocamento para a direita sem sinal | >>> |
| AND bit a bit | & |
| XOR bit a bit | ^ |
| OR bit a bit | |

Operadores de Atribuição

| DESCRIÇÃO | SÍMBOLO |
|---------------------|--------------------|
| Atribuição | = |
| Atribuição composta | OP= (como += e &=) |

A diferença entre o == (igualdade) e === (igualdade estrita) é que o operador de igualdade forçará valores de diferentes tipos antes de verificar a igualdade. Por exemplo, comparar a cadeia de caracteres "1" com o número 1 fará uma comparação como true. O operador de igualdade estrita, por outro lado, não forçará valores para diferentes tipos e, portanto, a cadeia de caracteres "1" não será comparada como igual ao número 1.

A tabela a seguir lista os operadores JavaScript, ordenados da maior até a menor precedência. Operadores com a mesma precedência são avaliados da esquerda para a direita.

| OPERADOR | DESCRIÇÃO |
|-----------------|--|
| . [] () | Acesso a campos, indexação de matriz, chamadas de função e agrupamento de expressões |
| ++ -- - ~ ! new | Operadores unários, tipo de dados de retorno, criação de objetos indefinidos |
| * / % | Multiplicação, divisão, divisão de módulo |
| + - + | Adição, subtração, concatenação de cadeias de caracteres |
| << >> >>> | Deslocamento de bits |
| < <= > >= | Menor que, menor que ou igual a, maior que, maior que ou igual a |
| == != === !== | Igualdade, desigualdade, igualdade estrita e desigualdade estrita |
| & | AND bit a bit |
| ^ | XOR bit a bit |

| | |
|-------|--|
| | OR bit a bit |
| && | Lógico AND |
| | OU lógico |
| ?: | Condicional |
| = OP= | Atribuição, atribuição com operação (como += e &=) |

PARA SABER MAIS

- [Aptana Studio 3](#)
- [Sublime Text is a sophisticated text editor for code](#)
- [The NetBeans IDE provides enhanced JavaScript editing features](#)
- [5 razões pelas quais JavaScript pode ser a próxima grande linguagem](#)

EXERCÍCIOS

<!--Construa as quatro operações matemáticas básicas em JavaScript.-->

11- ESTRUTURAS DE SELEÇÃO

Normalmente, instruções em um script JavaScript são executadas uma após a outra, na ordem em que são escritas. Isso é chamado de execução sequencial e é a direção padrão do fluxo do programa. Estruturas de seleção transferem o fluxo do programa para outra parte do script. Ou seja, em vez de executar a próxima instrução na sequência, outra instrução é executada.

Para tornar um script útil, essa transferência de controle deve ser feita de maneira lógica. A transferência do controle do programa se baseia em uma decisão, que é o resultado de qual instrução é verdadeira (retornar um booleano true ou false). Você cria uma expressão e, depois, testa se o seu resultado é verdadeiro. Há dois tipos principais de estruturas de programa que fazem isso.

O primeiro é a estrutura de seleção. Você usa para especificar rotas alternativas do fluxo do programa, criando uma junção no seu programa (como uma bifurcação em uma estrada). Há quatro estruturas de seleção disponíveis em JavaScript.

- A estrutura de seleção única (if).
- A estrutura de seleção dupla (if/else).
- O operador ternário embutido (?:)
- Estrutura de seleção múltipla (switch).

JavaScript oferece suporte para instruções condicionais if e if...else. Em instruções if, uma condição é testada e, se ela corresponder ao teste, o código JavaScript relevante será executado. Na instrução if...else, o código diferente será executado se a condição for reprovada no teste. A forma mais simples de uma instrução if pode ser escrita em uma linha, mas instruções if e if...else de várias linhas são muito mais comuns.

Os exemplos a seguir demonstram sintaxes que você pode usar com instruções if e if...else. O primeiro exemplo mostra o tipo mais simples de teste booleano. Se (e somente se) o item entre parênteses for avaliado como (ou puder ser forçado como) verdadeiro, a instrução ou o bloco de instruções depois de se será executado.

JavaScript

```
var numero1 = 23;
var numero2 = 63;
if (numero1 == numero2) {
    alert("Os dois números são iguais")
}
else {
    if (numero1 > numero2) {
        alert("O primeiro número é maior que o segundo")
    }
    else {
        alert("O primeiro número é menor que o segundo")
    }
}
```

PARA SABER MAIS

- [JavaScript - Switch Case](#)
- [JavaScript If...Else Statements](#)

EXERCÍCIOS

<!--Construa um algoritmo em JavaScript que pergunta se um individuo é masculino ou feminino e mostra o resultado.

-->

12- FUNÇÕES

Funções JavaScript realizam ações e também podem retornar valores. Às vezes, estes são os resultados de cálculos ou comparações. Funções também são chamadas "métodos globais".

Funções combinam várias operações em um único nome. Isso permite simplificar o seu código. Você pode escrever um conjunto de instruções, nomeá-las e executar o conjunto inteiro chamando-o e transmitindo a ele todas as informações necessárias.

Você transmite informações a uma função colocando-as entre parênteses após o nome da função. Os fragmentos de informações que são transmitidos para uma função se chamam argumentos ou parâmetros. Algumas funções não usam argumentos, enquanto outras usam um ou mais argumentos. Em algumas funções, o número de argumentos depende de como você está usando a função.

A linguagem JavaScript oferece suporte para dois tipos de funções: funções internas da linguagem e funções que você mesmo cria e pode usá-las quando necessário. Uma definição de função consiste em uma instrução de função e um bloco de instruções JavaScript.

ESCREVENDO FUNÇÕES

Uma função deve-se definir com uma sintaxe especial que vamos conhecer a seguir:

JavaScript

```
function nomeFuncao ( ) {  
    //instruções da função  
    .  
    .  
    .  
}
```

Primeiro escreve-se a palavra função, reservada para este uso. Seguidamente se escreve o nome da função, que como os nomes de variáveis podem ter números, letras e algum caractere adicional como um hífen abaixo. A seguir se colocam entre chaves as diferentes instruções da função. As chaves no caso das funções não são opcionais, ademais é útil colocá-las sempre como se vê no exemplo, para que seja visto facilmente a estrutura de instruções que engloba a função.

Vejamos um exemplo de função para escrever na página uma mensagem de boas vindas dentro de etiquetas <h1> para que fique mais ressaltado.

JavaScript

```
function escreverBoasvindas( ) {  
    document.write("<h1> -- OLA TODOS -- </h1>");  
}
```

Simplesmente escreve na página um texto, é uma função tão simples que o exemplo não expressa suficientemente o conceito de função, mais já veremos outras mais complexas. As etiquetas H1 não se escrevem na página, e sim são interpretadas como o significado da mesma, neste caso que escrevemos um cabeçalho de nível um. Como estamos escrevendo em uma página web, ao colocar etiquetas HTML se interpretam como o que são.

Quando se chamam às funções: Para executar uma função temos que chamá-la em qualquer parte da página, com isso conseguiremos que se executem todas as instruções que tem a função entre as duas chaves. Para executar a função utilizamos seu nome seguido dos parênteses.

JavaScript

```
NomeDaFuncao( )
```

Dentro das funções podemos declarar variáveis, inclusive os parâmetros são como variáveis que se declaram no cabeçalho da função e que se iniciam ao chamar a função. Todas as variáveis declaradas em uma função são locais a essa função, ou seja, somente terão validade durante a execução da função.

Podemos declarar variáveis em funções que tenham o mesmo nome que uma variável global à página. Então, dentro da função a variável que terá validade é a variável local e fora da função terá validade a variável global à página.

Em troca, se não declaramos as variáveis nas funções se entenderá por javascript que estamos fazendo referência a uma variável global à página, de modo que se não está criada, a variável a cria, mas sempre global à página no lugar de local à função.

PARÂMETROS

As estruturas que vimos anteriormente sobre funções não são as únicas que devemos aprender para manejá-las em toda a sua potência. As funções também têm uma entrada e uma saída, que se podem utilizar para receber e devolver dados.

Os parâmetros se usam para mandar valores à função, com os quais ela trabalhará para realizar as ações. São os valores de entrada que recebem uma função. Por exemplo, uma função que realizasse uma soma de dois números teria como parâmetros a esses dois números. Os dois números são a entrada, assim como a saída seria o resultado, mais isso será visto mais tarde.

Vejamos um exemplo anterior no qual criávamos uma função para mostrar uma mensagem de boas vindas à página web, mas que agora passaremos um parâmetro que vai conter o nome da pessoa a qual se vai saudar.

JavaScript

```
function escreverBoasvindas(nome) {  
    document.write("<h1> -- OLA " + nome + " -- </h1>");  
}
```

Como podemos ver no exemplo, para definir na função um parâmetro temos que por o nome da variável que vai armazenar o dado que passarmos. Essa variável, que neste caso se chama nome, terá como valor o dado que passarmos a esta função quando a chamarmos, além disso, a variável terá vida durante a execução da função e deixará de existir quando a função terminar sua execução.

Para chamar a uma função que tem parâmetros coloca-se entre parêntesis o valor do parâmetro. Para chamar à função do exemplo haveria que escrever:

JavaScript

```
escreverBoasvindas("Alberto Garcia");
```


Ao chamar a função assim, o parâmetro nome toma como valor "Alberto Garcia" e ao escrever a saudação na tela escreverá "Olá Alberto Garcia" entre etiquetas <H1>.

Os parâmetros podem escrever qualquer tipo de dados, numérico, textual, booleano ou um objeto. Realmente não especificamos o tipo do parâmetro, por isso devemos ter um cuidado especial ao definir as ações que realizamos dentro da função e ao passar valores à função para assegurarmos que tudo é consequente com os tipos de nossas variáveis ou parâmetros.

Uma função pode receber tantos parâmetros quanto quisermos e para expressá-lo colocam-se os parâmetros separados por vírgulas dentro dos parênteses. Vejamos rapidamente a sintaxe para que a função de antes receba dois parâmetros, primeiro, o nome a quem saudar e segundo, a cor do texto.

JavaScript

```
function escreverBoasvindas(nome, corTexto) {  
    document.write("<h1 style='color:" + corTexto + "'>-- OLA " + nome + " -- </h1>");  
}
```

Chamaríamos a função com esta sintaxe. Entre parênteses colocaremos os valores dos parâmetros.

JavaScript

```
var meuNome = "Pedro"  
var minhaCor = "red"  
escreverBoasvindas(meuNome, minhaCor)
```

Entre parênteses, duas variáveis no lugar de dois textos entre aspas. Quando colocamos variáveis entre os parâmetros na verdade o que estamos passando à função são os valores que contêm as variáveis e não as mesmas variáveis.

Para seguir a linha do uso de parâmetros em nossos programas Javascript, temos que indicar que os parâmetros das funções se passam por valor. Isto quer dizer que mesmo que modifiquemos um parâmetro em uma função a variável original que havíamos passado não mudará seu valor. Pode-se ver facilmente com um exemplo.

JavaScript

```
function passoPorValor(meuParametro) {  
    meuParametro = 32;  
    document.write("mudei o valor a 32");  
}  
var minhaVariavel = 5  
passoPorValor(minhaVariavel);  
document.write("o valor da variavel e: " + minhaVariavel);
```

No exemplo, temos uma função que recebe um parâmetro, e que modifica o valor do parâmetro atribuindo-lhe o valor 32. Também temos uma variável, que iniciamos a 5 e posteriormente chamamos a função passando esta variável como parâmetro. Como dentro da função modificamos o valor do parâmetro poderia acontecer da variável original mudasse de valor, mas como os parâmetros não modificam o valor original das variáveis, esta não muda de valor. Deste modo, ao imprimir na tela o valor de minhaVariavel se imprimirá o número 5, que é o valor original da variável, no lugar de 32 que era o valor col o que havíamos atualizado o parâmetro.

RETORNANDO VALORES

As funções também podem retornar valores, de modo que ao executar a função poderá se realizar ações e obter um valor de retorno. Por exemplo, uma função que calcula o quadrado de um número terá como entrada -tal como vimos- a esse número e como saída terá o valor resultante de encontrar o quadrado desse número. Uma função que devolva o dia da semana teria como saída em dia da semana.

Vejamos um exemplo de função que calcula a média de dois números. A função receberá os dois números e retornará o valor da média.

JavaScript

```
function media(valor1, valor2) {  
  var resultado  
  resultado = (valor1 + valor2) / 2  
  return resultado  
}
```

Para especificar o valor que retornará a função se utiliza a palavra return seguida do valor que se deseja devolver. Neste caso se devolve o conteúdo da variável resultado, que contém a média dos dois números. Para receber os valores que devolve uma função se coloca o operador de atribuição =. Para ilustrar isto, pode-se ver este exemplo, que chamará à função média() e salvará o resultado da média em uma variável para logo, imprimi-la na página.

JavaScript

```
var minhaMedia;  
minhaMedia = media(12, 8);  
document.write(minhaMedia);
```

Em uma mesma função podemos colocar mais de um return. Logicamente só vamos poder retornar uma coisa, mas dependendo do que tenha acontecido na função poderá ser de um tipo ou de outro, com uns dados ou outros. Nesta função podemos ver um exemplo de utilização de múltiplos return. Trata-se de uma função que devolve um 0 se o parâmetro recebido era par e o valor do parâmetro se este era ímpar.

JavaScript

```
function multiploReturn(numero) {  
  var resto = numero % 2  
  if (resto == 0)  
    return 0  
  else  
    return numero  
}
```

Para averiguar se um número é par encontramos o resto da divisão ao dividi-lo entre 2. Se o resto é zero é porque era par e devolvemos um 0, em caso contrário -o número é ímpar- devolvemos o parâmetro recebido.

PARA SABER MAIS

- [A function is a JavaScript procedure](#)

13 - EVENTOS

O desenvolvimento dos mais distintos recursos que você utiliza hoje ao navegar na Internet, acessando seu webmail, comprando produtos, participando de promoções, e tudo mais só se tornou viável de implementação com o uso do JavaScript. Embora todos estes serviços sejam possíveis de se implementar diretamente com HTML/CSS e uma linguagem server-side, o JavaScript facilita, otimiza e aprimora a maioria dos quesitos funcionais de interação com o usuário. As ações do usuário no navegador só podem ser controladas através dos scripts, com eventos.

Um evento consiste em uma ação, que pode ser disparado pelo próprio fluxo de navegação, como finalizar o download do arquivo corrente, ou então por alguma atividade do usuário, como passar o mouse sobre um elemento, ou pressionar uma tecla.

A programação orientada a eventos possibilita a criação e a funcionalidade de toda a Web como a conhecemos hoje. São menus, validação de dados, animações de elementos, facilidades de navegação e uma série de outros quesitos.

Como você deve contextualizar a programação orientada a eventos? Pode ser mais simples do que você imagina. Imagine o objeto de um aparelho de DVD, seria como se você pudesse programar uma rotina da seguinte natureza ao aparelho: "ao terminar o filme, ejetar a bandeja do disco". E no caso da programação do código, como é possível vislumbrar algo dessa natureza? Algo como "ao carregar o documento inteiro, imprimir algo a mais na página" pode ser uma primeira amostragem. Confira no exemplo:

JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>Eventos </title>
  <script type="text/javascript">
    function escrever() {
      var teste = 1;
      var TESTE = 2;

      document.write(teste);
      document.write(TESTE);
    }
  </script>
</head>
<body onload="escrever();">
</body>
</html>
```

Analisando o código: há uma declaração de uma função com o nome escrever. Ela possui duas variáveis distintas declaradas: teste e TESTE, com valores distintos. O que faz a função? Ela escreve no documento os valores das variáveis declaradas, assim que o documento for descarregado no navegador do cliente. Isto pode ser notado pelo evento onload da tag

<body> </body>. Existem muitos eventos que se pode aplicar, inclusive mais de um para o mesmo elemento. Isso significa que é possível você definir rotinas distintas para eventos distintos, que operem sobre um objeto comum.

Analogamente o aparelho de DVD: se o botão de reprodução for pressionado uma vez, inicia-se a atividade; mas caso o mesmo botão seja pressionado e mantido pressionado, o filme é avançado em uma velocidade ligeiramente mais rápida.

Aplicando isto no nosso modelo web: "caso o usuário aponte o cursor para o botão da soma, o cálculo é processado; o resultado só é exibido caso o botão seja pressionado". Veja no código abaixo, uma proposta para resolução:

JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>Eventos </title>
  <script defer type="text/javascript">
    var resultado = 0;
    function calcular() {
      var v1 = 10;
      var v2 = 20;
      resultado = v1 + v2;
    }

    function imprimir() {
      document.write(resultado);
    }
  </script>
</head>
<body>
  <button onmouseover="calcular();" onclick="imprimir();" >Teste! </button>
</body>
</html>
```

EVENTOS DE JANELA

- onbeforeprint() - Disparado antes da impressão da página. Você pode usá-lo para modificar, esconder ou exibir elementos, preparando a página para impressão.
- onafterprint() - Disparado após a impressão da página. Você pode usá-lo para reverter o status anterior à impressão.
- onhashchange() - A última porção da URL, após o hash (#), foi modificada.
- onoffline() - O agente de usuário ficou offline.
- ononline() - O agente de usuário está novamente conectado.
- onredo() - O usuário disparou a ação de "Refazer".
- onundo() - O usuário disparou a ação de "Desfazer".

São aplicados ao elemento <body> </body>

- `onload()` - Executa o script assim que o download do documento for finalizado para o navegador do usuário.
- `onunload()` - Executa o script quando o documento corrente for descarregado pelo usuário como sair da URL ou fechar o navegador.

HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Eventos </title>
  <script defer type="text/javascript">
    function Carregado(event) {
      alert("Página carregada");
    }
    function Sair() {
      return "Deseja Sair?";
    }
  </script>
</head>
<body onload="Carregado();" onbeforeunload="return Sair();">
  <a href="http://www.google.com.br/">Teste!</a>
</body>
</html>
```

EVENTOS DE FORMULÁRIO

Aplica-se aos elementos presentes em tags `<form>` `</form>`, como `<input>` `<input>` e `<select>` `</select>`.

- `onchange()` - Dispara o script assim que o elemento sofrer alguma alteração.
- `onsubmit()` - Realiza a rotina do script quando o formulário é enviado para processamento, através do seu envio.
- `onreset()` - Executa o script quando o formulário é reinicializado.
- `onselect()` - Processa o script quando o elemento é selecionado.
- `onblur()` - Quando o elemento perde o foco do usuário, o script é processado.
- `onfocus()` - Assim que o elemento é focado, o script ganha sua execução.
- `oninput()` - O usuário entrou com dados no campo
- `oninvalid()` - O campo não passou pela validação

HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Eventos </title>
  <script defer type="text/javascript">
    function Verifica(obj) {
      alert("O valor foi alterado para " + obj.value)
    }
  </script>
</head>
<body>
  <form id="teste">
    Teste:<input type="text" id="campo" name="campo" onchange="Verifica(this);" />
  </form>
</body>
</html>
```

PARA SABER MAIS

- [JavaScript HTML DOM Events](#)
- [JavaScript Events](#)

14- MANIPULANDO FORMULÁRIOS

Os objetos de maior controle via JavaScript são os que compõe um formulário de entrada ou consulta de dados em um documento web. Para todos os campos de formulários da web, há uma série de propriedades e métodos que são comuns.

| PROPRIEDADE/MÉTODO | EXEMPLO |
|--|---|
| value = <i>String</i> , contém a informação do campo. | <code>var campo = document.getElementById('campo');</code> <code>var valor = campo.value;</code> |
| disabled = <i>Boolean</i> , habilita/desabilita o campo. | <code>var campo = document.getElementById('campo');</code> <code>campo.disabled = true;</code> |
| focus = <i>Método</i> , seta o foco do cursor para o campo. | <code>var campo = document.getElementById('campo');</code> <code>campo.focus();</code> |

A seguir exemplo de uma rotina para obter o valor de um campo texto.

JavaScript

```
var campo = document.getElementById('campo');
window.alert('O valor do campo é: ' + campo.value);
```

Para bloquear uma lista de seleção.

JavaScript

```
var combo = document.getElementById('combo');
combo.disabled = true;
```

Posicionar o cursor com o foco em um campo de senha.

JavaScript

```
var senha = document.getElementById('senha');
senha.focus();
```

INPUT TYPE SUBMIT, RESET E BUTTON

Consistem nos botões que efetivamente realizam alguma atividade frente a alguma ação do usuário. São os botões que enviam os dados para processamento (submit), limpam os campos (reset) ou permitem total personalização pelo programador (button).

INPUT TYPE FILE, HIDDEN E TEXTAREA

Para os campos de entrada de informações pelo usuário há um método que possibilita selecionar o seu conteúdo, o select.

| PROPRIEDADE/MÉTODO | EXEMPLO |
|---|--|
| select = <i>Método</i> , seleciona o campo e seu conteúdo. | <code>var campo = document.getElementById('campo');</code> <code>campo.select();</code> |
| readonly = <i>Boolean</i> , define se o campo é somente para leitura. Oficialmente, exclusivo de TEXTAREA. | <code>var campo = document.getElementById('campo');</code> <code>campo.readonly = false;</code> |

Exemplos de rotina para selecionar os dados de um campo de texto

JavaScript

```
var campo = document.getElementById('campo');
campo.select();
```

INPUT TYPE RADIO E CHECKBOX

Os botões de rádio (input type radio) são utilizados para escolher uma dentre algumas opções disponíveis, enquanto que as caixas de diálogo (input type checkbox) podem ser marcadas ou desmarcadas.

| PROPRIEDADE/MÉTODO | EXEMPLO |
|--|--|
| checked = <i>Boolean</i> , define se um elemento <i>checkbox</i> está marcado ou não. | <code>var campo = document.getElementById('campo');</code> <code>campo.checked = true;</code> |

Exemplos de uma rotina para marcar uma caixa de seleção com JS

JavaScript

```
var campo = document.getElementById('campo');
campo.checked = true;
```

SELECT

As "combos" são os elementos apresentados nos formulários que apresentam uma listagem de opções, onde o usuário deve selecionar um ou mais itens, dependendo da configuração do mesmo.

| PROPRIEDADE/MÉTODO | EXEMPLO |
|---|---|
| options = <i>Array</i> , estrutura que contém todos os elementos presentes em um elemento. | <code>var campo = document.getElementById('campo');</code> <code>var dimensao = campo.options.length;</code> |
| selectedIndex = <i>Objeto</i> , contém o objeto que está selecionado na listagem. | <code>var campo = document.getElementById('campo');</code> <code>var item = campo.options[campo.selectedIndex];</code> <code>var valor = item.value;</code> |

| | |
|---|--|
| option value = <i>String</i> , armazena o valor do atributo <i>value</i> da tag OPTION dentro da listagem. | <pre>var campo = document.getElementById('campo'); var v = campo.options[campo.selectedIndex].value;</pre> |
| option text = <i>String</i> , armazena o texto que está delimitado entre as tags OPTION. | <pre>var campo = document.getElementById('campo'); var t = campo.options[campo.selectedIndex].text;</pre> |

Exemplos de rotina para contar o número de itens presentes em uma listagem.

JavaScript

```
var combo = document.getElementById('combo');
window.alert('A lista apresenta ' + combo.options.length + ' itens');
```

Exibindo o texto e o valor da opção selecionada

HTML

```
var combo = document.getElementById('combo');
var valor = combo.options[combo.selectedIndex].value;
var texto = combo.options[combo.selectedIndex].text;

window.alert('Valor: ' + valor + '\nTexto: ' + texto);
```

VALIDAÇÃO DE FORMULÁRIOS

A sistemática da validação de formulários em páginas da web visa contemplar requisitos essenciais para a operacionalidade da aplicação. Com os detalhes de elementos de formulário expostos no tópico recente, você já pode por em prática a bagagem adquirida até este momento para aplicar suas primeiras validações de dados.

Para campos de entrada de dados pelo usuário, aqueles que forem essenciais para a regra de negócio da aplicação devem ser validados nestes quesitos. A validação tem por objetivo persistir a informação não nula e não vazia (que são diferentes).

JavaScript

```
function validarLogin() {
    var campoNome = document.getElementById('nome');
    var campoSenha = document.getElementById('senha');
    if ((campoNome.value == null) || (campoNome.value == '')) {
        window.alert('O seu nome de usuário não foi informado!');
        return (false);
    }
    else if ((campoSenha.value == null) || (campoSenha.value == '')) {
        window.alert('Sua senha não foi informada!');
        return (false);
    }
}
```

A rotina acima poderia ser invocada em momentos distintos, mas em especial, na própria submissão dos dados do formulário ela encontra seu espaço adequado. Ao utilizar o comando

return na invocação da rotina, o seu valor de retorno (booleano em todos os casos) para a função a ser executada impedirá o formulário de ser processado, "congelando" o usuário na página corrente. Caso o return não seja utilizado, o formulário será submetido independente do valor retornado na função.

HTML

```
<form id="login" action="cadastro.php" onsubmit="return validarLogin();">
  <label for="nome">CPF</label>
  <input type="text" name="nome" id="nome" size="11" maxlength="11" />
  <label for="senha">Senha</label>
  <input type="password" name="senha" id="senha" size="8" /> <br />
  <input type="submit" name="ok" value="OK" />
</form>
```

Um dos quesitos que os desenvolvedores web geralmente necessitam implementar em algum tipo de aplicação é a verificação do tamanho do conteúdo presente em um campo do tipo textarea. Isto porque geralmente a informação ali posta pelo usuário do sistema irá integrar-se a uma base de dados, em uma tabela, respeitando o tamanho de sua coluna vinculada no relacionamento.

HTML

```
function iniciarContagem() {
  document.getElementById('status').value = 255;
}

function contar() {
  var objTexto = document.getElementById('msg');
  var gasto = objTexto.value.length;
  var limite = 255;
  var restante = limite - (1 + gasto);

  if (restante < 0) {
    window.alert('Limite de conteúdo!');
    return false;
  }
  else {
    document.getElementById('status').value = restante;
    return true;
  }
}
```

CUSTOM VALIDATORS

Validações padrão, embora atendam a maioria dos casos, não são suficientes para todas as situações. Muitas vezes você vai querer escrever sua própria função de validação Javascript. Há alguns detalhes na especificação do HTML5 que vão ajudá-lo com isso:

O novo evento oninput é disparado quando algo é modificado no valor de um campo de formulário. Diferente de onchange, que é disparado ao final da edição, oninput é disparado ao editar. É diferente também de onkeyup e onkeypress, porque vai capturar qualquer modificação no valor do campo, feita com mouse, teclado ou outra interface qualquer.

O método `setCustomValidity` pode ser invocado por você. Ele recebe uma string. Se a string for vazia, o campo será marcado como válido. Caso contrário, será marcado como inválido.

Com isso, você pode inserir suas validações no campo de formulário e deixar o navegador fazer o resto. Não é mais preciso capturar o evento submit e tratá-lo. Veja, por exemplo, este formulário com validação de CPF.

HTML

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8" />
  <title>Custom validator</title>
  <!-- O arquivo cpf.js contém a função valida CPF, que recebe uma string e retorna true ou false. -->
  <script src="cpf.js"></script>
  <script>
    function vCPF(i) {
      i.setCustomValidity(validaCPF(i.value) ? "" : 'CPF inválido!')
    }
  </script>
</head>
<body>
  <form>
    <label>CPF: <input name="cpf" oninput="vCPF(this)" /></label>
    <input type="submit" value="Enviar" />
  </form>
</body>
</html>
```

PARA SABER MAIS

- [Qual a dose certa de JavaScript?](#)
- [JavaScript Form Validation](#)
- [jQuery Validation Plugin](#)

BÔNUS - DOM E HTML5

O DOM (Document Object Model) é uma interface independente de plataforma e linguagem que permite aos programas e scripts atualizarem dinamicamente a estrutura, conteúdo e estilo de documentos XML, HTML e demais. Isso significa que o DOM não é apenas utilizado em Javascript, de fato, outras linguagens de programação podem fazer uso do DOM em suas funções. A primeira versão do DOM (DOM Nível 1) foi lançada em 1998 pela W3C, a segunda (DOM Nível 2) em 2000, e, por fim, a terceira versão (DOM nível 3) foi lançada em 2004.

Podemos trabalhar com três partes relevantes de APIs (Application Programming Interface) do DOM: A DOM Core API, que é o núcleo do DOM e fornece tudo o que você precisa para criar, remover e alterar elementos da árvore do DOM de maneira genérica. A DOM HTML API, que é uma parte do DOM voltada especificamente para HTML e foi criada para trazer muitas implementações do BOM (Browser Object Model) que existiam em diversos navegadores antes da padronização. Ela é mais fácil para ser utilizada, pois, tem os mesmos atributos dos elementos HTML. E, por fim, a DOM XML API, que é voltada para arquivos XML.

Ainda que alguns dos navegadores não sigam tudo à risca, atualmente é bem mais simples trabalhar com Javascript (ECMAScript) do que já foi um dia.

O Modelo de Objetos do Documento é a interface entre a linguagem Javascript e os objetos HTML. DOM é o método padrão para construção de aplicações ricas com Javascript e é amplamente conhecido e utilizado. Basicamente, este modelo implementa uma hierarquia de composição, apresentando descendência de elementos. Isto é facilmente compreendido se você abstrair uma estrutura de árvore: há uma raiz, um caule (tronco), e uma diversidade de galhos (ramos) que se propagam ao longo da estrutura. Observe o seguinte trecho HTML simples:

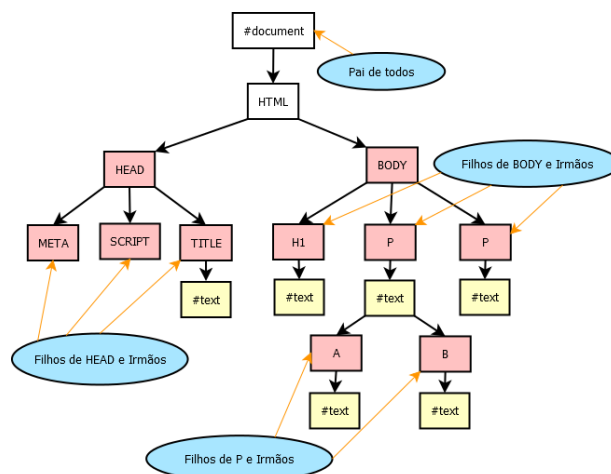
HTML

```
<html>
<head>
  <meta charset="UTF-8">
  <title>Demo</title>
  <script src="meu_arquivo_javascript.js"></script>
</head>
<body>
  <h1 id="id_h1" class="classe_h1">Sou um cabeçalho!</h1>
  <p id="id_p" class="classe_p">
    Um texto qualquer dentro de uma tag de parágrafo. Aqui também
    temos outras tags, como <a href="#"> um link</a> , ou um texto <b>em negrito</b>.
  </p>
  <p id="id_p" class="classe_p">
    Este é outro parágrafo.
  </p>
</body>
</html>
```

Agora veja como o DOM cria sua árvore a partir do HTML acima:

| nodeName | id | class |
|-----------|-------|-----------|
| #document | | |
| html | | |
| HTML | | |
| HEAD | | |
| #text | | |
| META | | |
| #text | | |
| TITLE | | |
| #text | | |
| #text | | |
| SCRIPT | | |
| #text | | |
| #text | | |
| BODY | | |
| #text | | |
| H1 | id_h1 | classe_h1 |
| #text | | |
| #text | | |
| P | id_p | classe_p |
| #text | | |
| A | | |
| #text | | |
| #text | | |
| B | | |
| #text | | |
| #text | | |
| P | id_p2 | classe_p2 |
| #text | | |
| #text | | |

Existem elementos pai (parent), filhos (childs) e irmãos (siblings). Estes elementos são caracterizados na forma como estão na árvore, veja o mesmo exemplo na imagem a seguir:



Como você pode perceber cada elemento tem seus pais e filhos. É necessário compreender a estrutura da árvore para navegar entre seus elementos utilizando Javascript. Elementos pais são tags que têm outros elementos dentro de si. Por exemplo: um `<p></p>` que tem um link `<a>` dentro de si, é pai deste link. Por outro lado, o link `<a>` que está dentro de `<p></p>` é filho deste `<p></p>`.

HTML

```
<p>Sou o pai, e <a href="#">eu o filho</a></p>
```

Se um elemento tem mais de uma tag dentro de si, esses elementos são irmãos. Por exemplo, se uma `<div></div>` tem dois elementos `<p></p>` dentro de si, ambos os elementos `<p></p>` são irmãos.

HTML

```
<div>
  <p>Eu sou filho da DIV.</p>
  <p>Eu sou filho da DIV também, e irmão do feioso aí acima.</p>
  <h1>Sou um cabeçalho, filho da DIV e irmão dos dois aí acima.</h1>
</div>
```

ACESSO HIERÁRQUICO

A primeira maneira que você dispõe de acessar os elementos que compõe um documento da web com JavaScript é usar o acesso hierárquico. Considere a pequena página XHTML a seguir, e veja como isto funciona:

HTML

```
<html>
<head>
  <title>Estrutura DOM </title>
</head>
<body>
  <form name="login" id="login" method="post" action="login.php">
    <fieldset>
      <legend>Login</legend>
      <label for="usuario">Usu&aaacute;rio</label>
      <input type="text" name="usuario" id="usuario" size="8" /> <br />
      <label for="senha">Usu&aaacute;rio</label>
      <input type="text" name="senha" id="senha" size="8" maxlength="4" />
    </fieldset>
  </form>
  <p>OK</p>
</body>
</html>
```

O principal objeto de uso é a própria janela onde o documento está sendo exibido: este é o objeto `window`. O documento contido nesta janela é o objeto `document`. A partir dele, todas as suas tags podem ser acessadas hierarquicamente. Por exemplo: para acessar o campo do usuário presente nesta tela, o código em JavaScript seria o seguinte:

JavaScript

```
var campoUsuario = window.document.forms[0].elements[0];
```

Esta página específica apresenta somente um formulário, com dois campos. Por se tratar do primeiro – e único – elemento presente na página, a indexação deve ser feita numericamente iniciando em zero. Logo, `forms[0]` significa o primeiro formulário presente no documento e

elements[0] segue o mesmo raciocínio: é o primeiro elemento do primeiro formulário presente na página. Se um novo formulário for adicionado basta mudar o índice numérico dentro dos colchetes para acessar corretamente o objeto.

ACESSO UTILIZANDO O ATRIBUTO NAME

O atributo name está disponível nas principais tags da HTML que você necessite manipular em seus scripts. A programação pode lhe parecer mais clara e objetiva, uma vez respeitado o seu padrão para atribuição de nomes.

JavaScript

```
var campoUsuario = document.login.usuario;
```

Realmente, o código parece mais claro. Mas lembre-se que se algum atributo tiver seu nome alterado, você terá que ajustar seus scripts. Isto pode ser um tanto quanto inconveniente em ambientes corporativos ou aplicações distribuídas (desenvolvidas em camadas), onde o mesmo profissional que elabora o visual da página HTML, pode não ser o mesmo que programará o código dos scripts.

ACESSO UTILIZANDO O MÉTODO getElementById()

Este recurso é bastante robusto. Além de ser suportado pelos navegadores atuais, sob qualquer plataforma, este método nativo do objeto document pode acessar qualquer elemento de uma página da web, através do atributo id. Na prática, é muito semelhante ao modelo anterior (name), com a vantagem de ser padronizado.

JavaScript

```
var campoUsuario = document.getElementById('usuario');
```

Preste atenção que a sintaxe pode ser incômoda a princípio: há realmente três letras maiúsculas que fazem parte do comando getElementById (EBI). Com a prática, você irá se habituar logo com este comando, até porque ele será um dos mais utilizados. Em um documento HTML não pode haver atributos id duplicados. Isto garante o perfeito funcionamento deste método no escopo de todo o script.

ACESSO UTILIZANDO O MÉTODO getElementsByName()

Este método tem por objetivo acessar os objetos pela presença do atributo name, assim como o método getElementById faz com o id. A novidade é que além de acessar pelo atributo, é possível percorrer uma coleção com o mesmo valor de atributo numericamente, como uma lista. Ao contrário do atributo id, o name pode ser encontrado duplicado em uma página:

JavaScript

```
var campoUsuario = document.getElementsByName('login')[0];
```

As seguintes tags aceitam a utilização do atributo name: BUTTON, TEXTAREA, APPLET, SELECT, FORM, FRAME, IFRAME, IMG, A, INPUT, OBJECT, MAP, PARAM e META. Isto tem reflexos para

diferentes browsers caso você venha a utilizar name para outros elementos que não estejam de acordo com a especificação. Neste caso, utilize-se do getElementById.

ACESSO UTILIZANDO O MÉTODO `getElementsByName()`

Seguindo o padrão apontado pelo `getElementById`, este método nativo do objeto `document` tem a finalidade de acessar objetos diretamente pela tag que o representa. Esta solução pode ser muito útil para soluções de script que necessitem percorrer coleções de tags ou tipos de tags específicas em uma página web. Para a situação em abordagem, a sintaxe para acessar o login do usuário:

JavaScript

```
var campoUsuario = document.getElementsByName('input')[0];
```

Veja que o nome tag é na verdade um parâmetro de uso do método, do tipo string. E também há uma associatividade numérica, reconhecida pelo uso dos colchetes, para obter a exata tag na ordem em que ela está presente no documento. Sempre iniciando em zero (primeira tag) até o total do número de tags presentes no documento menos um (n-1). Como este método retorna uma array com todos os elementos, caso não encontre nenhum com a tag especificada, retornará uma array de zero posições. Assim, você pode iterar esta array e trabalhar com cada elemento encontrado.

ACESSO UTILIZANDO O MÉTODO `getElementsByClassName()`

Retorna todos os elementos do HTML que possuem a classe especificada. Com HTML5 você pode fazer:

JavaScript

```
var destaques = document.getElementsByClassName('destaque');
```


E isso retornará todos os elementos do HTML que possuem a classe "destaque". Observe:

HTML

```
<!DOCTYPE html>
<html>
<body>
  <div class="destaque"> </div>
  <div class="destaque">Second div element with class="destaque".</div>
  <p>Click the button to change the text of the first div element with class="example" (index 0).</p>
  <button onclick="myFunction()"> Tente</button>
  <p><strong>Note:</strong> The getElementsByClassName() method is not supported in Internet Explorer and earlier versions.</p>

  <script type="text/javascript">
    function myFunction() {
      var x = document.getElementsByClassName("destaque");
      x[0].innerHTML = "Hello World!";
    }
  </script>
</body>
</html>
```

ACESSO UTILIZANDO O MÉTODO getSelection()

Os objetos document e window possuem um método getSelection(), que retorna um objeto Selection com a seleção atual em um documento. A seleção tem, entre outros, os seguintes métodos e propriedades:

| PROPRIEDADE/MÉTODO | DESCRIÇÃO |
|-------------------------------|--|
| anchorNode | O elemento que contém o início da seleção. |
| focusNode | O elemento que contém o final da seleção. |
| selectAllChildren(parentNode) | Seleciona todos os filhos de parentNode. |
| deleteFromDocument() | Remove a seleção do documento. |
| rangeCount | A quantidade de intervalos na seleção. |
| getRangeAt(index) | Retorna o intervalo na posição index. |
| addRange(range) | Adiciona um intervalo à seleção. |

Um objeto Selection é um conjunto de intervalos, de classe Range. Cada intervalo possui, entre outros, os seguintes métodos e propriedades:

| PROPRIEDADE/MÉTODO | DESCRIÇÃO |
|-------------------------|--|
| deleteContent() | Remove o conteúdo do intervalo. |
| setStart(parent,offset) | Seta o início do intervalo para o caractere na posição offset dentro do elemento DOM parent. |
| setEnd(parent,offset) | Seta o final do intervalo para o caractere na posição offset dentro do elemento DOM parent. |
| Propriedade/Método | Descrição |
| deleteContent() | Remove o conteúdo do intervalo. |
| setStart(parent,offset) | Seta o início do intervalo para o caractere na posição offset dentro do elemento DOM parent. |
| setEnd(parent,offset) | Seta o final do intervalo para o caractere na posição offset dentro do elemento DOM parent. |

MÉTODOS DE MANIPULAÇÃO HTML

Veja algumas das funcionalidades que estão disponíveis nativamente para programar alguns recursos em seus scripts:

appendChild : adiciona nodos na estrutura do documento;

JavaScript

```
<html>
<head>
  <title>JS DOM</title>
  <script type="text/javascript">
    function CriarParagrafo() {
      var objNovoParagrafo = document.createElement('p');
      var strTexto = document.createTextNode('Informação do parágrafo.');
```

objNovoParagrafo.appendChild(strTexto);

document.getElementById('pindice').appendChild(objNovoParagrafo);

```
    }
  </script>
</head>
<body>
  <h1>Teste</h1>
  <div id="pagina">
    <h3>Exemplo</h3>
    <p id="pindice" onclick="CriarParagrafo();">Inserir conteúdo a partir deste parágrafo</p>
  </div>
</body>
</html>
```

innerHTML : conteúdo inserido entre o nodo, no padrão HTML;

JavaScript

```
<html>
<head>
  <title>JS DOM</title>
  <script type="text/javascript">
    function Conteudo() {
      var objTag = document.getElementById('pindice');
```

window.alert(objTag.innerHTML);

```
    }
  </script>
</head>
<body>
  <h1>Teste</h1>
  <div id="pagina">
    <h3>Exemplo</h3>
    <p id="pindice" onclick="Conteudo();">O que há <i>aqui</i>.</p>
  </div>
</body>
</html>
```

insertBefore : adiciona o novo nodo após uma referência de outro nodo; Os dois parâmetros utilizados por este método tem a finalidade de identificar o novo objeto de conteúdo a ser adicionado (primeiro parâmetro), com referência a um objeto irmão de mesmo nível (segundo parâmetro).

JavaScript

```
<head>
<title>JS DOM</title>
<script type="text/javascript">
  function Inserir(objeto) {
    var objNovoParagrafo = document.createElement('p');
    var strTexto = document.createTextNode('Conteudo criado!');
    objNovoParagrafo.appendChild(strTexto);
    objeto.parentNode.insertBefore(objNovoParagrafo, objeto);

  }
</script>
</head>
<body>
  <h1>Teste</h1>
  <div id="pagina">
    <h3>Exemplo</h3>
    <p id="p1" onclick="Inserir(this);">Adicionar conteudo antes deste elemento...</p>
    <p id="p2" onclick="Inserir(this);">ou deste.</p>
  </div>
</body>
```

removeChild : exclui o nodo filho parametrizado. Caso não hajam filhos, o próprio nodo é excluído; O parâmetro identifica qual é o nodo a ser excluído a partir do nodo pai. Pode-se utilizar um valor numérico ou, como no caso exposto, passar um objeto a ser excluído.

JavaScript

```
<head>
<title>JS DOM</title>
<script type="text/javascript">
  function Inserir(objeto) {
    var objNovoParagrafo = document.createElement('p');
    var strTexto = document.createTextNode('Conteudo criado!');
    objNovoParagrafo.appendChild(strTexto);
    objeto.parentNode.insertBefore(objNovoParagrafo, objeto);
    objNovoParagrafo.onclick = function () { this.parentNode.removeChild(this); };
  }

</script>
</head>
<body>
  <h1>Teste</h1>
  <div id="pagina">
    <h3>Exemplo</h3>
    <p id="p1" onclick="Inserir(this);">Adicionar conteudo antes deste elemento...</p>
    <p id="p2" onclick="Inserir(this);">ou deste.</p>
  </div>
</body>
```

PARA SABER MAIS

- [The definitive source of the best JavaScript libraries, frameworks, and plugins](#)
- [Using the W3C DOM](#)

EXERCÍCIOS EXTRAS

1. HTML significa:

- a) Home Tool Markup Language
- b) Hyper Text Markup Language**
- c) Hyper Text Macro Language

2. Qual a forma correta de criar um checkbox?

- a) <check>
- b) <checkbox>
- c) <input type="check">
- d) <input type="checkbox"/>**

3. Para inserir uma imagem a tag correta é:

- a) **
- b)
- c)
- d) <imagem src = "img.jpg"/>

4. Quais as tags abaixo estão corretamente aninhadas:

- a) <p>Olá Mundo HTML</p>
- b) <p>Olá Mundo HTML </p>**
- c) <p>Olá Mundo HTML</p>

5. O elemento nav, permite que um conjunto de _____ seja incorporado e interpretado como um _____.

a) links, menus

b) menus, links

c) divs, menu

d) links, sessão

6. Um arquivo externo de CSS normalmente é ligado ao documento HTML como:

a) ending

b) external

c) link

d) object

7. Para criar uma lista com dropdown usamos as tags:

a) <select> e <dropdown>

b) <select> e <option>

c) <legend> e <select>

d) <select> e <legend>

8. Para criar uma função em JavaScript usamos a sintaxe:

a) funcao nome_funcao(){
 }
}

b) function nomeFuncao(){
 }
}

c) Function Nome da Função () {
 }
}

d) function nomeFuncao {

}

9. Nesta questão marque todas as alternativas que achares correta. Usamos JavaScript para:

a) turbinar as páginas de internet

b) criar interação com o usuário

c) aplicar efeitos visuais

d) melhorar o desempenho

10. Qual atributo indica a linguagem que vai tratar os dados vindos de um formulário:

a) action

b) inputy

c) imputy

d) post

11. O atributo placeholder pode ser usado juntamente com qual tag

a) div

b) canvas

c) span

d) inputy

12. Em JavaScript existem formas de ligar o script na página, como colocamos o código junto com a página é chamado de:

a) incorporada

b) inline

c) externo

d) nenhuma das alternativas acima é correta

13. Em JavaScript quando colocamos o script na tag como mostra o exemplo: `<input type="button" onClick="verificarSenhas()">` podemos afirmar que esta forma é conhecida como:

a) incorporada

b) inline

c) externo

d) nenhuma das alternativas acima é correta

14. Em JavaScript usamos duas funções básicas, uma chamada `document.write()` e outra chamada `alert()`, sobre elas diga, qual sua diferença respectivamente:

a) `document.write()` mostra a mensagem na página enquanto o `alert()` mostra a mensagem usando uma nova janela.

b) `document.write` mostra a mensagem usando uma janela, enquanto o `alert()` mostra a mensagem usando o corpo da página;

c) Não existe diferença entre as duas, ambas tem a mesma função e seu resultado é exatamente o mesmo, ou seja, mostrar a mensagem no corpo da página e não em uma janela.

d) Não existe diferença entre as duas, ambas tem a mesma função e seu resultado é exatamente o mesmo, ou seja, mostrar a mensagem em uma janela e não no corpo da página.

15. Para gerar uma caixa de diálogo com o usuário em JavaScript, usamos o comando:

a) alert

b) prompt

c) focus

d) nenhum dos acima indicado

16. As variáveis em JavaScript são declaradas de que forma:

a) var nomeVariavel tipoDados

b) var nomeVariavel

c) nomeVariavel tipoDados

d) var nomeVariavel: tipoDados

17. Os comentários em HTML, CSS e JavaScript são declarados da seguinte forma, respectivamente.

a) /*JS*/, /*CSS*/, /*HTML*/

b) /*HTML*/, /*CSS*/, /*JS*/

c) <!--HTML-->, /*CSS*/, /*JS*/

d) /*CSS*/, /*JS*/, <!--HTML-->