

HTML



# SUMÁRIO

INTRODUÇÃO .....	6
Client Side e Server Side.....	7
HTML, CSS e Javascript.....	7
HTML .....	8
Estrutura Básica.....	9
Exercícios de Fixação.....	9
Exercícios Complementares .....	11
Semântica HTML .....	11
Parágrafos .....	12
Exercícios de Fixação.....	13
Exercícios Complementares .....	14
Cabeçalhos .....	14
Exercícios de Fixação.....	16
Exercícios Complementares .....	18
Links.....	18
Exercícios de Fixação.....	20
Exercícios Complementares .....	20
Âncoras.....	21
Exercícios de Fixação.....	22
Exercícios Complementares .....	24
Imagens .....	24
Exercícios de Fixação.....	26
Exercícios Complementares .....	26
Tabelas .....	27
Exercícios de Fixação.....	30
Exercícios Complementares .....	32
Listas.....	32
Exercícios de Fixação.....	33
Exercícios Complementares .....	34
Exercícios de Fixação .....	35
Exercícios Complementares .....	36
Exercícios de Fixação .....	37
Exercícios Complementares .....	37
Formulário .....	38

Exercícios de Fixação .....	46
Exercícios Complementares .....	47
CSS .....	48
1. CSS .....	49
O QUE É CSS?.....	49
2. SELETORES COMPLEXOS.....	50
<i>O que é um seletor?</i> .....	51
Exemplo de funcionamento .....	51
3. GRADIENTE .....	53
“Stops” ou definindo o tamanho do seu gradiente .....	54
4. COLUMNS .....	55
5. TRANSFORM 2D.....	56
CSS Transform na prática .....	57
6. TRANSIÇÕES E ANIMAÇÕES.....	59
<i>O básico: propriedade transition</i> .....	59
<i>Propriedade animation e regra keyframe</i> .....	61
<i>Definindo ciclos</i> .....	65
7. BORDAS .....	67
<i>Dividindo a imagem</i> .....	67
Comportamento da imagem .....	68
8. MÚLTIPLOS BACKGROUNDS.....	70
9. MÓDULO TEMPLATE LAYOUT .....	71
Sintaxe e funcionamento .....	72
O funcionamento da propriedade display .....	73
Definindo a largura e altura dos slots .....	73
O funcionamento da propriedade position .....	75
Pseudo-elemento ::slot().....	76
10. CORES .....	77
RGB.....	77
RGBA e a diferença da propriedade OPACITY .....	78
currentColor.....	79
11. PAGED MEDIA .....	80
@page .....	81
Terminologia e Page Model (modelo de página) .....	81
Propriedade size.....	83

Page-size.....	84
12. @FONT-FACE.....	85
<i>Compatibilidade</i> .....	86
13. PRESENTATION-LEVELS .....	87
A propriedade presentation-level.....	88
Javascript.....	91
INTRODUÇÃO .....	92
CONSIDERAÇÕES INICIAIS .....	93
Variáveis .....	95
Operadores.....	96
Objetos .....	98
VAR .....	100
Continue.....	101
Funções .....	101
Comentários .....	102
Estruturas de Controle .....	103
For... .....	103
If...Else.....	104
While .....	105
Funções internas .....	105
Objetos JavaScript .....	106
Select.....	109
Button.....	111
Navigator.....	112
Form .....	114
CheckBox .....	116
Document.....	118
Date .....	120
History .....	124
Window .....	125
Reset.....	127
Link .....	128
Palavras reservadas.....	130
Tabela de cores .....	130
Referências.....	140

# INTRODUÇÃO

Durante muito tempo, a ideia de desenvolvimento web ficou associada apenas a construção de páginas que possuíam somente o intuito de oferecer aos usuários o acesso a um determinado conteúdo.

Porém, com a popularização da internet, novas necessidades foram surgindo em diversas áreas.

Na área de entretenimento, cada vez mais jogos online foram aparecendo. Diversas redes sociais ganharam forças graças à grande interatividade permitida entre os usuários. Gravadoras de música passaram a vender seus títulos através de canais especializados. No meio corporativo, as empresas passaram a adotar sistemas web para controlar as suas tarefas administrativas. Enfim, necessidades antes inexistentes surgiram em uma velocidade muito grande. Muitos sites deixaram de ser simples páginas para se tornarem verdadeiras aplicações.

Há cerca de 15 anos, era muito comum que um único desenvolvedor fosse responsável por todo o desenvolvimento de uma aplicação web. Essa pessoa era chamada de webmaster. Com o passar do tempo, o papel do webmaster como era conhecido foi desaparecendo. A complexidade e volume de trabalho para o desenvolvimento de uma aplicação web se tornaram muito grande para apenas uma pessoa ou até mesmo para um grupo pequeno de desenvolvedores (webmasters).

Hoje, a função de webmaster ainda existe, mas com um papel um pouco diferente. Geralmente, esse profissional apesar de possuir bons conhecimentos nas diversas tecnologias utilizadas apenas gerencia o desenvolvimento que realizado por outros profissionais.

Como as tarefas antes de responsabilidade do webmaster foram delegadas a outros profissionais, naturalmente, apareceram algumas especializações. Essas especializações podem ser classificadas em dois grupos: desenvolvedores front-end e back-end. Em geral os desenvolvedores front-end são responsáveis pela interface com a qual os usuários interagem enquanto os desenvolvedores backend são responsáveis pelo funcionamento interno das aplicações.

## Client Side e Server Side

Os usuários acessam a interface de uma aplicação web através de navegadores (browsers). Os desenvolvedores front-end devem conhecer bem o funcionamento dos navegadores e das tecnologias e linguagens relacionadas a eles. Essas tecnologias e linguagens são categorizadas como **client side**. Atualmente, as principais linguagens e tecnologias client side são HTML, CSS, Javascript, Adobe Flash, Microsoft Silverlight e VBScript.

Por outro lado, os desenvolvedores back-end trabalham com linguagens como Java, C#, VB.NET, PHP, Ruby, Python, SQL entre outras. Essas linguagens atuam do lado do servidor por isso são classificadas como **server side**. Isso não significa que os desenvolvedores front-end não precisam conhecer as linguagens utilizadas pelo back-end e vice-versa. Na prática, ocorre uma especialização dos profissionais em determinadas tecnologias que podem tender mais para o front-end ou para o back-end.

## HTML, CSS e Javascript

Como acabamos de ver, as principais linguagens e tecnologias client side são HTML, CSS, Javascript, Adobe Flash, Microsoft Silverlight e VBScript. De todas elas as três primeiras são as mais importantes e atualmente estão em maior evidência. Cada uma das três linguagens possui um papel bem específico que podemos resumir da seguinte maneira: o código HTML será responsável por prover o conteúdo de uma página, o código CSS cuidará da formatação visual do conteúdo apresentado e o código Javascript permitirá que as páginas possuam algum tipo de comportamento (“inteligência”) e que alguma interação possa ser estabelecida com os usuários.

Nos próximos capítulos, discutiremos em detalhes cada uma dessas três linguagens.

# HTML

Quando acessamos uma página web, estamos interessados na informação contida nessa página.

Essa informação pode estar na forma de texto, imagem ou vídeo. Em geral, o conteúdo de uma página web é definido com a linguagem HTML (HyperText Markup Language). HTML é uma linguagem de marcação originalmente proposta por Tim Berners-Lee no final da década de 1980. O objetivo de Tim Berners-Lee era criar um mecanismo simples que pudesse ser utilizado por qualquer pessoa que quisesse disseminar documentos científicos.

Desde sua proposta até os dias de hoje, a linguagem HTML sofreu diversas alterações. A cada versão, novos recursos são adicionados e problemas corrigidos. A versão mais atual da especificação da linguagem HTML é a 5 (cinco). Essa versão ainda não foi finalizada, ela está na fase de “trabalho em progresso” (working draft). Porém, já existem navegadores implementando alguns dos novos recursos do HTML5.

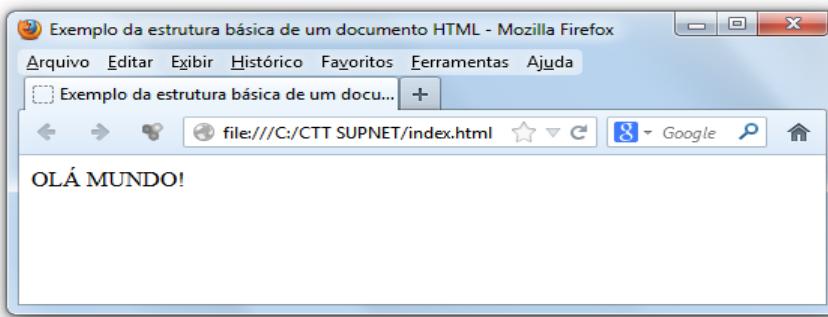
As especificações da linguagem HTML são publicadas pelo World Wide Web Consortium mais conhecido por sua sigla W3C. Além do HTML, o W3C também é responsável por linguagens como o XML, o SVG e pela interface DOM (Document Object Model), por exemplo.

## Estrutura Básica

Um documento HTML é composto por elementos que possuem uma tag, atributos, valores e possivelmente filhos que podem ser um texto simples ou outros elementos. Cada elemento deve obrigatoriamente possuir uma tag e ela deve ser definida entre parênteses angulares (< e >).

Veja o exemplo:

```
<html>
<head>
  <title>Exemplo da estrutura básica de um documento HTML </title>
</head>
<body>
  <p>Olá mundo !</p>
</body>
</html>
```



Código HTML 2.1: Exemplo da estrutura básica de um documento HTML

No exemplo acima, temos um elemento HTML representado pela tag “p” e um texto simples “Olá Mundo!” filho desse elemento.

## Exercícios de Fixação

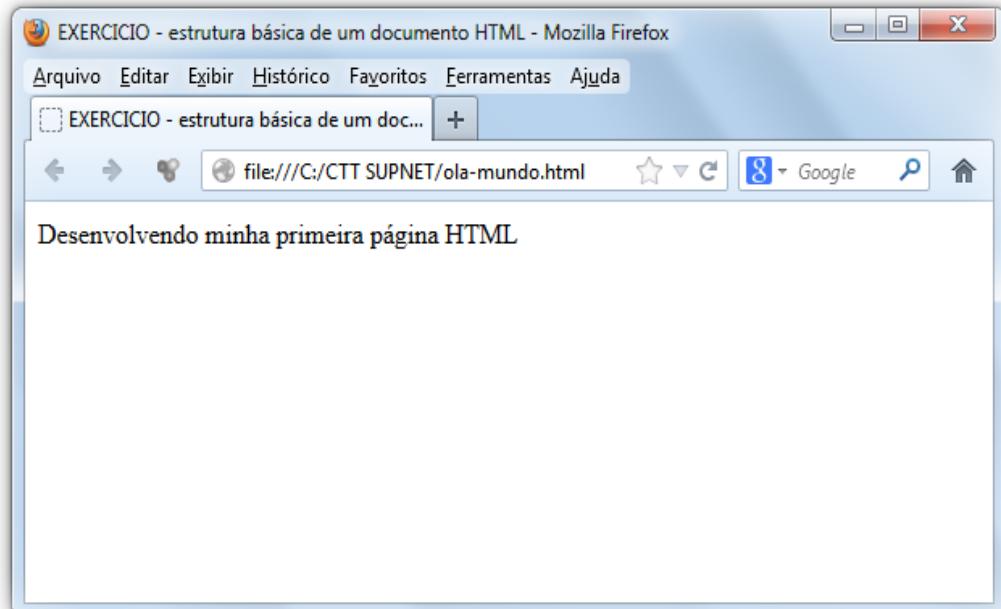
Na pasta **Desktop** do seu usuário, crie uma nova pasta com o seu primeiro nome. Dentro dessa pasta, crie um diretório chamado **html**. Para facilitar, utilize apenas letras minúsculas em todas as pastas e arquivos que criarmos durante o curso. Agora, utilizando um editor de texto, crie um novo arquivo chamado *ola-mundo.html* e salve-o dentro da pasta *html*.

Em seguida, insira o seguinte código dentro do arquivo *ola-mundo.html*:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>EXERCICIO - estrutura básica de um documento HTML </title>
</head>
<body>
<p>Desenvolvendo minha primeira página HTML</p>
</body>
</html>
```

Código HTML 2.2: *ola-mundo.html*

Abra o arquivo *ola-mundo.html* em um navegador e veja o resultado.



## Exercícios Complementares

1- Crie uma página HTML que exiba o nome deste curso duas vezes.

### Semântica HTML

De acordo com a especificação da linguagem HTML, cada elemento possui um propósito bem definido. Para o funcionamento correto das páginas de uma aplicação web, é fundamental respeitar o propósito de cada elemento e utilizá-lo nas condições corretas. Muitos autores utilizam o termo semântica HTML ao se referirem ao uso correto dos elementos da linguagem HTML. Por exemplo:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exemplo de uso correto da semântica HTML </title>
</head>
<body>
<p>Este é um texto para mostrar o significado da tag p.</p>
</body>
</html>
```

*Código HTML 2.4: Exemplo de uso correto da semântica HTML*

No exemplo acima, utilizamos o elemento p para definir um parágrafo. De acordo com a especificação da linguagem HTML, esse elemento deve ser utilizado justamente para definir parágrafos. Portanto, ele foi aplicado corretamente.

Agora, vejamos outro exemplo:

```
<html>
<head>
<title>Meus amigos - Site do Jonas - Criado pelo Jonas </title>
</head>
<body>
<address>
  Renan Santos
  renan.santos@supnet.com.br
  aluno da SUPNET Treinamentos
  Rua Frei Gaspar, 536 - Centro - São Vicente, SP
  CEP 11310-060
</address>
<address>
  Marcelo Mariano
  marcelo.mariano@supnet.com.br
</address>
```

```

instrutor da SUPNET Treinamentos
Av. Nações , 1533 - Vila Margarida - São Vicente, SP
CEP 11330-300
</address>
</body>
</html>

```

*Código HTML 2.5: Exemplo de uso incorreto da semântica HTML*

Dessa vez, utilizamos o elemento address. De acordo com a especificação, o elemento address deve ser utilizado para fornecer informações de contato dos autores do documento ou da maior parte do documento. Normalmente, esse elemento aparece no início ou no final das páginas.

Se observarmos o exemplo mais atentamente, trata-se de uma página do site do Jonas (repare no título da página). O autor da página é o Jonas e não o Rafael ou o Marcelo. Portanto, o elemento address foi aplicado incorretamente. Além disso, devemos evitar o uso desse elemento para informar endereços postais amenos que essas informações sejam relevantes para o documento.

## Parágrafos

Os parágrafos dentro de um documento HTML, em geral, são definidos através do elemento p. Uma das principais características desse elemento é que ele ocupa horizontalmente todo o espaço definido pelo elemento pai. Esse é o comportamento dos **elementos de bloco** que discutiremos com mais detalhes posteriormente.

Por enquanto, o importante é sabermos que, devido ao comportamento de bloco do elemento p, o navegador ajustará o texto do parágrafo à largura do elemento pai realizando todas as quebras de linha necessárias. Caso seja necessário forçar uma quebra de linha, podemos utilizar a elemento br.

Confira o exemplo:

```

<html>
<head>
<meta http - equiv =" Content - Type " content =" text / html ;
charset =UTF - 8">
<title>Exemplo de quebra de linha em um parágrafo </title>
</head>
<body>

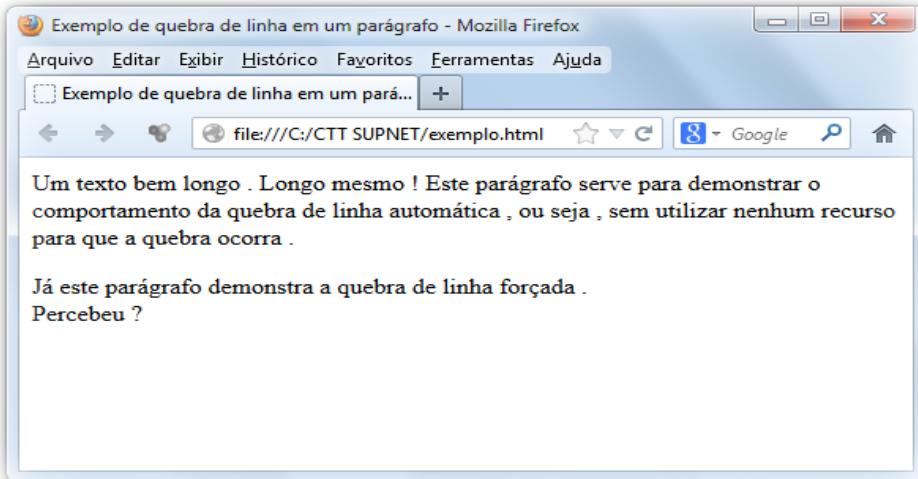
```

```

<p>Um texto bem longo. Longo mesmo! Este parágrafo serve para demonstrar o comportamento da quebra de linha automática, ou seja, sem utilizar nenhum recurso para que a quebra ocorra.</p>
<p>Já este parágrafo demonstra a quebra de linha forçada.<br/>Percebeu ?</p>
</body>
</html>

```

*Código HTML 2.6: Exemplo de quebra de linha em um parágrafo*



*Figura 2.2: Exemplo de quebra de linha em um parágrafo*

## Exercícios de Fixação

2- Crie um novo documento HTML chamado **p-quebra-de-linha.html** na pasta **html**. Em seguida, abra esse arquivo em um navegador (se necessário, redimensione a janela do navegador para verificar o comportamento da quebra de linha).

```

<html >
<head >
<meta http - equiv = " Content - Type " content = " text / html ;
charset =UTF - 8">
<title >Exemplo de quebra de linha em um parágrafo </ title >
</ head >
<body >
<p>Um texto bem longo. Longo mesmo! Este parágrafo serve para demonstrar o comportamento da quebra de linha automática, ou seja, sem utilizar nenhum recurso para que a quebra ocorra.</p>

<p>Já este parágrafo demonstra a quebra de linha forçada.<br/>Percebeu ?</p>
</ body >
</ html >

```

*Código HTML 2.7: p-quebra-de-linha.html*

## Exercícios Complementares

3- Crie um documento HTML e force uma quebra de linha em qualquer parte de um parágrafo.

Dica: utilize o site [www.lipsum.com](http://www.lipsum.com) para gerar um texto fictício.

### Cabeçalhos

Assim como em um livro, uma página HTML pode conter uma hierarquia de títulos para estabelecer uma divisão de seu conteúdo. Para conseguirmos realizar essa tarefa devemos utilizar as tags de cabeçalho h1, h2, h3, h4, h5 e h6.

Os sufixos numéricos de 1 a 6 indicam o nível do título dentro da hierarquia de títulos do documento.

Veja o exemplo:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exemplo de cabeçalhos </title>
</head>
<body>
<h1>Título 1</h1>
<h2>Título 2</h2>
<h3>Título 3</h3>
<h4>Título 4</h4>
<h5>Título 5</h5>
<h6>Título 6</h6>
</body>
</html>
```

*Código HTML 2.9: Exemplo de cabeçalhos*

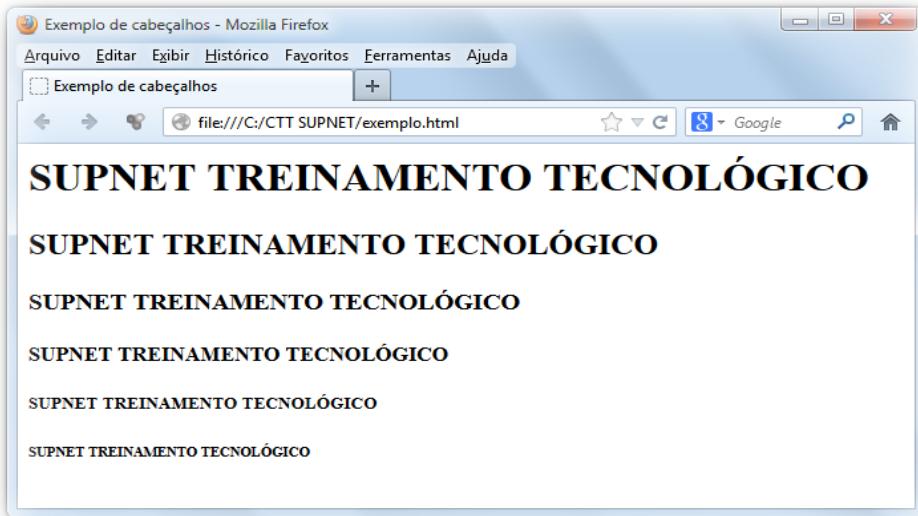


Figura 2.3: Exemplo de cabeçalhos

Perceba que cada nível possui um tamanho diferente de fonte. Esse tamanho é determinado pelo navegador e pode ser alterado através de regras CSS que veremos posteriormente.

Devemos utilizar os cabeçalhos com cautela, pois eles são utilizados como parâmetros de ranqueamento da página por diversos buscadores como Google, Yahoo e Bing, por exemplo. O uso correto das tags de cabeçalho faz parte das técnicas de SEO (Search Engine Optimization) que, como o próprio nome já indica, são técnicas que ajudam a melhorar o ranqueamento de páginas dentro dos buscadores.

De acordo com as técnicas de SEO devemos tomar os seguintes cuidados ao utilizarmos as tags de cabeçalhos:

- Utilizar apenas uma tag `<h1>` por página.
- Utilizar no máximo duas tags `<h2>` por página.

## Exercícios de Fixação

4- Crie um novo documento HTML chamado **cabecalhos-1.html** com o conteúdo abaixo na pasta *html*. Em seguida, abra esse arquivo em um navegador.

```
<html >
<head >
<title >SUPNET - Desenvolvimento Web com HTML , CSS e Javascript </
title >
</ head >
<body >
<h1 >SUPNET - Desenvolvimento Web com HTML, CSS e Javascript </h1>
<p>Atualmente, praticamente todos os sistemas corporativos possuem
interfaces web. Para quem deseja atuar no mercado de
desenvolvimento de software, é obrigatório o conhecimento das
linguagens: HTML, CSS e JavaScript.</p>
<p>Essas linguagens são utilizadas para o desenvolvimento da camada
de apresentação das aplicações web.</p>
<h2 >Pré - requisitos </h2 >
<p> Familiaridade com algum sistema operacional (Windows / Linux /
Mac OS X)</p>
<p>Lógica de programação </p>

<h2 >Agenda </h2 >

<h3 >De Segunda a Sexta </h3 >
<p>xx/xx/ xxxx das 08:00 às 12:00 </p>
<p>xx/xx/ xxxx das 13:00 às 17:00 </p>
<p>xx/xx/ xxxx das 18:00 às 22:00 </p>

<h3>Aos sábados </h3>
<p>xx/xx/ xxxx das 08:00 às 12:00 </p>
<p>xx/xx/ xxxx das 13:00 às 17:00 </p>
</body>
</html>
```

Código HTML 2.10: *cabecalhos-1.html*

5- Imagine que você possua um site de culinária no qual você disponibiliza algumas receitas. Crie uma página com uma receita fictícia utilizando cabeçalhos para organizar conteúdo. Utilize quantos níveis de título achar necessário.

```

<html>
<head>
<title>Como preparar um delicioso macarrão instantâneo em 6 min.</title>
</head>
<body >
<h1>Como preparar um delicioso macarrão instantâneo em 6 min.</h1>

<p>Com esta receita você se tornará um profissional na arte de preparar um macarrão instantâneo.</p>

<h2> Ingredientes </h2>

<p>Macarrão instantâneo de sua marca favorita </p>
<p>600 ml de água </p>

<h2>Modo de preparo</h2>

<h3>No microondas</h3>

<p>Insira o macarrão instantâneo em um recipiente com 600 ml de água e programe o microondas por 6 minutos. Aperto o botão iniciar ou equivalente.</p>

<h4>Ponto importante </h4>
<p>Utilize um recipiente que permita o macarrão ficar totalmente submerso na água.</p>
<p>Quando ouvir o bip não saia correndo. O microondas não irá explodir, pois o bip significa que o macarrão está pronto.</p>

<h3>No fogão </h3>

<p>Ferva a água em uma panela.</p>
<p>Insira o macarrão e cozinhe-o por 3 minutos </p>

<h4>Ponto importante</h4>

<p>Utilize uma panela que permita o macarrão ficar totalmente submerso na água.</p>
<p>Não se distraia com a televisão ou qualquer outra coisa. Você poderá queimar a sua refeição </p>
</body>
</html>
```

*Código HTML 2.11: macarao.html*

## Exercícios Complementares

- 6- Utilizando as tags de cabeçalho, crie uma página HTML que exiba hierarquicamente o nome e uma curiosidade de alguns continentes, países, estados (províncias) e cidades.
- 7- Utilizando as tags de cabeçalho, crie uma página de um produto e suas especificações, observações e ou comentários. Utilize quantos níveis de título achar necessário.

## Links

Normalmente, um site é formado por um conjunto de páginas que estão interligadas de alguma forma. Para permitir que um usuário navegue de uma página para outra, devemos utilizar os links. Um link pode fazer a ligação de uma página para outra do mesmo site (link interno) ou para uma página de outro site (link externo).

Para criarmos um link, devemos utilizar a tag a. Porém, essa tag sem atributos não criará nenhum link interno ou externo. Para que um link seja criado, devemos, no mínimo, utilizar o atributo href com o caminho relativo ou absoluto de outra página. Veja o exemplo:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Exemplo de uso da tag a</title>
</head>
<body>
<p><a href="pagina2.html">Exemplo de link relativo </a></p>
<p><a href="outros/pagina3.html">Outro exemplo de link relativo
</a></p>
<p><a href="http://www.supnet.com.br">Exemplo de link absoluto
</a></p>
</body>
</html>
```

Código HTML 2.14: Exemplo de uso da tag a

Além do atributo href, podemos utilizar o atributo target para informar onde o documento deve ser aberto. Os possíveis valores para o atributo target são:

- \_blank - em uma nova janela ou aba
- \_self - na mesma janela ou frame do documento que contém o link
- \_parent - em um frame que seja o “pai” do frame no qual o link se encontra

- \_top - na mesma janela do documento que contém o link

Ao testar os valores acima, logo percebemos que \_self e \_top possuem o mesmo comportamento se a página que contém o link não estiver em um frame. Caso o link esteja em um frame e com o valor \_top no atributo target, o link será aberto na janela na qual o frame se encontra. Se o valor for \_self, o link será aberto no próprio frame.

Dentro de uma única página podemos ter diversos frames e, às vezes, queremos que um link de um determinado frame seja aberto em outro. Para realizarmos tal tarefa devemos inserir como o valor do atributo target o nome do frame no qual o link será aberto.

O comportamento padrão de um link é abrir o documento na mesma página ou frame caso o atributo target não seja utilizado.

```
<html >
<head >
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<body >
<p><a href="pagina1.html" target="_blank">Abre em outra
janela / aba </a></p>
<p><a href="pagina2.html" target="_self">Abre na mesma janela
</a></p>
<p><a href="pagina3.html">Abre na mesma janela </a></p>
</ body >
</ html >
```

Código HTML 2.15: Exemplo de uso da tag a com o atributo target



### Importante

Ao longo da evolução do HTML, a tag frame foi caindo em desuso até que no HTML5 foi totalmente retirada da especificação. Contudo a grande maioria dos navegadores ainda interpreta a tag corretamente mesmo dentro de um documento HTML5. Porém, devemos nos lembrar de que ainda estamos num momento de transição para o HTML5. Logo, para evitar problemas no futuro, evite o uso da tag frame ao máximo.

## Exercícios de Fixação

8- Crie dois documentos HTML em arquivos chamados **pagina1.html** e **pagina2.html** dentro da pasta **html** e em seu corpo crie quatro links: um que aponte para uma página externa e outros três que apontem para uma página interna de maneiras diferentes. Lembre-se de criar também a página para a qual o seu link interno irá apontar.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exemplo de uso da tag a com o atributo target </title>
</head>
<body>
<p><a href="http://www.supnet.com.br" target="_blank">Link externo</a></p>
<p><a href="pagina2.html" target="_self">Link interno</a></p>
<p><a href="pagina2.html" target="_top">Link interno</a></p>
<p><a href="pagina2.html">Link interno</a></p>
</body>
</html>
```

Código HTML 2.16: *pagina1.html*

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exemplo de uso da tag a com o atributo target </title>
</head>
<body>
<h1>Página 2</h1>
</body>
</html>
```

Código HTML 2.17: *pagina2.html*

## Exercícios Complementares

9- Crie vários links em uma página e para cada link escolha um target diferente. Crie também quantas páginas de destino forem necessárias (caso seja necessário).

10- Pesquise na internet como criar um iframe dentro de um documento HTML. Em seguida, crie uma página com alguns links e um iframe. Crie também alguns links na página contida no iframe. Para cada link em ambas as páginas utilizem valores diferentes para o atributo target e observe os resultados.

## Âncoras

Além de criar links para outras páginas, podemos criar links para uma determinada seção dentro da própria página na qual o link se encontra ou dentro de outra página. Esse recurso chama-se **ancoragem**, pois as seções para as quais queremos criar um link devem possuir uma **âncora**.

Para criarmos uma âncora devemos utilizar novamente a tag a, porém sem o atributo href. Dessa vez utilizaremos o atributo name para identificar a seção através de um nome.

O link também muda levemente, pois agora ao invés de passar somente o nome do arquivo da página como valor do atributo href deverá passar o nome da seção prefixada com o caractere #.

```
<html>
<head>
<title>Exemplo de uso da tag a como âncora </title>
</head>
<body>
<p><a href = "# mais_info">Veja mais informações </a></p>
<p><a href = " pagina2 . html # outra_ancora">Âncora em outra
página </a></p>
...
...
...
<a name = " mais_info">Mais informações </a>
<p>
...
...
...
</p>
</body>
</html>
```

*Código HTML 2.22: Exemplo de uso da tag a como âncora*

**Lembre-se**

Até a versão 4 do HTML e no XHTML, a especificação dizia para utilizarmos o atributo name da tag a para criarmos as âncoras. Porém, no HTML5, a recomendação do W3C é que o atributo id seja utilizado para tal propósito. Desenvolvedores mais preocupados em estar sempre atualizados podem ficar tranquilos e utilizar somente o atributo id ao invés do name, pois os navegadores mais modernos conseguem interpretar o uso de ambos os atributos em qualquer versão do HTML.

## Exercícios de Fixação

11- Crie um documento HTML em um arquivo chamado **ancora-pagina1.html** na pasta **html** que contenha um link que aponta para uma âncora dentro da própria página. Dica: insira um conteúdo suficientemente grande para que a barra de rolagem vertical do navegador apareça e coloque a âncora no final da página.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exercício sobre âncoras </title>
</head>
<body>
<p><a href="#sobre">Sobre o texto dessa página </a></p>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec egestas dolor quis turpis dictum tincidunt. Donec blandit tempus velit, sit amet adipiscing velit consequat placerat. Curabitur id mauris.</p>
<p> ... </p>
<p> ... </p>
<p> ... </p>

<p>At nisi imperdiet lacinia. Ut quis arcu at nisl ornare viverra. Duis vel tristique tellus. aecenas ultrices placerat tortor. Pellentesque feugiat accumsan commodo. Proin non urna justo, id pulvinar lacus.</p>

<a name="sobre">Sobre o texto dessa página </a>
<p>O texto dessa página foi gerado através do site:<br/>
<a href="http://www.lipsum.com/">http://www.lipsum.com</a></p>
</body>
</html>
```

*Código HTML 2.23: ancora-pagina1.html*

12- Crie um novo arquivo chamado **ancora-pagina2.html** na pasta **html** com um âncora chamada **outra\_ancora**. Dica: insira um conteúdo suficientemente grande para que a barra de rolagem vertical do navegador apareça e coloque a âncora no final da página.

```
<html >
<head >
<title >Exercício sobre âncoras </ title >
</ head >
<body >
<h1 >Ancora página 2</h1 >

<p>Lorem ipsum dolor sit amet , consectetur adipiscing elit . Donec justo massa, sodales sit amet eleifend a, elementum eu nibh. Donec, sit amet adipiscing velit consequat placerat . Curabitur id mauris </p>
<p> ... </p>
<p> ... </p>
<p> ... </p>

<p> at nisi imperdiet lacinia . Ut quis arcu at nisl ornare viverra. Duis vel tristique tellus . Maecenas ultrices placerat tortor . Pellentesque feugiat accumsan commodo . Proin non urna justo , id pulvinar lacus .</p>

<a name =" outra_ancora ">Mais uma âncora </a>

<p>Se você chegou aqui deu tudo certo! :) </p>
</ body >
</ html >
```

*Código HTML 2.24: ancora-pagina2.html*

Crie um novo link no arquivo **ancora-pagina1.html** que aponte para âncora **outra\_ancora** do arquivo **ancora-pagina2.html**.

```
<html >
<head >
<title >Exercício sobre âncoras </ title >
</ head >
<body >
<p><a href ="# sobre ">Sobre o texto dessa página </a></p>
<p><a href =" ancora - pagina2 . html # outra_ancora ">Âncora em outra página </a></p>

<p>Lorem ipsum dolor sit amet , consectetur adipiscing elit . Donec justo massa , sodales sit amet eleifend a, elementum eu nibh . Donec egestas dolor quis turpis dictum tincidunt . Donec blandit tempus velit , sit amet adipiscing velit consequat placerat . Curabitur id mauris .</p>
<p> ... </p>
<p> ... </p>
<p> ... </p>

<p>At nisi imperdiet lacinia . Ut quis arcu at nisl ornare viverra. Duis vel tristique tellus. Maecenas ultrices placerat tortor .
```

```
Pellentesque feugiat accumsan commodo . Proin non urna justo , id
pulvinar lacus .</p>
```

```
<a name =" sobre ">Sobre o texto dessa página </a>
<p>O texto dessa página foi gerado através do site:
<a href =" http :// www . lipsum . com /">http://www.lipsum.com
/<a></p>
</ body >
</ html >
```

*Código HTML 2.25: ancora-pagina1.html*

## Exercícios Complementares

13- Crie dois documentos HTML. No primeiro crie um link que aponte para o site da SUPNET e também coloque um texto qualquer. Em qualquer ponto deste texto, crie uma âncora. No segundo documento crie um link que aponte para a âncora criada no primeiro documento, coloque também um link qualquer ou um texto.

14- Dentro de um documento HTML crie três links. Dois devem apontar para âncoras de páginas externas e o último link deve aparecer no final da página e apontar para uma âncora no topo da própria página.

## Imagens

Certamente, os sites na internet seriam muito entediantes se não fosse possível adicionar imagens ao conteúdo das páginas. Como não queremos que as nossas páginas fiquem muito monótonas, neste capítulo, utilizaremos a tag img para melhorar um pouco a aparência das páginas com algumas imagens.

A tag img possui o atributo src que utilizaremos para informar qual imagem queremos carregar dentro de um documento HTML. O valor do atributo pode ser o caminho absoluto ou relativo de uma imagem.

```
<html>
<head>
<meta http - equiv =" Content - Type " content =" text / html ;
charset =UTF - 8">
<title>Exemplo de uso da tag img </title>
</head>
<body>
<h1>SUPNET TREINAMENTOS</h1>
<img src ="http://www.supnet.com.br/img/escrita-sup.png" />
<h2>Cursos </h2>
```

```
<p>
<img src ="S01.png" />
Curso 01 - Lógica de Programação
</p>
<p>
<img src ="S02.png" />
  Curso 02 - Desenvolvimento Web com HTML, CSS e JavaScript
</p>
<p>
<img src ="S03.png"/>
  Curso 03 - Curso Melhor Idade
</p>
<p>
<img src ="S04.png" />
  Curso 04 - Desenvolvimento .NET
</p>
...
...
...
</body>
</html>
```

Código HTML 2.29: Exemplo de uso da tag img

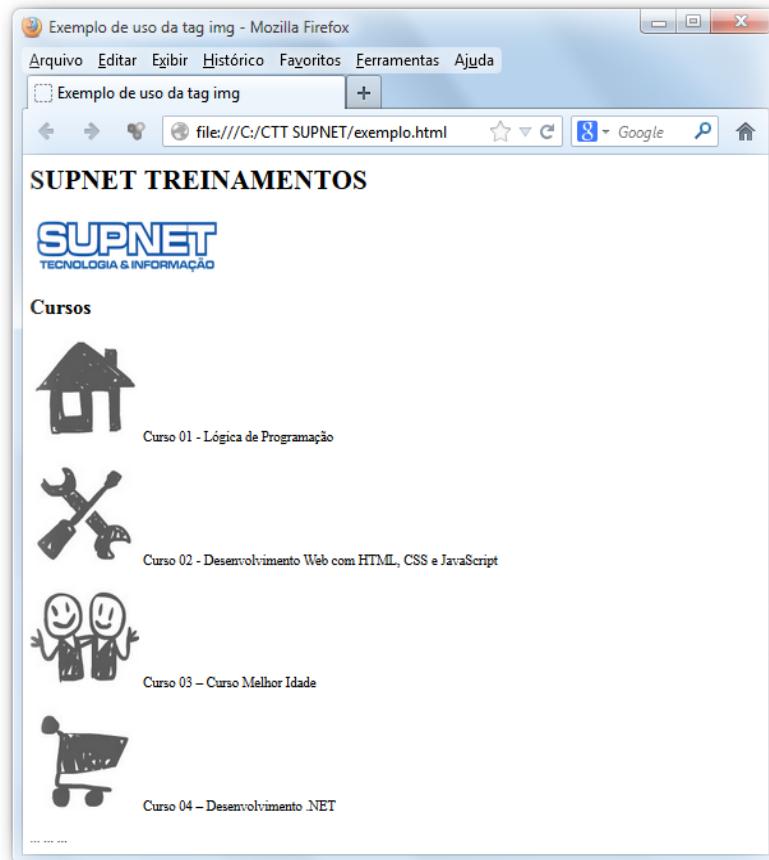


Figura 2.4: Exemplo de uso da tag img

## Exercícios de Fixação

**15-** Crie um documento HTML em um arquivo chamado **imagem.html** na pasta **html** que contenha alguns elementos IMG.

```
<html>
<head>
<title>Invenções que marcaram a história da tecnologia </title>
</head>
<body>

<h1>Transistor</h1>
<p>
<br>O termo transistor, cunhado por John R. Pierce para definir um resistor de transferência, se popularizou na década de 50 e possibilitou a evolução dos eletrônicos.</p>

<h2>Apple</h2>
<p>
<br>Em 5 de junho de 1977, Steve Wozniak e Jobs começaram a vender as primeiras unidades do computador de 8-bits que marcaria o início da computação pessoal </p>

<h2>Atari</h2>
<p>
<br>Lançado em 1977, o Atari chegou ao Brasil em 1983 com o modelo 2600. Não demorou muito para o pequeno console preto ganhar as multidões e invadir várias casas</p>

<h2>ICQ</h2>
<p>
<br>Fundada em 1996 por cinco israelenses, a Mirabilis passou a ser conhecida em todo o mundo depois de publicar a primeira versão do ICQ, em novembro do mesmo ano. O comunicador instantâneo fornecia um número para o usuário e permitia que ele se conectasse a outros para a troca de mensagens</p>

</body>
</html>
```

*Código HTML 2.30: imagem.html*

## 💻 Exercícios Complementares

**16 –** Em um documento HTML insira no mínimo duas imagens utilizando o elemento IMG.

## Tabelas

Suponha que você esteja desenvolvendo o site de uma empresa e precisa apresentar alguns relatórios em páginas HTML. Considere que os dados desses relatórios são obtidos de planilhas eletrônicas.

Daí surge a necessidade de definir dados de forma tabular em HTML. Para resolver esse problema, podemos utilizar o elemento `table` do HTML. Esse elemento permite apresentar um conjunto de dados na forma de tabelas. Veja o exemplo:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exemplo de uso da tag table </title>
</head>
<body>
<h1>Carros </h1>

<table>
<tr>
<th>Marca </th>
<th>Modelo </th>
<th>Ano </th>
</tr>
<tr>
<td>Toyota </td>
<td>Corolla </td>
<td>2013 </td>
</tr>
<tr>
<td>Honda </td>
<td>Civic </td>
<td>2013 </td>

</tr>
<tr>
<td>Mitsubishi </td>
<td>Lancer </td>
<td>2013 </td>
</tr>
<tr>
<td colspan="3">Última atualização : 06/2012 </td>
</tr>
</table>
</body>
</html>
```

*Código HTML 2.32: Exemplo de uso da tag table*

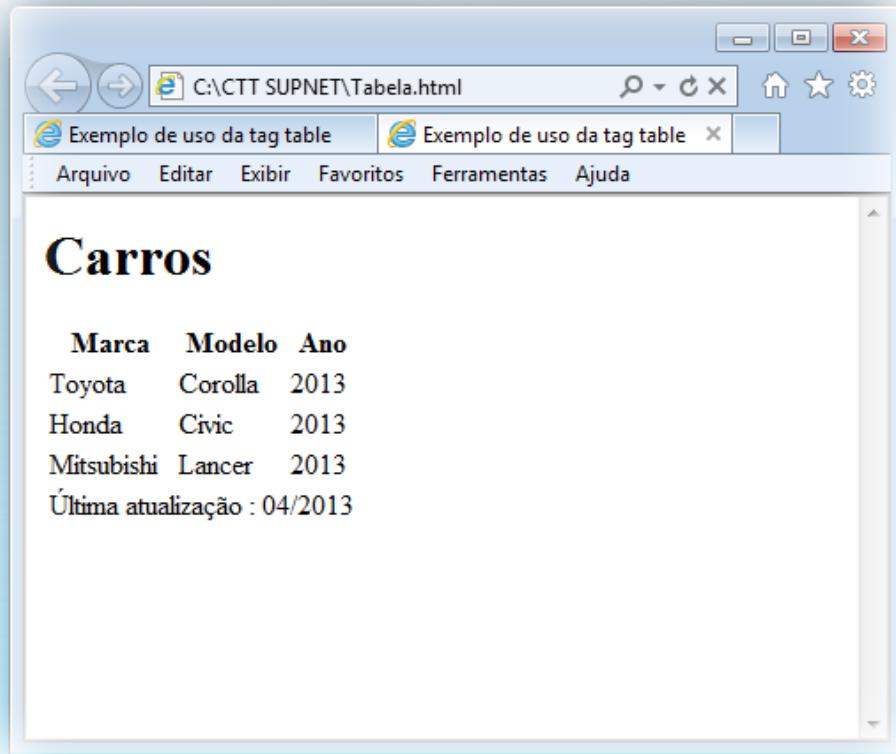


Figura 2.5: Exemplo de uso da tag table

Perceba que a tag table não é utilizada sozinha. Ela necessita de pelo menos um ou mais elementos tr que, por sua vez, necessitam de pelo menos um ou mais elementos th ou td.

O que significam essas tags?

- tr - define uma linha da tabela
- th - define uma célula de cabeçalho
- td - define uma célula

Existe uma outra forma de criar a mesma tabela:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html ;
charset=UTF-8">
<title>Exemplo de uso da tag table </title>
</head>
<body>
<h1>Carros </h1>

<table>
  <thead>
```

```

<tr>
<th>Marca </th>
<th>Modelo </th>
<th>Ano </th>
</tr>
</thead>
<tfoot>
<tr>
<td colspan ="3">Última atualização : 04/2013 </td >
</tr>
</tfoot>
<tbody>
<tr
<td>Toyota </td>
<td>Corolla </td>
<td>2013 </td>
</tr>
<tr>
<td >Honda </td>
<td>Civic </td>
<td>2012 </td>
</tr >
<tr >
<td>Mitsubishi </td>
<td>Lancer </td>
<td>2012 </td>
</tr>
</body>
</table>
</body>
</html>

```

*Código HTML 2.33: Exemplo de uso da tag table*



*Figura 2.6: Exemplo de uso da tag table*

Repare que visualmente não mudou absolutamente nada. Além disso apareceram mais algumas tags: thead, tfoot e tbody.

### O que significam essas tags?

- thead - define o cabeçalho da tabela
- tfoot - define o rodapé da tabela
- tbody - define o corpo da tabela

### Por que complicar? Qual o motivo da existência dessas tags?

- A tag thead, assim como a tfoot, servem para agrupar as linhas de cabeçalho e de rodapé, respectivamente.
- O código fica mais claro.
- Facilita a aplicação de estilos CSS (através do seletor de elemento)
- Pode permitir que o conteúdo do corpo da tabela possuísse rolagem\*.
- Ao imprimir a página com uma tabela muito extensa, pode permitir que o cabeçalho e rodapé fossem replicados em todas as páginas\*.

\* Esses recursos podem existir ou não, pois os desenvolvedores de navegadores podem decidir não programá-los. Esses recursos são sugestões da especificação.

Outros dois atributos importantes para a construção de tabelas são colspan e rowspan que podem ser aplicados nos elementos td e th.

Como podemos observar nos exemplos dados, o atributo colspan faz com que a célula ignore o número de colunas definidas em seu valor. Analogamente, o atributo rowspan faz o mesmo, porém com linhas.

## Exercícios de Fixação

**17** - Crie uma página HTML em um arquivo chamado **tabela.html** na pasta **html** que contenha uma tabela de acordo com a imagem abaixo:

Marca	Modelo	Ano
Toyota	Corolla	2010
	Camry	2011
Honda	Civic	2004
	Fit	2012
	City	2011
Mitsubishi	Lancer	2012

Última atualização : 04/2013

Figura 2.7: Exercício para a tag table

As linhas vermelhas foram colocadas na imagem apenas para facilitar a visualização do problema.

```
<html>
<head>
<title >Exercício para a tag table </ title >
</ head>
<body>
<table>
<thead>
<tr>
<th>Marca </th>
<th>Modelo </th>
<th>Ano </th>
</tr>
</thead>
<tfoot>
<tr>
<td colspan ="3">Última atualização : 04/2013 </td >
</tr>
</tfoot>
<tbody>
<tr>
<td rowspan ="2">Toyota </td>
<td>Corolla </td>
<td>2010 </td>
</tr>
<tr>
<td>Camry </td>
<td>2011 </td>
</tr>
<tr >
<td rowspan ="3">Honda </td>
<td>Civic </td>
<td>2004 </td>
</tr>
<tr >
<td>Fit </td>
<td>2012 </td>
</tr>
<tr >
<td >City </td >
<td >2011 </td >
</tr >
<tr >
<td >Mitsubishi </td >
<td >Lancer </td >
<td >2012 </td >
</tr>
</tbody>
</table>
</body>
</html>
```

Código HTML 2.34: tabela.html

## Exercícios Complementares

**18** - Crie em um documento HTML uma tabela que contenha o continente / subcontinente, o nome e o idioma de algumas cidades do mundo.

### Listas

Em um documento HTML podemos ter três tipos de listas e cada uma delas deve ser utilizada de acordo com a sua semântica, ou seja, você deve escolher um tipo de lista para cada situação.

Os três tipos possíveis de listas são:

- Lista de definição - utilizada para exibir definições de termos. Funciona como nos dicionários, no qual temos uma palavra e em seguida o seu significado.
- Lista ordenada - utilizada para exibir qualquer conteúdo de forma ordenada.
- Lista sem ordem - utilizada para exibir qualquer conteúdo sem ordenação.

#### **Lista de definição**

Para criarmos uma lista de definição devemos utilizar a tag dl. O elemento dl deve possuir pelo menos um elemento filho dt seguido de um elemento dd. Um item em uma lista de definição é composto por um par de elementos dt e dd.

```
<html>
<head>
<title>Exemplo de uso da tag dl </ title >
</head>
<body>
<h1>SUPNET Treinamentos </h1>
<dl>
<dt>Lógica de Programação </dt>
<dd>
Conhecimentos em Lógica de Programação é o pré - requisito fundamental para que uma pessoa consiga aprender qualquer Linguagem de Programação ...
</dd>
<dt>Desenvolvimento Web com HTML, CSS e JavaScript </dt>
<dd>
Atualmente, praticamente todos os sistemas corporativos possuem interfaces web . Para quem deseja atuar no mercado de desenvolvimento
</dd>
<dt>SQL e Modelo Relacional </dt>
<dd>
```

Como as aplicações corporativas necessitam armazenar dados é fundamental que os desenvolvedores possuam conhecimentos sobre persistência de dados.

```
</dd>
</dl>
</body>
</html>
```

*Código HTML 2.36: Exemplo de uso da tag dl*



*Figura 2.8: Exemplo de uso da tag dl*

## Exercícios de Fixação

**19** - Crie um documento HTML em um arquivo chamado **restaurante.html** na pasta **html** que contenha o cardápio de um restaurante com os nomes dos seus pratos e uma breve descrição sobre os mesmos.

```
<html>
<head>
<title>Menu - Pizzaria </title>
</ head>
<body>
<h1>Menu - Pizzaria CTT </h1>

<dl>
<dt>À moda da casa </dt>
<dd>
Presunto coberto com mussarela , ovos e palmito .
</dd>
<dt>À moda do pizzaiolo </dt>
<dd>
Mussarela , presunto , ovos e bacon .
</dd>
<dt>Aliche </dt>
<dd >
```

```
Aliche , parmesão e rodelas de tomate .
</dd>

</dl>
</body>
</html>
```

*Código HTML 2.37: restaurante.html*

## Exercícios Complementares

**20** - Crie um documento HTML que contenha uma lista de alguns pontos turísticos do Brasil de que você tenha conhecimento e cite algumas atrações do mesmo.

### **Lista ordenada**

Para criarmos uma lista ordenada, devemos utilizar a tag ol. O elemento ol deve possuir pelo menos um elemento filho li.

```
<html>
  <head>
    <title>Exemplo de uso da tag ol </title>
  </head>
  <body>
    <h1>Macarrão instantâneo - SUPNET Receitas </h1>
    <h2>Modo de preparo </h2>

    <ol>
      <li>Ferver 600 ml de água em uma panela .</li >
      <li>Retirar o macarrão do pacote .</li >
      <li>Colocar o macarrão na panela no fogo baixo .</li >
      <li>Cozinhar o macarrão por 3 min .</li >
      <li>Temperar a gosto .</li >
    </ol >
  </body>
</html>
```

*Código HTML 2.39: Exemplo de uso da tag ol*

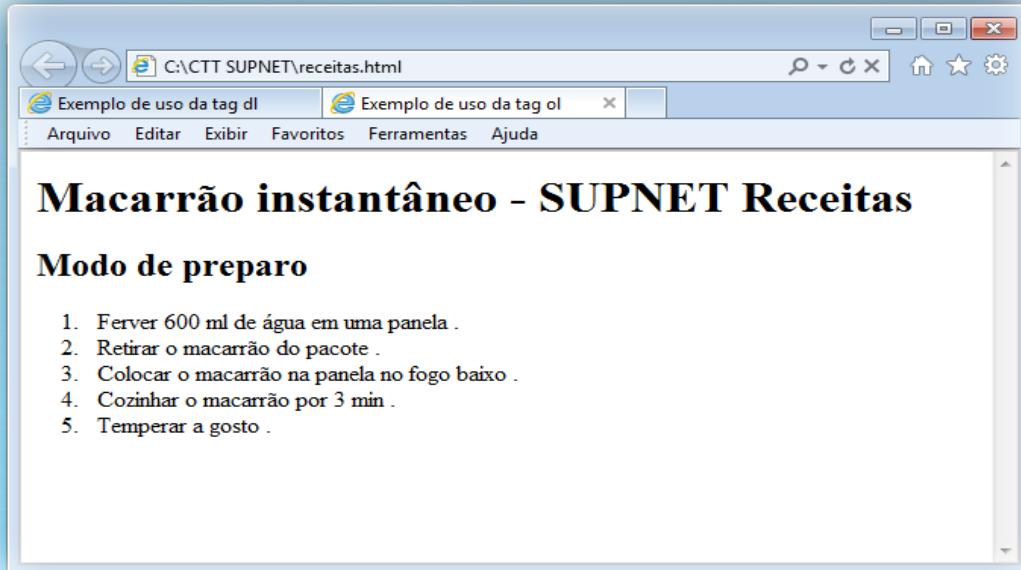


Figura 2.9: Exemplo de uso da tag ol

## Exercícios de Fixação

**21** - Crie um documento HTML em um arquivo chamado **manual.html** na pasta **html** que contenha um manual que explica passo-a-passo o uso de um caixa eletrônico para a operação de saque.

```
<html>
<head>
<title>Operação de saque - Manual do caixa eletrônico</title>
</head>
<body>
<h1>Operação de saque - Manual do caixa eletrônico</h1>

<ol>
<li>Insira o cartão </li>
<li>Digite a senha </li>
<li>Escolha a opção de saque </li>
<li>Informe o valor que deseja sacar </li>
<li>Insira o cartão novamente </li>
<li>Aguarde até a liberação do dinheiro </li>
</ol>
</body>
</html>
```

Código HTML 2.40: *manual.html*

## Exercícios Complementares

22 - Crie um documento HTML que contenha um manual que explique passo-a-passo a instalação, manutenção ou manuseio de um aparelho eletrônico.

### Listas sem ordem

Para criarmos uma lista sem ordem, devemos utilizar a tag ul. O elemento ul deve possuir pelo menos um elemento filho li.

```
<html>
<head>
<title>Exemplo de uso da tag dl </title>
</head>
<body>
<h1>K02 - Desenvolvimento Web com HTML , CSS e JavaScript </h1>
<h2>Pré - requisitos </h2>

<ul>
<li> Conhecimento de algum sistema operacional ( Windows / Linux / Mac OS X )</li>
<li> Lógica de programação </li>
</ul>
</body>
</html>
```

*Código HTML 2.42: Exemplo de uso da tag ul*



*Figura 2.10: Exemplo de uso da tag ul*

## Exercícios de Fixação

**23** - Crie um documento HTML em um arquivo chamado **lista-curso.html** na pasta **html** que contenha a lista dos cursos da Formação Básica da SUPNET.

```
<html>
<head>
<title> Formação Básica - SUPNET Treinamentos </title>
</head>
<body>
<dl>
<h1> Formação Básica </h1>

<ul>
<li>Curso 01 - Lógica de Programação </li>
<li>Curso 02 - Desenvolvimento Web com HTML , CSS e JavaScript
</li>
<li>Curso 03 - SQL e Modelo Relacional </li>
</ul>
</dl>
</body>
</html>
```

*Código HTML 2.43: lista-cursos.html*

## Exercícios Complementares

**24** - Crie um documento HTML que contenha a lista dos cursos da Formação Desenvolvedor Java da SUPNET.

## Formulário

Para trazermos um pouco mais de interatividade para as nossas páginas, podemos utilizar os elementos de formulário. Esses elementos recebem algum tipo de entrada do usuário, seja através de um clique ou digitando algum valor.

### A tag input

A tag input permite que o elemento que a contenha assuma diversas formas dependendo do seu atributo type. O atributo type pode receber os seguintes valores:

- text - cria uma caixa de texto de uma linha.
- password - cria uma caixa de texto de uma linha escondendo os caracteres digitados.
- checkbox - cria uma caixa que assume dois estados: checado e "deschecado". Em conjunto com o atributo name é possível que se crie um grupo de checkboxes no qual um ou mais checkboxes
  - ✓ seja "checado".
- radio - cria uma caixa que assume dois estados: checado e "deschecado". Em conjunto com o atributo name é possível que se crie um grupo de radios no qual apenas um radio seja "checado".
- button - cria um botão. Através do atributo value definimos o texto do botão.
- submit - cria um botão para o envio do formulário. Através do atributo value definimos o texto do botão.
- file - cria um botão que, ao ser clicado, abre uma caixa de diálogo para a escolha de um arquivo
- ✓ no computador do usuário.
  - reset - cria um botão que descarta todas as alterações feitas dentro de um formulário. Através
- ✓ do atributo value definimos o texto do botão.
  - image - cria um botão para o envio do formulário. Dese ser utilizado em conjunto com o atributo
- ✓ src para que uma imagem de fundo seja utilizada no botão.
  - hidden - cria um elemento que não fica visível para o usuário, porém pode conter um valor que será enviado pelo formulário.

Existem outros valores possíveis para o atributo type, porém eles fazem parte da especificação do HTML5 e nem todos os navegadores suportam tais valores.

```

<html>
  <head>
    <title>Exemplo de uso da tag input </title>
  </head>
  <body>
    <form action = " pagina . html " method = " get ">
      <p>
        text :
        <input type = " text " />
      </p>
      <p>
        password :
        <input type = " password " />
      </p>
      <p>
        checkboxes :
        <input type = " checkbox " name = " checkgroup " />
        <input type = " checkbox " name = " checkgroup " />
        <input type = " checkbox " name = " checkgroup " />
      </p>
      <p>
        radios :
        <input type = " radio " name = " checkgroup " />
        <input type = " radio " name = " checkgroup " />
        <input type = " radio " name = " checkgroup " />
      </p>
      <p>
        button :
        <input type = " button " value = " Botão " />
      </p>
      <p>
        submit :
        <input type = " submit " value = " Enviar " />
      </p>
      <p>
        file :
        <input type = " file " />
      </p>
      <p>
        reset :
        <input type = " reset " value = " Descartar alterações " />
      </p>
      <p>
        image :
        <input
          type = " image "
          src = " http://www.supnet.com.br/css/img/main-
          header-logo.png "
        >
      </p>
      <p>
        hidden :
        <input type = " hidden " value = " valor escondido " />
      </p>
    </form>
  </body>
</html>

```

Código HTML 2.45: Exemplo de uso da tag input

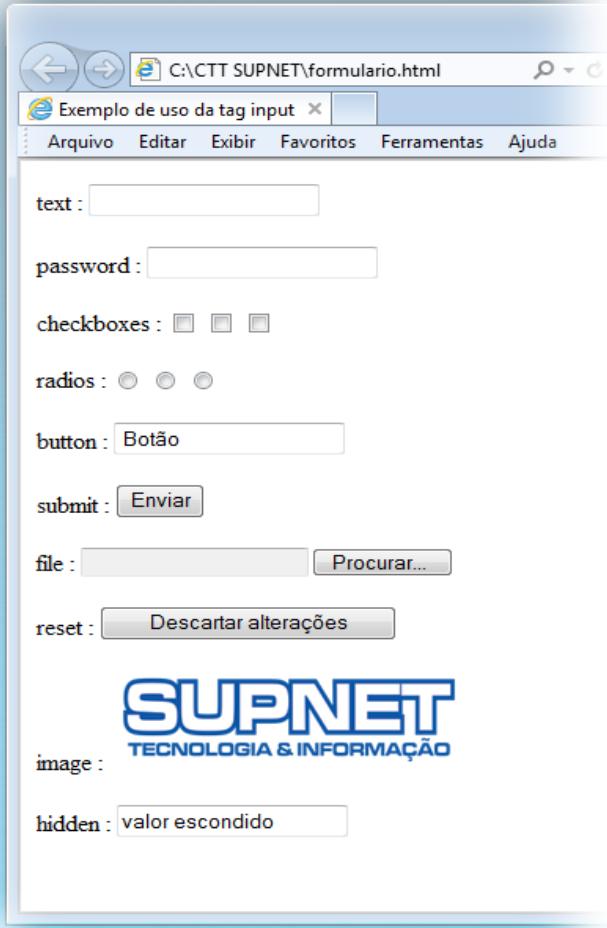


Figura 2.11: Exemplo de uso da tag input

#### A tag select

A tag select permite ao usuário escolher um ou mais itens de uma lista. O atributo multiple, quando presente, informa ao navegador que mais de um item pode ser selecionado.

A lista de itens deve ser informada através de elementos option. Tais elementos devem ser filhos diretos ou indiretos de elementos select. Além disso, cada item pode conter o atributo chamado value para informar o valor associado a uma determinada opção.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exemplo de uso da tag select</title>
</head>
<body>
<form action="pagina.html" method="get">
<p>
  Selecione uma cidade :
  <select>
    <option value="sao-paulo">São Paulo</option>
    <option value="rio-de-janeiro">Rio de Janeiro</option>
    <option value="porto-alegre">Porto Alegre</option>
    <option value="curitiba">Curitiba</option>
  </select>
</p>

<p>
  Selecione uma ou mais categorias de produtos ( mantenha a tecla "control" (ou "command" no Mac) pressionada para escolher mais de uma categoria ) :
  <select multiple="multiple">

    <option value="desktops">Desktops</option>
    <option value="notebooks">Notebooks</option>
    <option value="tablets">Tablets</option>
    <option value="celulares">Celulares</option>
  </select>
</p>
</form>
</body>
</html>
```

Código HTML 2.46: Exemplo de uso da tag select

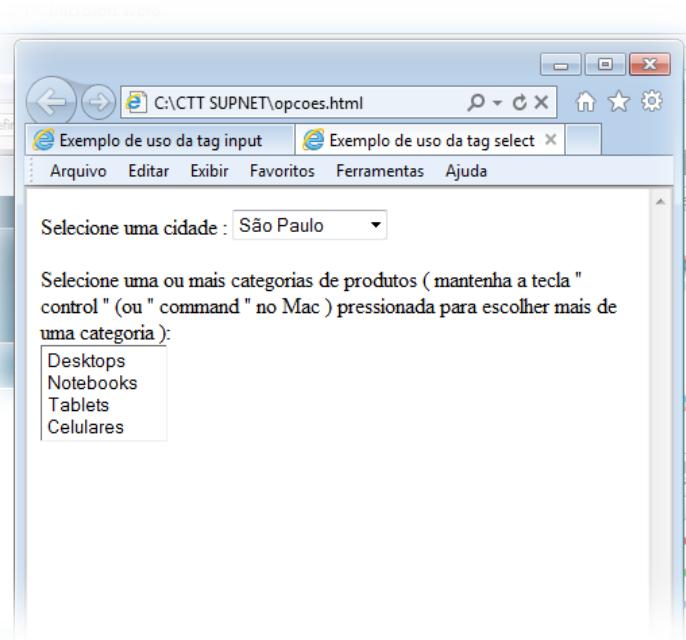


Figura 2.12: Exemplo de uso da tag select

Caso exista a necessidade de agrupar as opções de um elemento select, podemos utilizar o elemento optgroup em conjunto com o atributo label. Veja o exemplo:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exemplo de uso da tag select </title>
</head>
<body>
<form action="pagina.html" method="get">
<p>
  Selecione uma cidade :
  <select>
    <optgroup label="Região Sudeste">
      <option value="sao-paulo">São Paulo</option>
      <option value="rio-de-janeiro">Rio de Janeiro</option>
    </optgroup>
    <optgroup label="Região Sul">
      <option value="porto-alegre">Porto Alegre</option>
      <option value="curitiba">Curitiba</option>
    </optgroup>
  </select>
</p>
<p>
  Selecione uma ou mais categorias de produtos ( mantenha a tecla " control " (ou " command " no Mac ) pressionada para escolher mais de uma categoria ) :
  <select multiple="multiple">
```

```

<optgroup label ="De mesa ">
<option value =" desktops ">Desktops </ option >
</ optgroup >
<optgroup label =" Portáteis ">
<option value =" notebooks ">Notebooks </ option >
<option value =" tablets ">Tablets </ option >
<option value =" celulares ">Celulares </ option >
</ optgroup >
</ select >
</p>
</ form >
</ body >
</ html >

```

Código HTML 2.47: Exemplo de uso da tag select

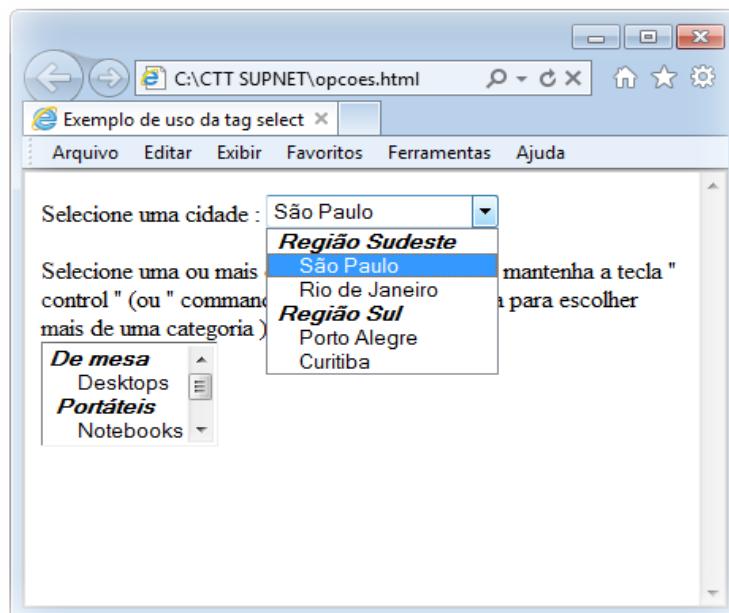


Figura 2.13: Exemplo de uso da tag select

#### A tag textarea

A tag textarea exibe uma caixa de texto na qual o usuário poderá inserir um texto qualquer. A diferença para a tag input com o atributo type com o valor text é que a tag textarea permite a inserção de textos mais longos e com quebras de linha.

```

<html >
<head >
<meta http - equiv =" Content - Type " content =" text / html ;
charset =UTF - 8">
<title>Exemplo de uso da tag textarea </title>
</head>
<body>
<form action ="pagina.html" method ="get">
<p>

```

```
textarea :  
<textarea>  
</textarea>  
</p>  
</form>  
</body>  
</html>
```

Código HTML 2.48: Exemplo de uso da tag textarea

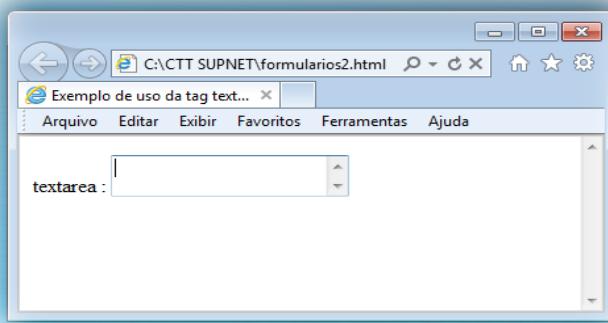


Figura 2.14: Exemplo de uso da tag textarea

### A tag label

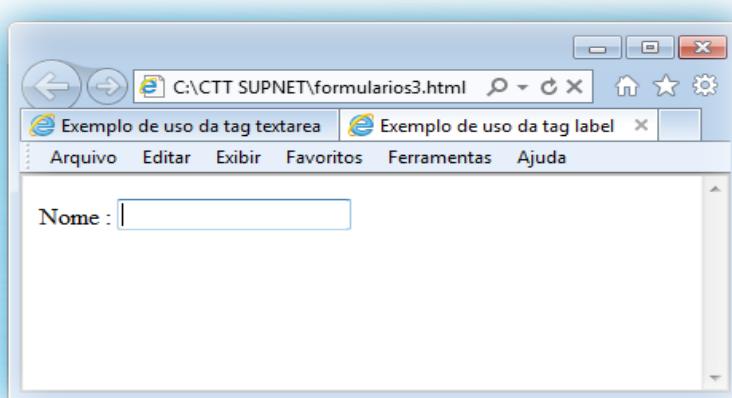
Em alguns dos exemplos anteriores foram inseridos textos ao lado dos elementos de formulário apresentados. Podemos pensar nesses textos como rótulos de cada elemento e é exatamente para esse fim que devemos utilizar a tag label do HTML.

Além de servir como rótulo, a tag label auxilia o usuário a interagir com os elementos do formulário. Utilizando o atributo for podemos fazer com que um elemento do formulário receba o foco.

Veja o exemplo:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exemplo de uso da tag label </title>
</head>
<body>
<form action="pagina.html" method="get">
<p>
<label for="nome">Nome :</label>
<input type="text" id="nome" />
</p>
</form>
</body>
</html>
```

*Código HTML 2.49: Exemplo de uso da tag label*



*Figura 2.15: Exemplo de uso da tag label*

Repare que o atributo for da tag label deve conter um valor igual ao do atributo id do elemento que desejamos focar.

### A tag form

Desde o momento em que começamos a falar sobre os elementos de formulário utilizamos a tag form, porém não falamos nada sobre ela. A tag form irá definir, de fato, o nosso formulário. Todos os elementos de formulário que vimos anteriormente devem ser filhos diretos ou indiretos de um elemento com a tag form para que os dados vinculados a esses elementos sejam enviados.

O papel do formulário é enviar os dados contidos nele para algum lugar, mas para onde? O atributo action é quem diz para onde os dados de um formulário deve ser enviado. Além disso, devemos informar a maneira como queremos que esses dados sejam enviados, ou seja, se queremos que eles sejam enviados através de uma requisição do tipo GET ou POST (métodos de envio definidos no protocolo HTTP).

## Exercícios de Fixação

**25** - Crie um documento HTML em um arquivo chamado **formulario.html** na pasta **html** com diversos elementos de formulário e para cada elemento crie um rótulo. Cada rótulo deve focar o elemento de formulário correspondente.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>A tag label e os elementos de formulário </title>
</head>
<body>
<form action="pagina.html" method="get">
<p>
<label for="nome">Nome :</label>
<input type="text" id="nome" />
</p>
<p>
<label for="senha">Senha :</label>
<input type="password" id="senha"/>
</p>
<p>
Sexo :
<input type="radio" name="sexo" id="masculino" />
<label for="masculino">Masculino </label>
<input type="radio" name="sexo" id="feminino" />
<label for="feminino">Feminino </label>
</p>
<p>
<label for="mensagem">Mensagem :</label>
<textarea id="mensagem"></textarea>
</p>
</form>
</body>
</html>
```

Código HTML 2.50: *formulario.html*

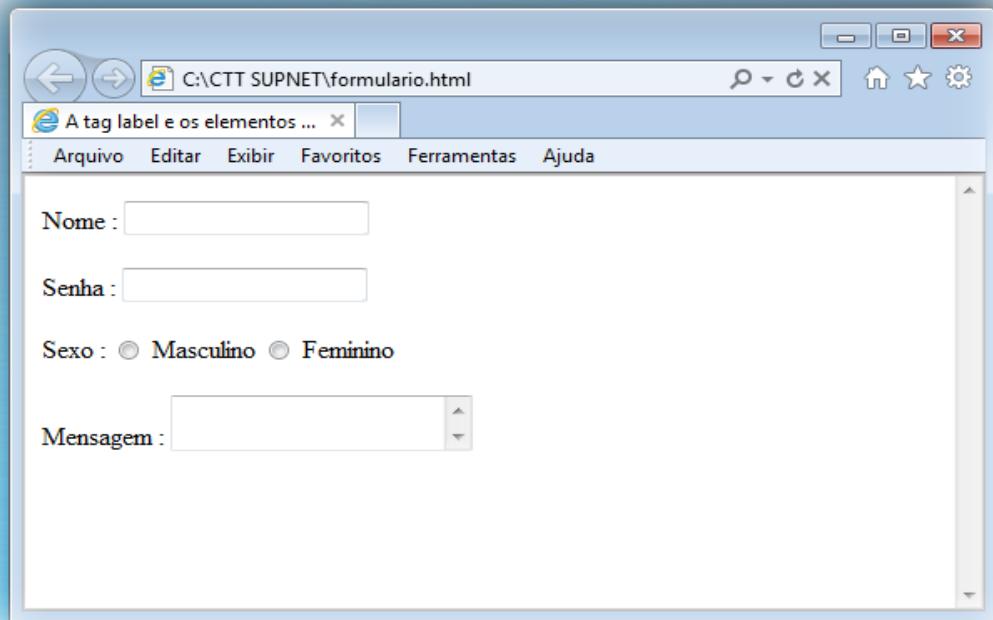


Figura 2.16: Formulário

## 💻 Exercícios Complementares

**26** – Em um documento HTML, crie um formulário que utilize as tags input, select, label e textarea. Tente utilizar todos os tipos do elemento input vistos neste capítulo.

# CSS

# 1. CSS

## O QUE É CSS?

O CSS formata a informação entregue pelo HTML. Essa informação pode ser qualquer coisa: imagem, texto, vídeo, áudio ou qualquer outro elemento criado. Grave isso: CSS formata a informação. Essa formatação na maioria das vezes é visual, mas não necessariamente. No CSS Aural, nós manipulamos o áudio entregue ao visitante pelo sistema de leitura de tela. Nós controlamos volume, profundidade, tipo da voz ou em qual das caixas de som a voz sairá. De certa forma você está formatando a informação que está em formato de áudio e que o visitante está consumindo ao entrar no site utilizando um dispositivo ou um sistema de leitura de tela. O CSS prepara essa informação para que ela seja consumida da melhor maneira possível.

O HTML5 trouxe poucas novidades para os desenvolvedores client-side. Basicamente foram criadas novas tags, o significado de algumas foi modificado e outras tags foram descontinuadas. As novidades interessantes mesmo ficaram para o pessoal que conhece Javascript. As APIs que o HTML5 disponibilizou são sem dúvida uma das features mais aguardadas por todos estes desenvolvedores. Diferentemente do CSS3 que trouxe mudanças drásticas para a manipulação visual dos elementos do HTML.

Com o CSS que nós conhecemos podemos formatar algumas características básicas: cores, background, características de font, margins, paddings, posição e controlamos de uma maneira muito artesanal e simples a estrutura do site com a propriedade float.

Você deve pensar: “mas com todas as características nós conseguimos fazer sites fantásticos, com design atraente e com a manutenção relativamente simples”. E eu concordo com você, mas outras características que nós não temos controles e dependemos de:

- 1) Algum programa visual como o Photoshop para criarmos detalhes de layout;
- 2) Javascript para tratarmos comportamentos ou manipularmos elementos específicos na estrutura do HTML;
- 3) Estrutura e controle dos elementos para melhorarmos acessibilidade e diversos aspectos do SEO;

Com as atualizações do CSS3 e com os browsers atualizando o suporte do CSS2.1, nós entramos em patamar onde sem dúvida o CSS é a arma mais poderosa para o designer web. Segue uma pequena lista dos principais pontos que podemos controlar nesse novo patamar:

- 1) selecionar primeiro e último elemento;
- 2) selecionar elementos pares ou ímpares;
- 3) selecionarmos elementos específicos de um determinado grupo de elementos;
- 4) gradiente em textos e elementos;
- 5) bordas arredondadas;
- 6) sombras em texto e elementos;
- 7) manipulação de opacidade;
- 8) controle de rotação;
- 9) controle de perspectiva;
- 10) animação;
- 11) estruturação independente da posição no código HTML;

É agora que começa diversão ao formatar sites com CSS.

## 2. SELETORES COMPLEXOS

A sintaxe do CSS é simples:

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

A propriedade é a característica que você deseja modificar no elemento. O valor é o valor referente a esta característica. Se você quer modificar a cor do texto, o valor é um Hexadecimal, RGBA ou até mesmo o nome da cor por extenso. Até aqui, nada diferente. Muitas vezes você não precisa aprender do que se trata a propriedade, basta saber que existe e se quiser decorar, decore. Propriedades são criadas todos os dias e não é um ato de heroísmo você saber todas as propriedades do CSS e seus respectivos valores.

Os seletores são a alma do CSS e você precisa dominá-los. É com os seletores que você irá escolher um determinado elemento dentro todos os outros elementos do site para formatá-lo. Boa parte da inteligência do CSS está em saber utilizar os seletores de uma maneira eficaz, escalável e inteligente.

## O que é um seletor?

O seletor representa uma estrutura. Essa estrutura é usada como uma condição para determinar quais elementos de um grupo de elementos será formatada.

Seletores encadeados e seletores agrupados são a base do CSS. Você os aprende por osmose durante o dia a dia. Para você lembrar o que são seletores encadeados e agrupados segue um exemplo abaixo:

Exemplo de seletor encadeado:

```
div p strong a {  
    color: red;  
}
```

Este seletor formata o link (a), que está dentro de um strong, que está dentro de P e que por sua vez está dentro de um DIV.

Exemplo de seletor agrupado:

```
strong, em, span {  
    color: red;  
}
```

Você agrupa elementos separados por vírgula para que herdem a mesma formatação.

Estes seletores são para cobrir suas necessidades básicas de formatação de elementos. Eles fazem o simples. O que vai fazer você trabalhar menos, escrever menos código CSS e também menos código Javascript são os seletores complexos.

Os seletores complexos selecionam elementos que talvez você precisaria fazer algum script em Javascript para poder marcá-lo com uma CLASS ou um ID para então você formatá-lo. Com os seletores complexos você consegue formatar elementos que antes eram inalcançáveis.

## Exemplo de funcionamento

Imagine que você tenha um título (h1) seguido de um parágrafo (p). Você precisa selecionar todos os parágrafos que vem depois de um título h1. Com os seletores complexos você fará assim:

```
h1 + p {  
    color: red;  
}
```

Esses seletores é um dos mais simples e mais úteis que eu já utilizei em projetos por aí. Não precisei adicionar uma classe com JQuery, não precisei de manipular o CMS para marcar esse parágrafo, não precisei de nada. Apenas apliquei o seletor.

Abaixo, veja uma lista de seletores complexos e quais as suas funções. Não colocarei todas as possibilidades aqui porque pode haver modificações, novos formatos ou outros formatos que podem ser descontinuados. Para ter uma lista sempre atualizada, siga o link no final da tabela.

PADRÃO	SIGNIFICADO	CSS
<b>elemento[attr]</b>	Elemento com um atributo específico.	2
<b>elemento[attr="x"]</b>	Elemento que tenha um atributo com um valor específico igual a "x".	2
<b>elemento[attr~="x"]</b>	Elemento com um atributo cujo valor é uma lista separada por espaços, sendo que um dos valores é "x".	2
<b>elemento[attr^="x"]</b>	Elemento com um atributo cujo valor comece exatamente com string "x".	3
<b>elemento[attr\$="x"]</b>	Elemento com um atributo cujo valor termina exatamente com string "x".	3
<b>elemento[attr*="x"]</b>	Elemento com um atributo cujo valor contenha a string "x".	3
<b>elemento[attr ="en"]</b>	Um elemento que tem o atributo específico com o valor que é separado por hífen começando com EN (da esquerda para direita).	2
<b>elemento:root</b>	Elemento root do documento. Normalmente o HTML.	3
<b>elemento:nth-child(n)</b>	Seleciona um objeto N de um determinado elemento.	3
<b>elemento:nth-last-child(n)</b>	Seleciona um objeto N começando pelo último objeto do elemento.	3
<b>elemento:empty</b>	Seleciona um elemento vazio, sem filhos. Incluindo elementos de texto.	3
<b>elemento:enabled</b> <b>elemento:disabled</b>	Seleciona um elemento de interface que esteja habilitado ou desabilitado, como selects, checkbox, radio button etc.	3
<b>elemento:checked</b>	Seleciona elementos que estão checados, como radio buttons e checkboxes.	3
<b>E &gt; F</b>	Seleciona os elementos E que são filhos diretos de F.	2
<b>E + F</b>	Seleciona um elemento F que precede imediatamente o elemento E.	2

Lista atualizada pelo W3C <http://www.w3.org/TR/css3-selectors/#selectors>

### 3. GRADIENTE

Uma das features mais interessantes é a criação de gradientes apenas utilizando CSS. Todos os browsers mais novos como Safari, Opera, Firefox e Chrome já aceitam essa feature e você pode utilizá-la hoje. Os Internet Explorer atuais (8 e 9) não reconhecem ainda, contudo você poderá utilizar imagens para estes browsers que não aceitam essa feature. Você pode perguntar: “Mas já que terei o trabalho de produzir a imagem do gradiente, porque não utilizar imagens para todos os browsers?” Lembre-se que se utilizar uma imagem, o browser fará uma requisição no servidor buscando essa imagem, sem imagem, teremos uma requisição a menos, logo o site fica um pouquinho mais rápido. Multiplique isso para todas as imagens de gradiente que você fizer e tudo realmente fará mais sentido.

Deixe para que os browsers não adeptos baixem imagens e façam mais requisições.

Veja abaixo um exemplo de código, juntamente com o fallback de imagem:

```
div {  
width:200px;  
height:200px;  
background-color: #FFF;  
/* imagem caso o browser não aceite a feature */  
background-image: url(images/gradiente.png);  
/* Firefox 3.6+ */  
background-image: -moz-linear-gradient(green, red);  
/* Safari 5.1+, Chrome 10+ */  
background-image: -webkit-linear-gradient(green, red);  
/* Opera 11.10+ */  
background-image: -o-linear-gradient(green, red);  
}
```

**Atenção:** Até que os browsers implementem de vez essa feature, iremos utilizar seus prefixos.

Como ficou:



## "Stops" ou definindo o tamanho do seu gradiente

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. O código de exemplo segue abaixo:

```
/* Firefox 3.6+ */
background-image: -moz-linear-gradient(green, red 20%);
/* Safari 5.1+, Chrome 10+ */
background-image: -webkit-linear-gradient(green, red 20%);
/* Opera 11.10+ */
background-image: -o-linear-gradient(green, red 20%);
```

Veja o resultado:



Se colocarmos um valor para o verde, nós iremos conseguir efeitos que antes só conseguíramos no Illustrator ou no Photoshop. Segue o código e o resultado logo após:

```
/* Firefox 3.6+ */
background-image: -moz-linear-gradient(green 10%, red 20%);
/* Safari 5.1+, Chrome 10+ */
background-image: -webkit-linear-gradient(green 10%, red 20%);
/* Opera 11.10+ */
background-image: -o-linear-gradient(green 10%, red 20%);
```



Perceba que o tamanho da transição vai ficando menor à medida que vamos aumentando as porcentagens. Muito, mas muito útil.

## 4. COLUMNS

Com o controle de colunas no CSS, podemos definir colunas de texto de forma automática. Até hoje não havia maneira de fazer isso de maneira inteligente com CSS e o grupo de propriedades columns pode fazer isso de maneira livre de gambiarras.

### **column-count**

A propriedade column-count define a quantidade de colunas terá o bloco de textos.

```
/* Define a quantidade de colunas, a largura é definida
uniformemente. */
-moz-column-count: 2;
-webkit-column-count: 2;
```

### **column-width**

Com a propriedade column-width definimos a largura destas colunas.

```
/* Define qual a largura mínima para as colunas. Se as colunas
forem espremidas, fazendo com que a largura delas fique menor
que este valor, elas se transformam em 1 coluna
automaticamente */
-moz-column-width: 400px;
-webkit-column-width: 400px;
```

Fiz alguns testes aqui e há uma diferença entre o Firefox 3.5 e o Safari 4 (Public Beta).

O column-width define a largura mínima das colunas. Na documentação do W3C é a seguinte: imagine que você tenha uma área disponível para as colunas de 100px. Ou seja, você tem um div com 100px de largura (width). E você define que as larguras destas colunas (column-width) sejam de 45px. Logo, haverá 10px de sobra, e as colunas irão automaticamente ter 50px de largura, preenchendo este espaço disponível. É esse o comportamento que o Firefox adota.

Já no Safari, acontece o seguinte: se você define um column-width, as colunas obedecem a esse valor e não preenchem o espaço disponível, como acontece na explicação do W3C e como acontece também no Firefox. Dessa forma faz mais sentido para mim.

Como a propriedade não está 100% aprovada ainda, há tempo para que isso seja modificado novamente. Talvez a mudança da nomenclatura da classe para column-min-width ou algo parecido venha a calhar, assim, ficamos com os dois resultados citados, que são bem úteis para nós de qualquer maneira.

## column-gap

A propriedade column-gap cria um espaço entre as colunas, um gap.

```
/* Define o espaço entre as colunas. */
-moz-column-gap: 50px;
-webkit-column-gap: 50px;
```

Utilizamos aqui os prefixos -moz- e -webkit-, estas propriedades não funcionam oficialmente em nenhum browser. Mas já podem ser usados em browsers como Firefox e Safari.

## 5. TRANSFORM 2D

A propriedade transform manipula a forma com que o elemento aparecerá na tela. Você poderá manipular sua perspectiva, escala e ângulos. Uma transformação é especificada utilizando a propriedade transform. Os browsers que suportam essa propriedade, a suportam com o prefixo especificado.

Os valores possíveis até agora estão especificados abaixo:

### scale

O valor scale modificará a dimensão do elemento. Ele aumentará proporcionalmente o tamanho do elemento levando em consideração o tamanho original do elemento.

### skew

Skew modificará os ângulos dos elementos. Você pode modificar os ângulos individualmente dos eixos X e Y como no código abaixo:

```
-webkit-transform: skewY(30deg);
-webkit-transform: skewX(30deg);
```

### translation

O translation moverá o elemento no eixo X e Y. O interessante é que você não precisa se preocupar com floats, positions, margins e etc. Se você aplica o translation, ele moverá o objeto e pronto.

### rotate

O rotate rotaciona o elemento levando em consideração seu ângulo, especialmente quando o ângulo é personalizado com o transform-origin.

## CSS Transform na prática

Veja o código abaixo e seu respectivo resultado:

```
img {
    -webkit-transform: skew(30deg); /* para webkit */
    -moz-transform: skew(30deg); /* para gecko */
    -o-transform: skew(30deg); /* para opera */
    transform: skew(30deg); /* para browsers sem prefixo */
```

O código acima determina que o ângulo da imagem seja de 30deg. Colocamos um exemplo para cada prefixo de browser. Ficando assim:



### **Várias transformações em um único elemento**

Para utilizarmos vários valores ao mesmo tempo em um mesmo elemento, basta definir vários valores separando-os com espaços em uma mesma propriedade transform:

```
img {
    -webkit-transform: scale(1.5) skew(30deg); /* para webkit */
    -moz-transform: scale(1.5) skew(30deg); /* para gecko */
    -o-transform: scale(1.5) skew(30deg); /* para opera */
    transform: scale(1.5) skew(30deg); /* para browsers sem prefixo */
}
```

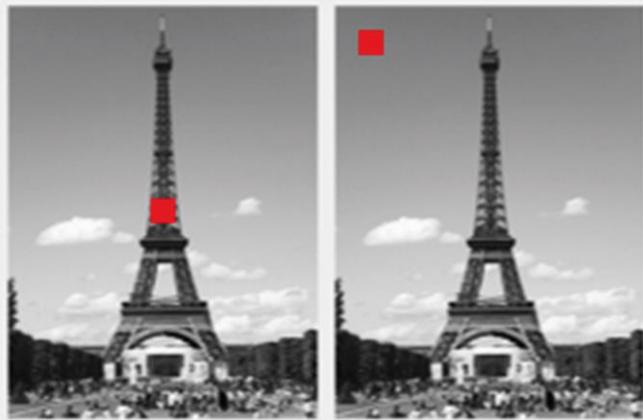
### **transform-origin**

A propriedade transform-origin define qual o ponto do elemento a transformação terá origem. A sintaxe é idêntica ao background-position. Observe o código abaixo:

```
img {
    -webkit-transform-origin: 10px 10px; /* para webkit */
    -moz-transform-origin: 10px 10px; /* para webkit */
    -o-transform-origin: 10px 10px; /* para webkit */
    transform-origin: 10px 10px; /* para webkit */
}
```

Como padrão as transições sempre acontecem tendo como ponto de âncora o centro do objeto.

Há algumas situações que pode ser que você queira modificar essa âncora, fazendo com que, por exemplo, a rotação aconteça em algum dos cantos do elemento. O código de exemplo acima fará com que a transformação aconteça a partir dos 10px do topo no canto esquerdo. Veja a comparação entre o padrão e o resultado do código acima.



A propriedade transform fica mais interessante quando a utilizamos com a propriedade transition, onde podemos executar algumas animações básicas manipulando os valores de transformação.

## 6. TRANSIÇÕES E ANIMAÇÕES

Durante muito tempo o CSS só serviu para pintar quadradinhos e mais nada. Desde quando o pessoal do WaSP organizou todo o movimento dos Padrões Web fazendo com que todos os desenvolvedores, fabricantes de browsers e até mesmo o W3C acreditassesem no poder dos padrões não houve grandes atualizações no CSS. Praticamente formatávamos font, background, cor, tamanhos e medidas de distância e posição.

A propriedade transition, a regra keyframe e outras propriedades vieram vitaminar a função que o CSS tem perante o HTML acrescentando maneiras de produzirmos animações e transições. Não estou dizendo que você fará animações complicadas, com diversos detalhes técnicos e etc. Para esse tipo de resultado existem outras ferramentas, incluindo Canvas e SVG, que com certeza farão um trabalho melhor com menos esforço. Mas é fato que as animações viajam CSS nos ajudará a levar a experiência do usuário para outro patamar.

Lembrando que o nível de suporte de algumas dessas técnicas ainda é muito baixo. A propriedade transition funciona em boa parte dos browsers atuais. Mas a regra keyframe que nos permite controlar as fases de uma animação ainda é muito restrita. Para uma tabela mais atual e detalhada de suporte e compatibilidade, faça uma procura no Google. Onde for possível demonstrarei o código com o prefixo dos browsers que suportam as propriedades.

### O básico: propriedade transition

A propriedade transition é praticamente autoexplicativa. Sua sintaxe tão simples que talvez até dispense explicações mais elaboradas. Vamos começar com o código abaixo:

```
a {  
color: white;  
background: gray;  
}
```

No código definimos que o link terá sua cor de texto igual a preta e seu background será cinza.

O resultado esperado é que ao passar o mouse no link a cor do texto seja modificada, mudando do branco para o preto e que a cor de background mude de cinza para vermelho. O código abaixo faz exatamente isso:

```
a {  
color: white;  
background: gray;  
}  
a:hover {  
color: black;  
background: red;  
}
```

O problema é que a transição é muito brusca. O browser apenas modifica as características entre os dois blocos e pronto. Não há nenhuma transição suave entre os dois estados.

Podemos fazer a mudança de um estado para outro utilizando a propriedade transition. Suponha que ao passar o mouse, as mudanças acontecessem em um intervalo de meio segundo. Bastaria colocar a propriedade transition no a:hover e pronto. Ao passar o mouse, o browser modificaria as características do link com uma pequena transição de meio segundo. O código seria como se segue abaixo:

```
a:hover {  
    color: black;  
    background: red;  
    -webkit-transition: 0.5s;  
}
```

Dessa forma a transição apenas acontece quando o hover é ativado. O problema é que ao tirar o mouse, o browser volta bruscamente para as características iniciais. Para modificar isso basta inserir também a propriedade transition no estado inicial.

```
a {  
    color: white;  
    background: gray;  
    -webkit-transition: 0.5s;  
}  
a:hover {  
    color: black;  
    background: red;  
    -webkit-transition: 0.5s;  
}
```

O que a propriedade transition faz é comparar os valores das propriedades em comum entre os dois estados do link ou de qualquer outro elemento, assim ela modifica suavemente os valores quando há a ativação da função. Esta é uma técnica simples e que serve para manipularmos transições básicas como cor, tamanho, posição etc.

Agora suponha que em um bloco há uma determinada propriedade que no outro bloco não há, como no código abaixo:

```
a {  
    border: 1px solid orange;  
    color: white;  
    background: gray;  
    -webkit-transition: 0.5s;  
}  
a:hover {  
    color: black;  
    background: red;  
    -webkit-transition: 0.5s;  
}
```

Nesse caso o browser detecta que há uma propriedade no primeiro estado, mas não no segundo, por isso ele não faz a transição desta propriedade, apenas das propriedades em comuns.

Abaixo veja o código. copie em um arquivo HTML e veja o efeito:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="utf-8">
<title>CSS Transition</title>
<style type="text/css" media="screen">
a {
color:white;
background:gray;
-webkit-transition: 0.5s linear;
}
a:hover {
color:black;
background:red;
-webkit-transition: 0.5s linear;
}
</style>
</head>
<body>
<a href="#">Link! Hello World!</a>
</body>
</html>
```

## **Propriedade animation e regra keyframe**

A propriedade transition trabalha de forma muito simples e inflexível. Você praticamente diz para o browser qual o valor inicial e o valor final para que ele aplique a transição automaticamente, controlamos praticamente apenas o tempo de execução. Para termos mais controle sobre a animação temos a propriedade animation que trabalha juntamente com a rule keyframe.

Basicamente você consegue controlar as características do objeto e suas diversas transformações definindo fases da animação. Observe o código abaixo e veja seu funcionamento:

```
@-webkit-keyframes rodar {
from {
-webkit-transform:rotate(0deg);
}
to {
-webkit-transform:rotate(360deg);
}
}
```

O código acima define um valor inicial e um valor final. Agora vamos aplicar esse código a um elemento. Minha ideia é fazer um DIV girar. ;-)

O código HTML até agora é este. Fiz apenas um div e defini este keyframe:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<title></title>
<meta charset="utf-8">
<style>
@-webkit-keyframes rodaroda {
from {
-webkit-transform:rotate(0deg) ;
}
to {
-webkit-transform:rotate(360deg) ;
}
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

Primeiro você define a função de animação, no caso é o nosso código que define um valor inicial de 0 graus e um valor final de 360 graus. Agora vamos definir as características deste DIV.

```
div {
width:50px;
height:50px;
margin:30% auto 0;
background:black;
}
```

Nas primeiras linhas defini qual será o estilo do div. Ele terá uma largura e uma altura de 50px. A margin de 30% do topo garantirá um espaço entre o objeto e o topo do documento, e background preto.

A propriedade animation tem uma série de propriedades que podem ser resumidas em um shortcode bem simples. Veja a tabela logo a seguir para entender o que cada propriedade significa:

Propriedade	Definição
<b>animation-name</b>	Especificamos o nome da função de animação
<b>animation-duration</b>	Define a duração da animação. O valor é declarado em segundos.
<b>animation-timing-function</b>	Descreve qual será a progressão da animação a cada ciclo de duração. Existem uma série de valores possíveis e que pode ser que o seu navegador ainda não suporte, mas são eles: ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>) [, ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>)]*
	O valor padrão é ease.
<b>animation-interation-count</b>	Define o número de vezes que o ciclo deve acontecer. O padrão é um, ou seja, a animação acontece uma vez e pára. Pode ser também infinito definindo o valor infinite no valor.
<b>animation-direction</b>	Define se a animação irá acontecer ou não no sentido inverso em ciclos alternados. Ou seja, se a animação está acontecendo no sentido horário, ao acabar a animação, o browser faz a mesma animação no elemento, mas no sentido anti-horário. Os valores são alternate ou normal.
<b>animation-play-state</b>	Define se a animação está acontecendo ou se está pausada. Você poderá por exemplo, via script, pausar a animação se ela estiver acontecendo. Os valores são running ou paused.
<b>animation-delay</b>	Define quando a animação irá começar. Ou seja, você define um tempo para que a animação inicie. O valor 0, significa que a animação começará imediatamente.

Voltando para o nosso código, vamos aplicar algumas dessas propriedades:

```
div {
    width:50px;
    height:50px;
    margin:30% auto 0;
    background:black;

    -webkit-animation-name: rodaroda;
    -webkit-animation-duration: 0.5s;
    -webkit-animation-timing-function: linear;
    -webkit-animation-iteration-count: infinite;
    -webkit-animation-direction: alternate;
}
```

Veja que na propriedade animation-name chamamos o mesmo nome que demos na nossa função de keyframe logo no começo da explicação. Depois definimos uma duração de ciclo de meio segundo. Definimos que o comportamento da animação será linear, e com a propriedade animation-iteration-count definimos que ele girará infinitamente. E por último definimos pelo animation-direction que a animação deverá ser alternada, ou seja, o DIV girará para um lado, e quando alcançar o final da animação, o browser deverá alternar essa animação.

Podemos melhorar esse código utilizando a versão shortcode, que é mais recomendado. Veja a ordem que devemos escrever as propriedades animation em forma de shortcode:

```
animation: [<animation-name> || <animation-duration> || <animation-timing-function> || <animation-delay> || <animation-iteration-count> || <animation-direction>] [, [<animation-name> || <animation-duration> || <animation-timing-function> || <animation-delay> || <animation-iteration-count> || <animation-direction>] ]*
```

Aplicando isso ao nosso código:

```
div {
    width:50px;
    height:50px;
    margin:30% auto 0;
    background:black;

    -webkit-animation: rodaroda 0.5s linear infinite alternate;
```

Pronto. Agora temos um elemento que gira sem parar, hora para direita hora para esquerda.

## Definindo ciclos

Nós definimos no keyframe do nosso último exemplo apenas um início e um fim. Mas e se quiséssemos que ao chegar na metade da animação o nosso elemento ficasse com o background vermelho? Ou que ele mudasse de tamanho, posição e etc.? É aí onde podemos flexibilizar melhor nosso keyframe definindo as fases da animação. Por exemplo, podemos dizer para o elemento ter uma cor de background diferente quando a animação chegar aos 10% do ciclo, e assim por diante.

Veja o exemplo:

```
@-webkit-keyframes rodaroda {
0% {
-webkit-transform:rotate(0deg);
}
50% {
background:red;
-webkit-transform:rotate(180deg);
}
100% {
-webkit-transform:rotate(360deg);
}
}
```

Definimos acima que o início da animação o elemento começará na posição normal, 0 graus.

Quando a animação chegar aos 50% do ciclo, o elemento deverá ter girado para 180 graus e o background dele deve ser vermelho. E quando a animação chegar a 100% o elemento deve ter girado ao todo 360 graus e o background, como não está sendo definido, volta ao valor padrão, no nosso caso black, que foi definido no CSS onde formatamos este DIV.

Logo nosso elemento girará pra sempre e ficará alternando a cor de fundo de preto para vermelho. Fiz um exemplo bem simples modificando apenas o background, mas você pode muito bem definir um position e modificar os valores de left e top para fazer o elemento se movimentar.

No exemplo também defini apenas 3 estágios (0%, 50% e 100%) você pode definir um maior número de estágios: 5%, 10%, 12%, 16% e etc... Adequando as fases da animação às suas necessidades.

Há exemplos muito interessantes na internet onde podemos ver todo o poder das animações feitas pela CSS. Veja o exemplo que fizemos aqui neste texto no endereço:  
<http://migre.me/4ubym>

## 7. BORDAS

Definir imagem para as bordas é uma daquelas propriedades da CSS que você se pergunta como vivíamos antes de conhecê-la. É muito mais fácil entender testando na prática, por isso sugiro que se você estiver perto de um computador, faça testes enquanto lê este texto. A explicação pode não ser suficiente em algumas partes, mas a prática irá ajudá-lo a entender.

Esta propriedade ainda está em fase de testes pelos browsers, por isso utilizaremos os prefixos para ver os resultados. Utilizarei apenas o prefixo do Safari, mas o Firefox já entende essa propriedade muito bem.

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.

Segue um exemplo da sintaxe abaixo:

```
a {
display:block;
width:100px;
-webkit-border-image: url(border.gif) 10 10 10 10 stretch;
}
```

Acima definimos uma imagem com o nome de border.gif, logo depois definimos o width de cada uma das bordas do elemento. A sintaxe é igual a outras propriedades com 4 valores: top, right, bottom, left. E logo depois colocamos qual o tipo de tratamento que a imagem vai receber.

### *Dividindo a imagem*

Para posicionar a imagem devidamente em seu objeto o browser divide a imagem em 9 seções:



Quando a imagem é colocada no elemento, o browser posiciona os cantos da imagem juntamente com os cantos correspondentes do elemento. Ou seja, o bloco 1 da imagem é colocado no canto superior esquerdo, o 3 no canto superior direito e assim por diante. Se você fizer o teste, a imagem aparecerá no elemento como se estivesse desproporcional. Isso é normal porque a imagem deve seguir as proporções do elemento e não as suas próprias.

## Comportamento da imagem

O comportamento da imagem é a parte mais importante porque define como o centro da imagem (no caso do nosso exemplo a seção de número 5), irá ser tratado. Há vários valores, mas que é mais simples de se entender é a stretch, que estica e escala a imagem para o tamanho do elemento aplicado. Há outros valores como round e repeat. Mas hoje alguns browsers não tem tanto suporte e acabam ou ignorando esses valores ou como no caso do Safari e o Chrome, interpretam o round como o repeat. Vamos nos concentrar com o stretch e você entenderá como funciona a propriedade.

## Aplicação

Vamos utilizar a imagem abaixo para aplicar a propriedade. Iremos fazer um botão ao estilo iPhone. A coisa é simples e sugiro que você faça testes individualmente brincando com os valores das bordas e com diversas outras imagens para ver como funciona o recurso.



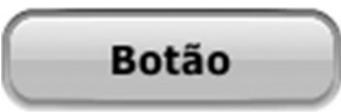
Irei aplicar o estilo em um link. O código que irei aplicar segue abaixo:

```
{
display:block;
width:100px;
text-align:center;
font:bold 13px verdana, arial, tahoma, sans-serif;
text-decoration:none;
color:black;
}
```

Para inserir a imagem, colocamos as duas linhas abaixo:

```
border-width:10px;  
-webkit-border-image: url(button.png) 10 stretch;
```

Defini com a primeira linha que a borda teria 10px de largura com a propriedade border-width. Na segunda linha define que a imagem utilizada seria a button.png, que as áreas da imagem teriam que se estender por 10px, e o valor stretch define que a imagem cobrirá o elemento inteiro, o resultado segue abaixo:



**Botão**

Para você ver como tudo ali é meio estranho. Se eu diminuir o valor de 10 do border-image, que é o valor que define quanto a imagem deve se estender nas bordas, veja como fica:



**Botão**

Temos o border-width definindo a largura da borda, mas não temos nenhum valor dizendo quanto dessa largura a imagem deve tomar.

Os efeitos são bem estranhos quando esses valores estão mal formatados. Por isso, teste na prática essa propriedade para que não haja problemas a programar em seus projetos. O pulo da gato, para mim, é a propriedade border-width.

## 8. MÚLTIPLOS BACKGROUNDS

Quem nunca teve que criar um background onde havia um gradiente em cima, embaixo e dos lados? Você sabe que para criar algo parecido você não pode utilizar uma imagem só. A solução até hoje seria criar 4 elementos divs aninhados (ou seja, um dentro do outro) e aplicar um pedaço deste background em cada um destes elementos para dar a sensação de um background único.

O resultado é interessante mas o meio que isso é feito não é nada bonito. Você era obrigado a declarar 4 elementos “inúteis” no seu HTML apenas para compensar um efeito visual. A possibilidade de atribuirmos múltiplos backgrounds em apenas um elemento é a feature que vai ajudá-lo a não sujar seu código.

A sintaxe para múltiplos backgrounds é praticamente idêntica a sintaxe para definir um background. Segue abaixo um exemplo:

```
div {  
width:600px;  
height:500px;  
background:  
url(img1.png) top left repeat-x,  
url(img2.png) bottom left repeat-y;  
}
```

Definimos apenas uma propriedade background, o valor dessa propriedade em vez de conter apenas um valor como normalmente fazemos, poderá haver vários, com suas respectivas definições de posição, repeat e attachment (fixed).

## 9. MÓDULO TEMPLATE LAYOUT

Talvez você me chame de louco, mas para mim a parte mais fácil de desenvolver um site com CSS é o planejamento e diagramação do layout. Coincidemente é a parte que mais os desenvolvedores tem problemas crossbrowser ou por falta de recursos mais avançados. Mas se você parar para pensar, apenas uma propriedade cuida dessa parte, que é a propriedade float. De longe, para mim, o float é a propriedade mais importante que há no CSS. Se o IE não soubesse o que é float, até hoje nós não estaríamos fazendo sites com CSS. O float cuida de toda a diagramação do site, desde os elementos que definirão as áreas mestres do site até os pequenos detalhes de imagens e ícones.

A propriedade float é muito simples de se entender. O problema não é o funcionamento, mas os efeitos que a propriedade float causa nos elementos próximos. Se você pede para duas colunas ficarem flutuando à esquerda e outra coluna à direita, o rodapé sobe. Ou se você coloca um elemento envolvendo outros elementos com float, esse elemento perde a altura. Estes são problemas corriqueiros que já tem soluções inteligentes e que não apresentam chateações mais graves.

Infelizmente o float não é o ideal para a diagramação e organização dos elementos do layout. Ele resolve muitos problemas, mas deixa a desejar em diversos sentidos. O float está completamente ligado a ordem dos elementos no HTML. Existem técnicas que você consegue fazer quase que qualquer organização visual sem encostar no código HTML. Mas há outras necessidades que invariavelmente você precisará modificar a ordem dos elementos no meio do HTML para que a diagramação do site saia conforme o esperado. Essa organização do HTML pode alterar desde programação server-side e até resultados de SEO e acessibilidade. Por isso é interessante que o HTML fique organizado de forma que ele supre as necessidades dessas bases. Sua organização visual deve ser independente desta organização.

Tendo em vista estes e outros problemas o W3C criou um novo módulo. Na verdade ele não é o único, e nem pode ser para que tenhamos diversas formas de trabalhar. O módulo em questão é chamado de Template Layout. Esse módulo consiste em uma forma de criarmos e organizarmos os elementos e informações do layout de forma menos espartana e mais flexível.

Podemos dividir a construção do layout em duas grandes partes: 1) Definição dos elementos mestres e grid a ser seguido. 2) Formatação de font, cores, background, bordas etc.

As propriedades nesta especificação trabalham associando uma política de layout de um elemento. Essa política é chamada de template-based positioning (não tem nada a ver com a propriedade position, pelo contrário é uma alternativa) que dá ao elemento uma grid invisível para alinhar seus elementos descendentes.

Porque o layout deve se adaptar em diferentes janelas e tamanhos de papéis, as colunas e linhas do grid deve ser fixas ou flexíveis dependendo do tamanho.

O W3C mostra alguns casos comuns:

- Páginas complexas com múltiplas barras de navegação em áreas com posições fixas como publicidade, submenus e etc.
- Formulários complexos, onde o alinhamento de labels e campos podem ser facilmente definidos com as propriedades deste módulo com a ajuda das propriedades de margin, padding e tables.
- GUIs, onde botões, toolbars, labels, ícones etc, tem alinhamentos complexos e precisam estar sempre alinhados caso o tamanho ou a resolução da tela mudam.
- Medias que são paginadas, como medias de impressão onde cada página são divididos em áreas fixas para conteúdos de gêneros diferentes.

Template-based positioning são uma alternativa para a propriedade position com o valor absolute. Antigamente lembro-me que quase todos os desenvolvedores tentavam organizar e diagramar layouts utilizando position. Não que seja errado, mas definitivamente não é a melhor forma. Costumo dizer em meus cursos e palestras que position é para detalhes. Nada muito grande, mas para pequenos detalhes. Usamos position para posicionarmos elementos que não tem relação direta com sua posição no código HTML. Ou seja, não importa onde o elemento esteja, o position:absolute; irá posicionar o elemento nas coordenadas que você quiser.

## Sintaxe e funcionamento

O módulo Template Layout basicamente define slots de layout para que você encaixe e posicione seus elementos. O mapeamento dos slots é feito com duas propriedades que já conhecemos que este módulo adiciona mais alguns valores e funcionalidades, são as propriedades position e display.

A propriedade display irá definir como será o Grid. Quantos slots e etc.

A propriedade position irá posicionar seus elementos nestes slots.

Veja o código HTML abaixo:

```
<div class="geral">
<nav class="menu">...</nav>
<aside class="menulateral">...</aside>
<aside class="publicidade">...</aside>
<article class="post">...</article>
<footer>...</footer>
</div>
```

Agora iremos definir a posição destes elementos. O código CSS seria assim:

```
.geral {
display: "aaa"
"bcd"
"eee";
}
nav.menu {position:a;}
aside.menulateral {position:b;}
```

```
aside.publicidade {position:d;}
article.post {position:c;}
footer {position:e;}
```

## O funcionamento da propriedade display

A propriedade display define a organização dos slots. Um slot é um local onde o seu elemento ficará. Cada elemento fica em um slot.

Aqui o elemento display trabalha como um table, onde seu conteúdo será colocando em colunas e linhas. A única diferença é que o número de linhas e colunas não dependem do conteúdo é fixa pelo valor da propriedade. E a outra principal diferença é que a ordem dos descendentes no código não importa.

Existem alguns valores para que você trabalhe: letra, @ (arroba) e “.” (pronto).

Cada letra diferente é um slot de conteúdo diferente. O @ define um sinal para um slot padrão. E o “.” (ponto) define um espaço em branco.

Quando repetimos as letras como no exemplo anterior, tanto na horizontal quanto na vertical, é formado um slot único que se expande para o tamanho da quantidade de slots. Lembra-se do colspan ou rowspan utilizados na tabela? Pois é, funciona igualzinho.

Não é possível fazer um slot que não seja retangular ou vários slots com a mesma letra. Um template sem letra nenhuma também não é possível. Um template com mais de um @ também é proibido. Se houverem esses erros a declaração é ignorada pelo browser.

Pra definir a altura da linha (row-height) podemos utilizar o valor padrão “auto”, que define altura que a altura da linha é feito de acordo com a quantidade de conteúdo no slot. Você pode definir um valor fixo para a altura. Não é possível definir um valor negativo. Quando definimos um \* (asterisco) para a altura, isso quer dizer que todas as alturas de linha serão iguais.

A largura da coluna (col-width) é definida com valores fixos, como o row-height. Podemos definir também o valor de \* que funciona exatamente igual ao altura de linha, mas aplicados a largura da coluna. Há dois valores chamados max-content e min-content que fazem com que a largura seja determinada de acordo com o conteúdo.

Outro valor é o minmax(p,q) que funciona assim: a largura das colunas são fixadas para ser maiores ou iguais a p e menores ou iguais a q. Pode haver um espaço branco (white space) em volta de p e q. Se q > p, então q é ignorado e o minmax(p,q) é tratado como minmax(p,p). O valor fit-content é o equivalente a minmax(min-content, max-content).

## Definindo a largura e altura dos slots

Para definir a altura dos slots, utilizamos uma sintaxe diretamente na propriedade display. Veja abaixo um exemplo de como podemos fazer isso:

```
display: "a a a" /150px
```

```
"b c d"
"e e e" / 150px
100px 400px 100px;
```

Formatando de uma maneira que você entenda, fica assim:

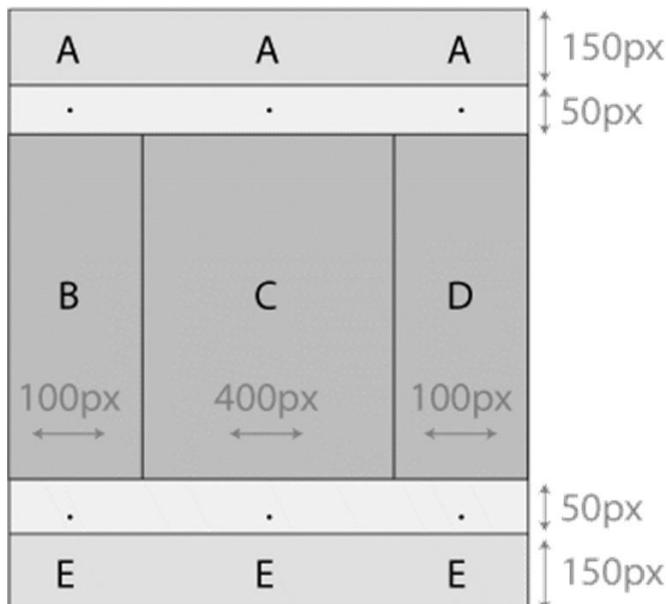
```
display: "a a a" /150px
"b c d"
"e e e" /150px
100px 400px 100px;
```

Ou seja, a primeira coluna do grid terá 100px de largura, a segunda 400px e a terceira 100px.

As medidas que coloquei ao lado, iniciando com uma / (barra) definem a altura das linhas. Logo a primeira linha terá 150px e a terceira linha também. A linha do meio, como não tem altura definida terá a altura de acordo com a quantidade de conteúdo.

O espaço entre as colunas são definidos com “.” (pontos). Veja o exemplo abaixo:

```
display: "a a a" /150px
". . ." /50px
"b c d"
". . ." /50px
"e e e" /150px
100px 400px 100px;
```



No exemplo acima fiz duas colunas no código compostas por pontos em vez de fazer com letras. Isso quer dizer que entre as colunas do grid haverá um espaço em branco de 50px de altura. Veja a imagem abaixo para entender melhor o código:

## O funcionamento da propriedade position

O valor da propriedade position especifica qual linha e coluna o elemento será colocado no template. Você escreve apenas uma letra por elemento. Vários elementos podem ser colocados em um mesmo slot. Logo estes elementos terão o mesmo valor de position.

Abaixo veja uma imagem que pegamos diretamente do exemplo do W3C. O layout é muito simples:

• navigation	<b>Weather</b> There will be weather	<b>Football</b> People like football.  <b>Chess</b> There was a brawl at the chess tournament	<b>Your Horoscope</b> You're going to die (eventually).
		Copyright some folks	

Este layout é representado pelo código abaixo. Primeiro o HTML:

```
<ul id="nav">
<li>navigation</li>
</ul>
<div id="content">
<div class="module news">
<h3>Weather</h3>
<p>There will be weather</p>
</div>
<div class="module sports">
<h3>Football</h3>
<p>People like football.</p>
</div>
<div class="module sports">
<h3>Chess</h3>
<p>There was a brawl at the chess tournament</p>
</div>
<div class="module personal">
<h3>Your Horoscope</h3>
<p>You're going to die (eventually).</p>
</div>
<p id="foot">Copyright some folks</p>
</div>
```

Agora veja o CSS com toda a mágica:

```
body {
display: "a b"
10em *;
}
#nav {
position: a;
}
#content {
```

```

position: b;
display: "c . d . e "
". . . . " /1em
". . f . .
* 1em * 1em *;
}
.module.news {position: c;}
.module.sports {position: d;}
.module.personal {position: e;}
#foot {position: f;}

```

Lembre-se que não importa a posição dos elementos no código. E essa é a mágica. Podemos organizar o código HTML de acordo com nossas necessidades e levando em consideração SEO, Acessibilidade e processo de manutenção. O HTML fica totalmente intacto separado de qualquer formatação. Muito, mas muito interessante.

## Pseudo-elemento ::slot()

Você pode formatar um slot específico simplesmente declarando-o no CSS. Suponha que você queira que um determinado slot tenha um fundo diferente, alinhamento e etc... Essa formatação será aplicada diretamente no slot e não no elemento que você colocou lá. Fica mais simples de se formatar porque você não atrela um estilo ao elemento e sim ao slot. Se você precisar posicionar aquele elemento em outro lugar, você consegue facilmente.

```

body { display: "aaa"
"bcd" }
body::slot(b) { background: #FF0 }
body::slot(c), body::slot(d) { vertical-align: bottom }

```

As propriedades aplicadas no pseudo elemento slot() seguem abaixo:

- Todos as propriedades background.
- vertical-align
- overflow
- box-shadow, block-flow e direction ainda estão sendo estudados pelo W3C se elas entrarão ou não.

# 10. CORES

## RGBA

Normalmente em web trabalhamos com cores na forma de hexadecimal. É a forma mais comum e mais utilizada desde os primórdios do desenvolvimento web. Mesmo assim, há outros formatos menos comuns que funcionam sem problemas, um destes formatos é o RGB. O RGB são 3 conjuntos de 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. Exemplo:

```
verdana;  
}
```

Este código RGB define que o background o elemento P será amarelo

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

## RGBA e a diferença da propriedade OPACITY

Até então nós só podíamos escrever cores sólidas, sem nem ao menos escolhermos a opacidade dessa cor. 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. Veja o exemplo abaixo e compare as imagens.

A primeira é a imagem normal, sem a aplicação de opacidade:



UM TEXTO PARA LER!



Agora com o bloco branco, marcado com um P, com opacidade definida. Perceba que o background e também a cor do texto ficaram transparentes.

Isso é 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 3 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:



Veja um exemplo de código aplicado abaixo:

```
p {
background:rgba(255,255,0, 0.5);
padding:10px;
font:13px verdana;
}
```

O último valor é referente ao canal Alpha, onde 1 é totalmente visível e 0 é totalmente invisível. No exemplo acima está com uma opacidade de 50%.



## ***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 abaixo:

```
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:

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

Veja que apliquei 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 seletores separados, ou seja, você não consegue atrelar o valor da propriedade `color` ao `currentColor` de outro seletor.

## 11. PAGED MEDIA

Com certeza você já deve ter tentado ler um livro ou uma apostila em algum site na web e preferiu imprimir o texto para ler off-line, no papel por ser mais confortável ou por ser mais prático quando não se está conectado. Existem vários motivos para que um leitor queira imprimir o conteúdo de um site, principalmente sites com textos longos e pesados. Durante muito tempo o principal motivo era que ler na tela do computador era cansativo. Hoje isso ainda é um problema, mas com o avanço das telas e do aparecimento das tablets no mercado, você consegue passar mais tempo na frente de uma tela lendo grandes quantidades de texto. O problema é que geralmente a organização de páginas e o conteúdo não é exatamente confortável para passarmos horas lendo.

Outro problema comum é que nós desenvolvedores não temos uma maneira fácil de formatar páginas. Na verdade temos, mas é um pouco de gambiarra e claro, não é maneira correta. A especificação de Paged Media traz nos possibilita formatar as páginas, transparências (aqueles “plásticos” que usamos com retroprojetores) ou até mesmo páginas que serão vistas pelo monitor. Controlaremos suas medidas, tamanhos, margens, quebras de páginas e etc...

**Nota:** Para você não se confundir, quando digo páginas, quero dizer páginas físicas, de papel, não páginas web, ok? ;-)

## @page

Definiremos com CSS3 um modelo de página que especifica como o documento será formatado em uma área retangular, chamada de page box, com larguras e alturas limitadas. Nem sempre o page box tem referência correspondente para uma folha de papel física, como normalmente conhecemos em diversos formatos: folhas, transparências e etc. Esta especificação formata o page box, mas é o browser ou o meio de acesso que o usuário está utilizando que tem a responsabilidade de transferir o formato do page box para a folha de papel no momento da impressão.

O documento é transferido no modelo da página para um ou mais page boxes. O page box é uma caixa retangular que será sua área de impressão. Isso é como se fosse um viewport do browser. Como qualquer outro box, a page box tem margin, border, padding e outras áreas. O conteúdo e as áreas de margin do page box tem algumas funções especiais:

A área de conteúdo do page box é chamada de area da página ou page area. O conteúdo do documento flui na área de página. Os limites da área da página na primeira página estabelece o retângulo inicial que contém o bloco do documento.

A área da margem da page box é dividido em 16 caixas de margem ou margin boxes. Você pode definir para cada caixa de margem sua própria borda, margem, padding e áreas de conteúdo. Normalmente essas caixas de margem são usadas para definir headers e footers do documento. Confesso que o nome utilizado (caixas de margem) é meio estranho.

As propriedades do page box são determinadas pelas propriedades estabelecidas pelo page context, o qual é a regra de CSS @page. Para definir a largura e altura de uma page box não se usa as propriedades width e height. O tamanho da page box é especificada utilizando a propriedade size no page context.

## Terminologia e Page Model (modelo de página)

O page box tem algumas áreas simples de se entender que facilitará a explicação. Veja abaixo uma imagem e uma explicação de suas respectivas áreas:

### **Page box**

O page box é onde tudo acontece. Tenha em mente que o page box é o viewport das medias impressas. É lá que conterá as áreas de margem, padding, border e onde o texto será consumido.

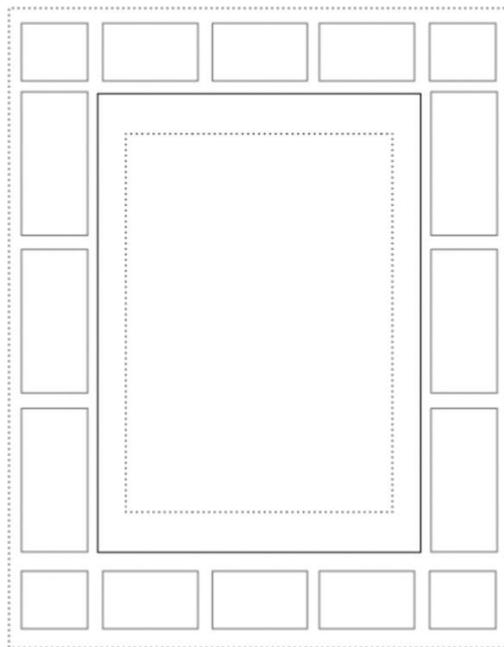
A largura e altura do page box é determinada pela propriedade size. Em um caso simples, o page box tem a largura e a altura de uma folha. Entretanto em casos complexos onde page box difere das folhas de papel em valores e orientações já que você pode personalizar de acordo com sua necessidade.

**Page area**

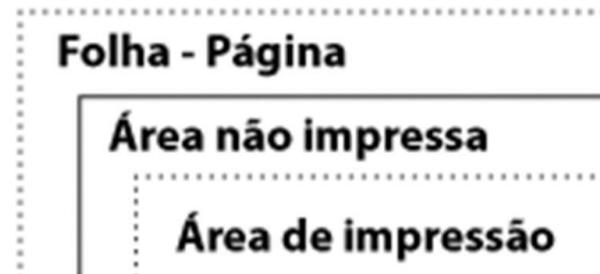
A page area é a área de conteúdo (content area) do page box.

**Margin box**

Margin boxes contém boxes para header e footer. São conjunto de 16 boxes onde você pode inserir conteúdo útil como número da página, título do livro, etc, etc, etc. Essas áreas ficam fora do Page area. Cada um tem suas margins, paddings e bordas individuais. Veja o diagrama abaixo para visualizar melhor.

**Page sheet**

A folha, a página, a superfície que será impresso o conteúdo. A ilustração abaixo mostra a representação de uma folha.

**Non-printable area - Área não impressa**

A área de não impressão é a área onde o dispositivo de impressão não é capaz de imprimir. Esta área depende do dispositivo que você está utilizando. O page box fica dentro da área de impressão.

## Área de impressão

A área impressa é onde o dispositivo de impressão é capaz de imprimir. A área de impressão é o tamanho da page sheet menos a área de não impressão. Como a área de não impressão, a área útil de impressão depende muito do dispositivo. O dispositivo pode ajustar o conteúdo para que seja impresso sem problemas nessa área. Cada dispositivo tem seu meio de ajuste.

## Propriedade size

A propriedade size especifica o tamanho e a orientação da área do de conteúdo, o page box. O tamanho do page box pode ser definida com valores absolutos (px) ou relativos (%). Você pode usar três valores para definir a largura e a orientação do page box:

### **auto**

O page box irá ter o tamanho e orientação do page sheet escolhido pelo usuário.

### **landscape**

Define que a página será impressa em modo paisagem. O page box neste caso tem o mesmo tamanho da página, mas o lado maior é o horizontal. Se o tamanho da página não for definido, o tamanho é especificado pelo usuário e seu dispositivo.

### **portrait**

Define que a página será impressa em modo retrato. O page box neste caso tem o mesmo tamanho da página, mas o lado maior é o vertical. Se o tamanho da página não for definido, o tamanho é especificado pelo usuário e seu dispositivo.

Veja um exemplo abaixo:

```
@page {
  size: auto; /* auto is the valor padrão */
  margin: 10%; /* margem */
}
```

Como nessa caso a margem é variável, ela está sendo relativa às dimensões da página. Logo se a página uma A4, com as dimensões: 210mm x 297mm, as margens serão 21mm e 29.7mm.

Outro exemplo:

```
@page {
  size: 210mm 297mm; /* definem o page-sheet para um tamanho de
A4 */
}
```

## Page-size

O page-size pode ser especificado utilizando um dos media names abaixo. Isso é o equivalente a utilizar os valores escritos diretamente na propriedade size. Contudo é muito melhor utilizar o nome de um formato de formato de papel.

Formato	Descrição
A5	A página deve ser definida para o tamanho ISO A5: 148mm x 210mm.
A4	A página deve ser definida para o tamanho ISO A4: 210 mm x 297 mm.
A3	A página deve ser definida para o tamanho ISO A3: 297mm x 420mm.
B5	A página deve ser definida para o tamanho ISO B3 media: 176mm x 250mm.
B4	A página deve ser definida para o tamanho ISO B4: 250mm x 353mm.
letter	A página deve ser definida para o tamanho papel carta: 8.5 pol x 11 pol

O W3C tem uma especificação muito extensa que pode ser encontrada aqui:

## 12. @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, nós disponibilizamos as fontes necessárias em seu servidor e linkamos estas fontes no arquivo CSS. A sintaxe é bem simples e tem suporte a todos os navegadores, com algumas ressalvas.

```
@font-face {
    font-family: helveticaneue;
    src: url(helveticaNeueLTStd-UltLt.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. Você aplica a fonte como abaixo:

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

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. Menos requisições, mais velocidade. Veja o código abaixo:

```
@font-face {
    font-family: helveticaneue;
    src: local(HelveticaNeueLTStd-UltLt.otf),
         url(HelveticaNeueLTStd-UltLt.otf);
}
```

Abaixo segue uma série de formatos que podem ser usados e que os browsers podem adotar:

String	Font Format	Common Extensions
<b>truetype</b>	TrueType	.ttf
<b>opentype</b>	OpenType	.ttf, .otf
<b>truetype-aat</b>	TrueType with Apple Advanced Typography extensions	.ttf
<b>embedded-opentype</b>	Embedded OpenType	.eot
<b>woff</b>	WOFF (Web Open Font Format)	.woff
<b>svg</b>	SVG Font	.svg, .svgz

## Compatibilidade

As versões 7, 8 e 9 do Internet Explorer aceitam o @font-face apenas se a font for EOT. Você pode encontrar qualquer conversor online que esse problema é resolvido. Você pode converter suas fontes para EOT diretamente no Font Squirrel. O Safari, Firefox, Chrome e Opera aceitam fontes em TTF e OTF.

Para suprir a necessidade de atenção do Internet Explorer, você precisa especificar a URL da font .eot para que o Internet Explorer possa aplicar a font corretamente. A sintaxe fica desta forma:

```
@font-face {  
    font-family: helveticaNeue;  
    src: url(helveticaNeueLTStd-UltLt.eot);  
    src: url(helveticaNeueLTStd-UltLt.otf);  
}
```

### Kit de sobrevivência

O Font Squirrel fez um pequeno favor para toda a comunidade. É um sisteminha que converte suas fontes para os formatos necessários e te devolve para você utilizar em seus sites:

<http://migre.me/4qST9>

# 13. PRESENTATION-LEVELS

A informação na web é reutilizada de diversas maneiras. Toda informação publicada é reutilizada por diversos meios de acesso, seja o seu browser, leitor de tela ou robôs de busca. O HTML proporciona essa liberdade para a informação. Por ser uma linguagem muito simples, podemos reutilizar a informação marcada com HTML em diversos meios de acesso. Mas o HTML não cuida da forma com que o usuário vai visualizar a informação em seu dispositivo. O HTML apenas exibe a informação. A maneira que o usuário consome essa informação é diferente em cada um dos meios de acesso e dispositivos. É aí que entra todo o poder do CSS. O CSS formata a informação para que ela possa ser acessível em diversos usos agents (meios de acesso). Se você acessa o site do seu banco pelo monitor de 22" da sua casa ou pelo seu celular, a informação tem que aparecer bem organizada em ambos os cenários. É o CSS que organiza visualmente essas informações.

Além disso, podemos apresentar a informação de diversas formas em um mesmo dispositivo. Por exemplo: você pode ver uma galeria de imagens da maneira convencional, clicando nas thumbs das fotos ou ver em forma de slideshow. Podemos levar essas experiências para websites de conteúdo textual também. A especificação de presentation-levels é uma das especificações que levam o usuário a terem conteúdo mostrado de uma outra forma da qual estamos acostumados. É muito útil para apresentações de slides, com efeitos, transições e etc ou qualquer documento que seria mais bem apresentado no formato de apresentação, como uma proposta, documentos técnicos e etc.

## Como funciona o modelo

O modelo por trás da especificação é simples. Cada elemento no documento é definido como um “elemento de apresentação” ou no formato original “element's presentation level” - EPL.

O EPL pode ser explícito em uma folha de estilo ou calculado automaticamente baseado na posição do elemento pela estrutura do documento. É assim que o browser calcula para mostrar os elementos progressivamente, como se faz normalmente em programas de apresentação.

O elemento fica em um dos três seguintes níveis que também são representadas por classes: below-level, at-level e above-level. Dependendo da pontuação de EPL que o browser dá, o elemento fica em um determinado nível. Essas pseudo-classes podem e devem ser modificadas via CSS.

## A propriedade presentation-level

A propriedade presentation-level define como os valores de apresentação (EPL) de um determinado objeto devem ser calculados. São três valores possíveis: números inteiros, increment e same.

Quando definimos um valor inteiro, o elemento tem aquele valor fixo.

Quando colocamos increment, o valor do objeto aumenta um ponto em relação ao objeto anterior. Suponha que há duas LI em uma UL. A primeira LI tem o valor de 1, a segunda tem valor de 2 e assim por diante.

Quando definimos o valor same, o browser computa o mesmo valor do objeto anterior.

Isso tudo vai ficar mais esclarecido com os exemplos a seguir.

Utilizando o mesmo exemplo da especificação do W3C, temos o código abaixo:

```
<!DOCTYPE html>
<html>
<body>
<h1>strategies</h1>
<h2>our strategy</h2>
<ul>
<li>divide</li>
<li>conquer
<p>(in that order)</p>
</li>
</ul>
<h2>their strategy</h2>
<ul>
<li>obfuscate</li>
<li>propagate</li>
</ul>
</body>
</html>
```

Vamos definir o CSS de presentation-levels para esse HTML adicionado o código CSS:

```
@media projection {
  h1 { page-break-before: always }
  li { presentation-level: increment }
  :below-level { color: black }
  :at-level { color: red }
  :above-level { color: silver }
}
```

Definimos que o H1 irá sempre iniciar em uma nova página.

Mas o mais importante é a propriedade presentation-level que definimos para a LI. Isso quer dizer que a cada LI o browser contará mais um ponto.

As três pseudo-classes que falamos no começo do texto: below-level, at-level, above-level, que formata os elementos que foram mostrados anteriores, o que elemento que está sendo mostrado e o próximo elemento.

Sendo assim, o browser calcula a pontuação de cada um dos elementos utilizados no exemplo como mostra abaixo:

HTML	Valor de EPL
<h1>strategies</h1>	0
<h2>our strategy</h2>	0
<ul>	0
<li>divide</li>	1
<li>conquer</li>	2
</ul>	0
<h2>their strategy</h2>	0
<ul>	0
<li>obfuscate</li>	1
<li>propagate</li>	2
</ul>	0

Temos um outro exemplo, segue abaixo o HTML e logo depois a tabela com os valores de EPL:

```
<!DOCTYPE html>
<html>
<style>
@media projection {
h1 { presentation-level: 0; }
h2 { presentation-level: 1; }
h3 { presentation-level: 2; }
body * { presentation-level: 3; }
:above-level { display: none; }
}
</style>
<body>
<h1>strategies</h1>
<h2>our strategy</h2>
<ul>
<li>divide</li>
<li>conquer</li>
</ul>
<h2>their strategy</h2>
<ul>
<li>obfuscate</li>
<li>propagate</li>
</ul>
</body>
</html>
```

Perceba que agora definimos no CSS que tudo dentro de body tem o valor de 3. Logo o H1 que foi definido como 0 pela propriedade presentation-level tem o valor de 3.

Definimos também display:none; para os próximos elementos. Agora veja a pontuação aplicada:

HTML	Valor de EPL
<h1>strategies</h1>	3
<h2>our strategy</h2>	2
<ul>	0
<li>divide</li>	0
<li>conquer</li>	0
</ul>	0
<h2>their strategy</h2>	2
<ul>	0
<li>obfuscate</li>	0
<li>propagate</li>	0
</ul>	0

# JAVASCRIPT

# INTRODUÇÃO

JavaScript é uma linguagem para auxilio na criação de WebSites, as funções escritas em JavaScript podem ser embutidas dentro de seu documento HTML e outras linguagens web, possibilitando o incremento das funcionalidades do seu documento HTML com elementos interessantes. Sendo possível: responder facilmente a eventos iniciados pelo usuário, incluir efeitos que tornem sua página dinâmica. Logo, podemos criar sofisticadas páginas com a ajuda desta linguagem.

Ainda que os nomes sejam quase os mesmos, Java não é o mesmo que JavaScript. Estas são duas técnicas diferentes de programação na Internet. Java é uma linguagem de programação. JavaScript é uma linguagem de hiper-texto. A diferença é que você realmente pode criar programas em Java. Mas muitas vezes você precisa apenas criar um efeito bonito sem ter que se incomodar com programação. A solução então é JavaScript pois é fácil de entender e usar.

Podemos dizer que JavaScript é mais uma extensão do HTML do que uma linguagem de programação propriamente dita. O primeiro browser a suportar JavaScript foi o Netscape Navigator 2.0, mas a versão para o "Mac" parece apresentar muitos bugs.

Referências: <http://www.w3schools.com/js/>

## CONSIDERAÇÕES INICIAIS

Em documentos HTML, a utilização da linguagem JavaScript, se dá sob a forma de funções (applets), as quais são chamadas em determinadas situações ou em resposta a determinados eventos, estas funções podem estar localizadas em qualquer parte do código HTML, a única restrição é que devem começar com a declaração <SCRIPT> e termina com o respectivo </SCRIPT>, por convenção costuma-se colocar todas as funções no início do documento (estre as TAGs <HEAD> e </HEAD>, isso para garantir que o código JavaScript seja carregado antes que o usuário interaja com a Home Page), ou seja, antes do <BODY>.

Exemplo:

```
<HTML>
<HEAD>
<TITLE>Exemplo</TITLE>
<!--
Se houvesse alguma função seria bom declará-la aqui!!!
-->
</HEAD>
<BODY>
Esta linha está escrita em HTML
<SCRIPT>
document.write("Alô Mundo!!!!");
</SCRIPT>
Voltamos para o HTML
</BODY>
</HTML>
```

É importante ressaltar que todas as linhas devem ser terminadas com; (ponto e vírgula) a menos que a próxima instrução seja um “else” e se você precisar escrever mais de uma linha para executar uma condição seja ela em uma estrutura “for”, “if” ou “while”, este bloco de instruções deve estar entre “{ }” (chaves). Inclusive a definição de funções segue este modelo, ou seja, todo o código da função deve estar limitado por { (no início) e } (no final).

Um browser que não suporta JavaScript, ele não conhece a TAG <SCRIPT>. Ele ignora a TAG e logicamente tudo todo o código que estiver sendo limitado por ela, mostrando todo o código na tela como se fosse um simples texto HTML. Deste modo o usuário veria o código JavaScript do seu programa dentro do documento HTML e como certamente essa não deve ser sua intenção, existe um meio de esconder o código JavaScript dos browsers que não

conhecem esta linguagem, basta utilizar os comentários HTML “`<!-- - -->`”. O código do nosso exemplo anterior ficaria assim:

```
<HTML>
<HEAD>
<TITLE> Exemplo </TITLE>
<!--
...
Se houvesse alguma função seria bom declará-la aqui!!!
...
-->
</HEAD>
<BODY>
Esta linha está escrita em HTML
<SCRIPT>
<!-- Esconde o código JavaScript dos browsers mais antigos
document.write("Aqui já é JavaScript");
// -->
</SCRIPT>
Voltamos para o HTML
</BODY>
</HTML>
```

Se o browser não suportar JavaScript e não inserirmos o comentário HTML, o que apareceria na tela seria:

```
Esta é uma linha escrita em HTML
document.write("Aqui já é JavaScript");
Voltamos para o HTML
```

Note que esse artifício não esconde completamente o código JavaScript, o que ele faz é prevenir que o código seja mostrado por browsers mais antigos, porém o usuário tem acesso a todas as informações do código fonte de sua Home Page (tanto HTML, quanto JavaScript).

## Variáveis

Em JavaScript, variáveis dinâmicas podem ser criadas e inicializadas sem declarações formais.

Existem dois tipos de abrangência para as variáveis:

**"Global"** - Declaradas/criadas fora de uma função. As variáveis globais podem ser acessadas em qualquer parte do programa.

**"Local"** - Declaradas/criadas dentro de uma função. Só podem ser utilizadas dentro da função onde foram criadas e precisa ser definida com a instrução Var.

Com relação à nomenclatura, as variáveis devem começar por uma letra ou pelo caractere sublinhado “\_”, o restante da definição do nome pode conter qualquer letra ou número. É importante ressaltar que a variável **“Código”** é diferente da variável **“código”**, que por sua vez é diferente de **“CODIGO”**, sendo assim, muito cuidado quando for definir o nome das variáveis, utilize sempre um mesmo padrão.

Existem três tipos de variáveis: **Numéricas, Booleanas e Strings**.

Como já era de se esperar, as variáveis numéricas são assim chamadas, pois armazenam números, as Booleanas valores lógicos (True/False) e as Strings, seqüência de caracteres. As strings podem ser delimitadas por aspas simples ou duplas, a única restrição é que se a delimitação começar com as aspas simples, deve terminar com aspas simples, da mesma forma para as aspas duplas. Podem ser incluídos dentro de uma string alguns caracteres especiais, a saber:

- \t** - posiciona o texto a seguir, na próxima tabulação;
- \n** - passa para outra linha;
- \f** - form feed;
- \b** - back space;
- \r** - carriage return.

O JavaScript reconhece ainda um outro tipo de contudo em variáveis, que é o **NULL**. Na prática isso é utilizado para a manipulação de variáveis não inicializadas sem que ocorra um erro no seu programa. Quando uma variável possui o valor **NULL**, significa dizer que ela possui um valor desconhecido ou nulo. A representação literal para **NULL** é a string '**null**' sem os delimitadores. Quando referenciado por uma função ou comando de tela, será assim que **NULL** será representado. Observe que **NULL** é uma palavra reservada.

Você pode trabalhar ainda com Arrays, mas para isso será necessário algum conhecimento sobre Programação Orientada a Objetos.

## Operadores

Junto com funções e variáveis, operadores são blocos de construção de expressões. Um operador é semelhante a uma função no sentido de que executa uma operação específica e retorna um valor.

## Operadores unários e binários

Todos os operadores em JavaScript requerem um ou dois argumentos, chamados operandos. Aqueles que requerem um operando apenas são denominados operadores unários, e os que requerem dois operandos são chamados de operadores binários.

## Operador de String

Operador de concatenação (+)

Exemplo:

```
Nome1="José"
```

```
Nome2="Silva"
```

```
Nome = Nome1+ " da "+Nome2 // O resultado é: "José da Silva"
```

## Operadores Matemáticos

*Adição ( + )*

Exemplo:

```
V01=5
```

```
V02=2
```

```
V=V01+V02 // resulta em: 7
```

*Subtração ( - )*

Exemplo:

```
V01=5
```

```
V02=2
```

```
V=V01-V02 // resulta em: 3
```

*Multiplicação ( \* )*

Exemplo:

```
V01=5
```

```
V02=2
```

```
V=V01*V02 // resulta em: 10
```

*Divisão ( / )*

Exemplo:

```
V01=5
```

```
V02=2
```

```
V=V01/V02 // resulta em: 2.5
```

*Módulo da divisão ou resto ( % )*

Exemplo:

```
V01=5
```

```
V02=2
```

```
V=V01%V02 // resulta em: 1
```

***Incremento (++)***

Pode acontecer de duas formas:

++Variável ou

Variável++

Exemplo:

v01 = 5

v02 = ++v01 // Resulta em 6

v03 = v01 // Resulta em 6

Exemplo:

v01 = 5

v02 = v01++ // Resulta em 5

v03 = v01 // Resulta em 6

***Decremento (--):***

Pode acontecer de duas formas:

--Variável ou

Variável--

Exemplo:

v01 = 5

v02 = --v01 // Resulta em 4

v03 = v01 // Resulta em 4

Exemplo:

v01 = 5

v02 = v01-- // Resulta em 5

v03 = v01 // Resulta em 4

## Operadores relacionais

< Menor que

> Maior que

== Igual

!= Diferente

>= Maior ou igual

<= Menor ou igual

## Operadores lógicos

&& E lógico

|| Ou lógico

## Operadores de atribuição

= Atribuir

+= Soma ou concatenação e atribuição: x+=5 // é o mesmo que: x=x+5

-= Subtração e atribuição. x-=5 // é o mesmo que: x=x-5

\*= Multiplicação e atribuição.  $x*=5$  // é o mesmo que:  $x=x*5$   
/= Divisão e atribuição.  $x/=5$  // é o mesmo que:  $x=x/5$   
%= Módulo e atribuição.  $x\%=5$  // é o mesmo que:  $x=x\%5$

## Objetos

Quando compramos um televisor, recebemos um manual, que por mais simples que possa ser, traz sempre algumas especificações técnicas do aparelho. Por exemplo: Polegadas da tela, voltagem de trabalho, entre outras. Essas especificações técnicas transferido para o vocabulário da OOP são as propriedades do objeto (televisor). Em JavaScript essas propriedades nada mais são do que variáveis internas do objeto.

Um objeto está sujeito a determinados métodos. Um método geralmente é uma função que gera alguma informação referente ao objeto. Por exemplo ao mudar de canal, nós estamos executando uma função do televisor, o mesmo ocorre quando aumentamos ou diminuímos o volume.

Seguindo nosso exemplo, quando a tensão da rede sai da faixa de trabalho no caso de uma queda de tensão ou uma sobrecarga, o sistema de segurança da Tv, não permite que ocorram danos no aparelho, quando muito, queima o fusível da fonte de alimentação. Em aparelhos mais modernos, quando uma emissora sai do ar, a tela fica azul, sem aquele chiado irritante. Sendo assim podemos concluir que nosso objeto está sujeito a algumas situações, estas situações podem ocorrer a qualquer momento, e são chamadas de eventos.

## Criando Objetos

Trabalhar com objetos é a única forma de manipular os arrays, vejamos como: Digamos que desejamos programar uma lista de clientes, nosso objeto poderia ser definido assim:

```
Function CriaClientes(nome, endereco, telefone, renda)
{
  this.nome=nome;
  this.endereco=endereco;
  this.telefone=telefone;
  this.renda=renda;
}
```

A propriedade “**this**” especifica o objeto atual como sendo fonte dos valores passados a função. Agora, basta criar o nosso objeto:

```
Maria = New CriaClientes('Maria Aparecida', 'Rua Guilhotina dos Patos, S/N', '332-1148', 1300)
```

Para acessar as propriedades do objeto Maria, basta usar a seguinte sintaxe:

```
Maria.nome - retorna 'Maria Aparecida'
Maria.endereco - retorna 'Rua Guilhotina dos Patos, S/N'
Maria.telefone - retorna '332-1148'
Maria.renda - retorna 1300
```

Uma outra forma de referenciar as propriedades do objeto Maria, é:

```
Maria[0], Maria[1], Maria[2], Maria[3]
```

Uma forma mais prática de criar arrays poderia ser a seguinte:

```
Function Matriz(n)
{
  this.length=n
  for (var contador=1 ; contador <=n ; conatdor=contador+1)
  {
    this[contador]=""
  }
}
```

Para criar nossa matriz usariamos o seguinte comando:

```
Mes=Matriz(12)
```

O próximo passo seria incluir os dados:

```
Mes[1] = 'Janeiro'
Mes[2]='Fevereiro'
...
Mes[12]='Dezembro'
```

Podemos também utilizar os dois métodos ao mesmo tempo!

```
Clientes=New Matriz(3)
Clientes[1]=New CriaClientes("Charlene", "Rua A, 51", "331-0376", 1150)
Clientes[2]=New CriaClientes("José", "Rua das Avencas, S/N", "332-2781", 950)
Clientes[3]=New CriaClientes("Joaquim Manoel", "Rua Amancio Pinto, 171", , 1000)
```

Teríamos agora:

```
Clientes[1].nome = "Charlene";
Clientes[2].telefone="332-2781"
Clientes[3].telefone=null
```

## Comandos

Além das estruturas de controle, o JavaScript oferece alguns poucos comandos:

Break

Continue

Var

With

Function

Return

Comment

## VAR

Em JavaScript, nem sempre é necessário definir uma variável antes de utilizá-la, é o que ocorre com variáveis globais, porém, é importante ressaltar que a utilização da instrução “**var**”, a nível de documentação é muito bem-vinda. Já nas definições de variáveis locais, é obrigatório a utilização da instrução var.

Você pode armazenar um valor na própria linha de definição da variável, se não o fizer, para o JavaScript, esta variável possui um valor desconhecido ou nulo.

Não é obrigatoria a utilização de uma instrução “**var**” para cada variável declarada, na medida do possível, você pode declarar várias variáveis em uma só instrução var.

```
Var NomeDaVariável [ = <valor> ] ;
Var Contador = 0;
Var Inic="",Tolls=0,Name="TWR";
Var Teste; // Neste caso, Teste possui valor null
```

## with

Quando você precisa manipular várias propriedades de um mesmo objeto, provavelmente prefere não ser obrigado a repetir todas as vezes a digitação do nome do objeto. A instrução “**with**” permite fazer isso eliminando a necessidade de digitar o nome do objeto todas às vezes.

Forma geral:

```
with (<objeto>)
{
  ... Instruções
}
```

Por exemplo vamos supor que será necessário executar uma série de operações matemáticas:

```
with (Math)
{
  a=PI;
  b=Abs (x) ;
  c=E;
}
```

## Break

Pode ser executado somente dentro de loops “**for**”... ou “**while**”... e tem por objetivo o cancelamento da execução do loop sem que haja verificação na condição de saída do loop, passando a execução a linha imediatamente posterior ao término do loop.

Forma geral:

Break

Exemplo:

Neste exemplo, quando a variável x atinge o valor 5 o loop é cancelado, e é impresso o número 5 na tela.

```
For (var x=1 ; x < 10 ; x++)
{
If (x == 5)
{
Break
}
document.write(x) // resulta: 5
```

## Continue

Pode ser executado somente dentro de loops “**for**”... ou “**while**” ... e tem por objetivo o cancelamento da execução do bloco de comandos passando para o início do loop.

Forma geral:

Continue

Exemplo:

Neste exemplo, serão impressos os números de 1 a 10, com exceção do número 5, ou seja, quando a variável “**x**” atinge o valor 5 a execução do bloco de comandos é interrompida e o controle retorna ao início do loop, onde a variável “**x**” será incrementada.

```
For (var x=1 ; x < 10 ; x++)
{
If (x == 5)
{
continue
}
document.write(x)
}
```

## Funções

As funções podem ser definidas como um conjunto de instruções, agrupadas para executar uma determinada tarefa. Dentro de uma função pode existir uma chamada a outra função. Para as funções podem ser passadas informações, as quais são chamadas de parâmetros.

As funções podem ou não retornar alguma informação, o que é feito com o comando **Return**.

A definição de uma função é feita da seguinte forma:

```
Function NomeDaFunção([ parametro1, parametro2, ..., parametroN ])
{
...
[Return(ValorDeRetorno) ]
```

A chamada de funções é feita da seguinte forma:  
NomeDaFunção( [parâmetros] )

Funções são mais bem declaradas entre as tags `<head>` de sua página *HTML*. Funções são frequentemente chamadas por eventos acionados pelo usuário. Assim parece razoável colocar as funções entre as tags `<head>`. Elas são carregadas antes que o usuário faça alguma coisa que possa chamar uma função.

```
<html>
<head>
<script language="LiveScript">
Function hello(){
alert("Alô mundo!!!!");
}
</script>
</head>
<body>
...
<script>hello();</script>
...
</body>
</html>
```

## Comentários

Comentários podem ser formulados de várias maneiras, dependendo do efeito que você precisa. Para comentários longos de várias linhas, ou blocos de comentários, use:

/\* O barra-asterisco inicia um bloco de comentário que pode conter quantas linhas você precisar todo o texto após o barra asterisco é ignorado, até que asterisco-barra seja encontrado, terminando assim o bloco de comentário \*/

Para comentários de uma linha, use barra dupla (`//`) para introduzir o comentário. Todo o texto seguindo este símbolo até o próximo *carriage-return* será considerado um comentário e ignorado para fins de processamento. Exemplo:

`// este texto será tratado como comentário`

Os códigos JavaScript podem ser colocados em campos de comentário de modo que browsers antigos não mostrem o próprio código JavaScript, como vemos a seguir:

```
<html>
<head>
<script language="LiveScript">
```

```

<!-- hide script from old browsers
Function hello(){
alert("Alô mundo!!!!");
}
// end hiding contents -->
</script>
</head>
<body>
...
<script>hello();</script>
...
</body>
</html>

```

## Estruturas de Controle

Existem algumas estruturas de controle que lhe permitem modificar o fluxo de execução de um programa. Estas estruturas permitem executar o código baseado em condições lógicas ou um número determinado de vezes.

For...  
 For...In  
 If...Else...  
 While...

### For...

Repete uma seção do código um determinado número de vezes. Consiste de uma declaração que define as condições da estrutura e marca seu início. Esta declaração é seguida por uma ou mais declarações executáveis, que representam o corpo da estrutura.

Estabelece um contador inicializando uma variável com um valor numérico. O contador é manipulado através da *<ação>* especificada no comando toda a vez que o *loop* alcança seu fim, permanecendo nesse *loop* até que a *<condição>* seja satisfeita ou a instrução *Break* seja executada.

Forma geral:

```
For (<inicialização> ; <condição> ; <ação>)
{ Corpo da Estrutura }
```

No exemplo abaixo, o bloco de instruções será executado 10 vezes, pois a variável *contador* é inicializada com o valor 1 e o bloco de instruções será executado enquanto *contador* for menor que 11. A cada execução do bloco de instruções *contador* é incrementado.

```
For (var contador = 1; contador < 11; contador++)
{ document.write(Contador); }
```

## For...In

Este comando tem por objetivo, procurar a ocorrência de uma variável, dentro das propriedades de um objeto, ao encontrar a referida variável, um bloco de comandos pode ser executado.

Forma geral:

```
For (variavel In objeto)
{
    bloco de comandos
}
```

Exemplo: Esta função procura por uma propriedade do *Objeto*, cujo o nome esteja especificado pela variável *Procura*, onde *Nome* é uma string correspondendo ao nome do objeto.

```
Function SearchIn(Procura, Objeto, Nome)
{
Var ResultadoDaBusca = ""
For (Procura In Objeto)
{
document.write(Nome+"."+Procura+"="+Objeto[Procura]+"<BR>") ;
}
}
```

## If...Else...

A estrutura *If...* executa uma porção de código se a condição especificada for verdadeira. A estrutura pode também ser especificada com código alternativo para ser executado se a condição for falsa.

```
Function VerificaIdade(anos)
{
If anos >= 16
{
Return ('Já pode votar!')
}
else
{
Return ('Ainda é muito cedo para votar...')
}
}
```

Uma alternativa para economizar *If's* seria a utilização de uma expressão condicional, que funciona para situações mais simples, o exemplo acima ficaria da seguinte forma:

```
VariavelDeRetorno= (anos>=16) ? 'Já pode votar!' : 'Ainda é muito cedo para votar...'
```

## While

Outro tipo de *loop* é aquele baseado numa condição ao invés de no número de repetições. Por exemplo, suponha que você necessita que um determinado processo seja repetido até que um determinado teste dê um resultado verdadeiro ou seja executada a instrução *Break*.

Forma geral:

```
while (<condição>)
{ Corpo da Estrutura }
```

No exemplo abaixo, o bloco de instruções será executado 10 vezes, pois a variável *Contador* é inicializada com o valor 1 e o bloco de instruções será executado enquanto *Contador* for menor que 11. A cada execução do bloco de instruções *Contador* é incrementado.

```
Var Contador=1;
While ( Contador < 11 )
{ document.write(Contador++) ; }
```

## Funções internas

A linguagem JavaScript além dos recursos descritos anteriormente, ainda possui algumas funções internas, que não estão ligadas diretamente a nenhum objeto, porém isso não impede que essas funções recebam objetos como parâmetros. A seguir estas funções serão vistas detalhadamente:

***alert*** - Mostra uma caixa de alerta, seguido de um sinal sonoro e o botão de OK.

Ex: `alert('Esta é uma janela de alerta!')`

***confirm*** - Mostra uma caixa de diálogo, seguida de um sinal sonoro e os botão de OK e Cancel. Retorna um valor verdadeiro se o usuário escolher OK.

Ex: `retorno=confirm('Deseja prosseguir?')`

***escape*** - Obtém o código ASCII de um caracter que não seja alfa-numérico.

Ex: `document.write(escape("@"))`

***eval*** - Avalia uma expressão numérica, retornando um resultado também numérico.

Ex: `document.write(eval(10*9*8*7*6*5*4*3*2))`

***parseFloat*** - Converte uma *string* que representa um número, para um número com ponto flutuante. Caso a *string* não possa ser avaliada, a função retorna 0.

Ex: `document.write(parseFloat("-32.465e12"))`

***parseInt*** - Converte uma *string*, que representa um número em uma base predefinida para base 10. Caso a *string* possua um caracter que não possa ser convertido, a operação para, retornando o valor antes do erro.

Ex: `parseInt("string",base)`  
`parseInt("FF",15) // retorna 256`  
`parseInt("3A",10) // retorna 3`  
`parseInt("10",2) // retorna 2`

***prompt*** - Monta uma caixa de entrada de dados, de forma simplificada comparando-se com o objeto *text*.

Ex: `prompt(label [,valor])`

onde:

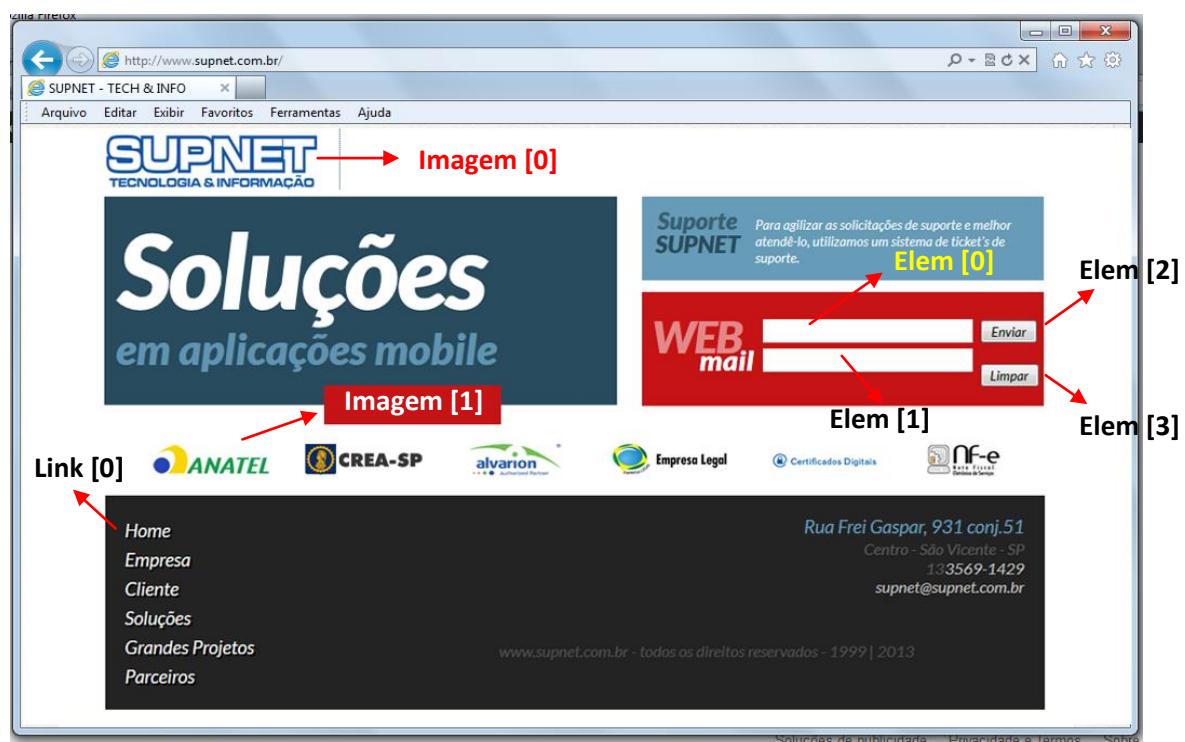
label - texto que aparece ao lado da caixa.

valor - é o conteúdo inicial da caixa.

## Objetos JavaScript

JavaScript organiza todos os elementos de uma Home Page dentro de uma hierarquia. Cada elemento é visto como um objeto. Os objetos podem ter propriedades, métodos e responder a certos eventos. Por isso é muito importante entender a hierarquia dos objetos HTML.

Você entenderá rapidamente como isto funciona com a ajuda de um exemplo. O exemplo seguinte é uma página HTML simples:



No exemplo acima, nós temos, um *link*, duas imagens, e um formulário com dois campos texto e dois botões. Do ponto de vista do JavaScript a janela do *browser* é um objeto *window*, que contém certos elementos, como a barra de *status*.

Dentro da janela, nós podemos carregar uma página *HTML*. Esta página é um objeto *document*. Desta forma o objeto *document* representa o documento *HTML* (que está

carregado no momento). O objeto *document* é muito importante, em JavaScript você sempre o usará muito. As propriedades e métodos do objeto *document* serão vistas detalhadamente, mais adiante.

Mas o que é mais importante é que todos os objetos *HTML* são propriedades do objeto *document*. Um objeto *HTML* pode ser por exemplo um *link* ou um formulário.

Nós podemos obter informações de diversos objetos e então manipulá-los. Para isso nós devemos saber como acessar os diferentes objetos. Você primeiro verifica o nome do objeto no diagrama de hierarquia. Se você então precisar saber como referenciar a primeira imagem na página *HTML*, basta seguir o caminho hierárquico. Você deve percorrer o diagrama de cima para baixo, o primeiro objeto é chamado **document**, a primeira imagem é representada por *Imagen[0]*. Desta forma nós podemos acessar este objeto em JavaScript, da seguinte forma: *document.Imagem[0]*.

Se você quiser saber o que o usuário digitou dentro do primeiro elemento do formulário, você

primeiro precisa pensar em como acessar esse objeto. Novamente nós seguiremos o diagrama de hierarquia, de cima para baixo. Siga o caminho que leva até *ELEM[0]*. Todos os nomes de objeto por onde você passou tem que constar na referência ao primeiro elemento do formulário. Desta forma você pode acessar o primeiro elemento texto assim: *document.Form[0].ELEM[0]*

Mas como obteremos agora, o texto digitado? Este elemento texto possui uma propriedade chamada *value* - não se preocupe agora, com propriedades, métodos ou eventos, eles serão vistos detalhadamente mais adiante - esta propriedade armazena o conteúdo do objeto, ou seja, o texto digitado. A seguinte linha de código obtém o texto digitado:

```
nome = document.forms[0].elements[0].value;
```

A *string* é armazenada na variável *name*. Nós podemos agora trabalhar com esta variável. Por exemplo, nós podemos criar uma janela *popup* com *alert("Oi "+name)*. Se a entrada for "Anderson" o comando *alert("Oi "+name)* abrirá uma janela popup com o texto "Oi Anderson".

Se você estiver trabalhando com páginas muito grandes pode ficar um pouco confuso referenciar objetos diretamente pelo endereçamento do diagrama de hierarquia, você pode ter referências do tipo: *document.forms[3].elements[15]* ou *document.forms[2].elements[21]*. Para evitar esse problema você pode dar nomes diferentes aos objetos, vejamos o seguinte fragmento de um documento *HTML*:

```
<form NAME="clientes">
Nome: <input TYPE="text" NAME="empresa" value=" ">
...
```

Dessa forma, em vez de usarmos, por exemplo:

```
document.forms[0].elements[0].value;
```

Podemos usar:

```
document.clientes.empresas.value;
```

Isto traz muitas facilidades, especialmente com páginas grandes e com muitos objetos. Observe que a caixa das letras faz diferença. Muitas propriedades dos objetos JavaScript não são apenas para leitura, você pode atribuir novos valores a algumas propriedades. Observe o exemplo:

#### Location

Este objeto contém informações sobre a URL da página atual.

Forma geral:

```
location.propriedade  
location.metodo()
```

Propriedades:

**hash** - Esta propriedade funciona de forma semelhante ao famigerado "*go to*" de algumas linguagens de programação. Normalmente é usado em *links*, que acessam a mesma página. Ex: O exemplo abaixo demonstra a utilização da propriedade *hash*, para criar um *link* para outro trecho da mesma página.

```
<HTML>  
...  
<A HREF = "location.hash='2'">Item 1</A>  
...  
<A NAME = "1"> </A>Item 1  
...  
<A NAME = "2"> </A>Item 2  
...  
</HTML>
```

**host** - Armazena uma *string* com o formato "hostname:port" da página HTML atual.  
Ex: alert('Demostraçāo da propriedade host: '+location.host)

**hostname** - Armazena uma string, com o IP da página HTML atual.  
Ex: alert('Demostraçāo da propriedade hostname:  
' + location.hostname)

**href** - String identica a mostrada na barra "location" do browser.  
Ex: alert('A URL desta página é: ' + location.href)

**pathname** - Contém uma string com o path da página HTML atual.  
Ex: alert('O path da URL desta página é: ' + location.pathname)

**port** - Armazena a porta por onde está sendo feita a conexão com o servidor.

Ex: alert('A porta usada para conexão com o servidor é: '+  
location.port)

**protocol** - String que armazena o protocolo de acesso a um determinado endereço.  
("http:","ftp:","file:").

Ex: alert('O protocolo de acesso para esta página é: '+  
location.protocol)

Métodos:

**toString** - Converte o conteúdo do objeto location para uma string.

Ex: alert('location.toString() = '+location.toString) // Este valor  
é o mesmo que location.href.

## Select

Cria uma listBox, no mesmo padrão que no Windows. Onde o usuário pode selecionar uma ou mais opções disponíveis, depende da configuração desejada pelo programador.

Forma geral:

```
<SELECT
NAME = "selectName"
[SIZE = tamanho]
[MULTIPLE]
[onBlur = "ação"]
[onChange = "ação"]
[onFocus = "ação"] >
<OPTION
VALUE = "optionValue"
[SELECTED] >
Texto
. . .
<OPTION...>
</SELECT>
```

onde:

**selectName** - Nome dado pelo programador, para o objeto select

**tamanho** - Número de linhas, da caixa select.

**MULTIPLE** - Se definido, permite que várias opções sejam selecionadas.

**ação** - Define o que fazer quando algum evento ocorrer.

**optionValue** - Valor que é enviado ao servidor, quando o formulário é submetido.

**SELECTED** - Se definido, informa a opção que será inicialmente selecionada.

Propriedades:

**length** - Informa o número de opções disponíveis.

Ex: selectName.length

**name** - Informa o nome que o programador definiu para o objeto select.

Ex: `selectName.name`

**options** - Vetor com todas as opções existentes no menu select.

Ex: `selectName.options[0..selectName.length-1]`

**selectedIndex** - Informa o índice do ítem que está selecionado.

Ex: `selectName.selectedIndex`

**defaultSelected** - Informa o ítem que detém a seleção inicial. Pode-se alterar este valor, desde que o formulário ainda não tenha sido exibido.

Ex: `selectName.options[Indice].defaultSelected`

**index** - Obtém o número do índice de uma opção em um menu select.

Ex: `selectName.options[Indice].index`

**selected** - Valor lógico referente a opção em questão. Se a opção estiver selecionada, retorna "1", caso contrário, retorna "0".

Ex: `selectName.options[indice].selected`

**text** - Armazena o texto que aparece como opção do menu select. Este texto é definido após a TAG <OPTION>.

Ex: `selectName.options[indice].text`

**value** - Armazena o campo VALUE, que é enviado ao servidor quando o formulário é submetido (enviado).

Eventos:

**onBlur** - Ocorre quando o objeto perde o foco.

**onChange** - Ocorre quando o objeto perde o foco e seu conteúdo foi alterado.

**onFocus** - Ocorre quando o objeto recebe o foco.

## Button

Este objeto mostra um botão 3-D (no mesmo padrão do Windows). Ao ser pressionado, ele produz um efeito de profundidade e geralmente chama uma função.

Pode ser utilizada para inúmeras aplicações, dependendo apenas de sua imaginação, a única precaução é defini-lo dentro de um formulário.

### Forma geral:

```
<Input Type="button" Name="NomeDoBotão" Value="Rótulo"
onClick="RespostaAoEvento">
```

### Onde:

Type - define o objeto

**Name** - define o nome do objeto para nossa aplicação. É por este nome que referenciamos alguma propriedade deste objeto

**Value** - define o que será escrito na face do botão

**onClick** - É o único evento possível para este objeto, normalmente define uma função a ser executada quando clicamos no botão.

### Propriedades:

**NAME:** Informa o nome do objeto button em forma de string, da mesma forma como definido no campo Name que foi utilizado na definição do botão. É importante não confundir o campo Name com a propriedade NAME, veja a diferença:

```
<input type="button" name="OK" value="Confirma"
onClick="ConfirmaInformacoes () ">
OK.Name // equivale a "OK"
```

**VALUE:** Informa o label do botão em forma de string da mesma forma como foi definido no campo Value que foi utilizado na definição do botão.

```
OK.Value // equivale a "Confirma"
```

### Métodos:

**click:** Este método simula um clique do mouse no objeto button, ou seja, executa um procedimento associado a um botão como se o botão tivesse sido pressionado mas sem que o usuário tenha realmente clicado.

```
OK.click() // executaria a função ConfirmaInformacoes
```

### Eventos associados:

**onClick:** Define o que fazer quando clicamos no objeto button

**Exemplo:**

```
<FORM>
<INPUT TYPE="button" VALUE="Clique aqui!!!" NAME="botao1"
onClick="alert('A propriedade NAME deste botão é:' +botao1.name + '\nA propriedade VALUE
deste botão
é:' +botao1.value)")>
</FORM>
```

## **Navigator**

Neste objeto ficam armazenadas as informações sobre o browser que está sendo utilizado. Forma geral:

Navigator.propriedade

**Propriedades:**

**appCodeName** - Armazena o codnome do browser.

Ex: Navigator.appCodeName

**appName** - Armazena o nome do browser.

Ex: Navigator.appName

**appVersion** - Armazena a versão do browser.

Ex: Navigator.appVersion

**userAgent** - Armazena o cabeçalho (user-agent) que é enviado para o servidor, no protocolo HTTP, isto serve para que o servidor identifique o software que está sendo usado pelo cliente.

Ex: Navigator.userAgent

**Exemplo:**

```
<HTML>
<HEAD>
<TITLE>JavaScript </TITLE>
<SCRIPT>
<!--
function getBrowserName() {
document.forms[0].elements[0].value =navigator.appName;
}
function getBrowserVersion() {
document.forms[0].elements[0].value = navigator.appVersion;
}
function getBrowserCodeName() {
document.forms[0].elements[0].value = navigator.appCodeName;
}
function getBrowserUserAgent() {
document.forms[0].elements[0].value = navigator.userAgent;
}
function getBrowserNameVersion() {
```

```
document.forms[0].elements[0].value = navigator.appName + " " +
navigator.appVersion;
}
// -->
</SCRIPT>
</HEAD>
<BODY >
<CENTER>
<FORM NAME="detect">
<INPUT TYPE="text" NAME="browser" SIZE=50 MAXLENGTH=50 VALUE=" Seus
dados
serão mostrados nesta janela ! ">
<BR><BR><BR>
<INPUT TYPE="button" VALUE="Nome do Navegador"
onClick="getBrowserName()">
<INPUT TYPE="button" VALUE="Versão do Navegador"
onClick="getBrowserVersion()">
<INPUT TYPE="button" VALUE="E-mail" onClick="getBrowserCodeName()">
<BR><BR>
<INPUT TYPE="button" VALUE="E-mail e versão"
onClick="getBrowserUserAgent()">
<BR> <BR>
<INPUT TYPE="button" VALUE="Nome e Versão"
onClick="getBrowserNameVersion()">
</FORM>
</BODY>
</HTML>
```

## Form

Os formulários têm muitas utilidades, uma das principais seria a transferência de dados dentro da própria página HTML, sem que para isso seja necessária a intervenção de qualquer outro meio externo.

Ao se criar um formulário na página HTML, automaticamente é criado uma referência para este formulário, que fica guardada na propriedade form do objeto document. Como você deve ter visto na página que trata do objeto document, a propriedade form é um vetor, e a cada formulário criado também é criado um novo elemento para este vetor, o índice inicial deste vetor é 0 (zero) e varia até o número de formulários do documento -1.

Como você pode notar cada formulário criado em uma página HTML, é considerado um objeto distinto, tendo suas próprias referências, métodos, propriedades e eventos associados. A forma de acessarmos diferenciadamente esses formulários dentro da página HTML é utilizar a propriedade form do objeto document.

Forma geral:

```
<FORM NAME="Nome"
[ACTION="URL"]
[METHOD="GET | POST"]
[onSubmit="evento"]>
```

Onde:

**Nome** = Nome do formulário, para futuras referências dentro da página HTML.

**URL** = Especifica o URL do servidor ao qual sera enviado o formulario.

**GET | POST** = metodos de transferencia de dados do browser para o servidor

Propriedades:

**action** - Especifica o URL do servidor ao qual sera enviado o formulario.

Ex: document.NomeDoFormulario.action

documet.GuestBook.action = [supnet@supnet.com.br](mailto:supnet@supnet.com.br)

**elements** - Vetor que armazena todos os objetos que são definidos dentro de um formulário (caixas de texto, botões, caixas de entrada de dados, checkboxes, botões de rádio). O número de elementos deste vetor varia de 0 (zero) até o número de objetos dentro do formulário -1.

Ex: document.NomeDoFormulario.elements[indice]

**method** - Seleciona um método para acessar o URL de ação. Os métodos são: get e post. Ambos os métodos transferem dados do browser para o servidor, com a seguinte diferença:

**method=get** - os dados de entrada são acrescentados ao endereço (URL) associado, como se fossem uma query (pesquisa a banco de dados) comum;

**method=post** - os dados não são acrescentados ao URL, mas fazem parte do corpo da mensagem enviada ao servidor.

*Obs.: O método mais usual é o post:*

Esta propriedade pode ser alterada, porém só surtirá efeito antes que o formulário seja mostrado na tela.

Ex: `document.NomeDoFormulario.method = "post" ( ou "get")`

Métodos:

**submit** - Transfere os dados do formulário para o endereço especificado em action, para serem processados. Funcionado de maneira análoga ao botão submit em HTML.

Ex: `document.NomeDoFormulario.submit( )`

Eventos:

**onSubmit** - Ocorre quando um botão do tipo **submit** recebe o clique do mouse, transferindo os dados de um formulário para o servidor especificado em action.

Os dados só são enviados se o evento receber um valor verdadeiro (true), valor este que pode ser conseguido como resultado a chamada de uma função que valida às informações do formulário.

Ex: `document.NomeDoFormulario.onSubmit ="return Valida_Informacoes(NomeDoFormulario)"`

Exemplo:

```
<HTML>
<HEAD>
<TITLE>Exemplo - Objeto Form</TITLE>
</HEAD>
<BODY>
<FORM action="mailto:esaex@canudos.ufba.br" method="POST">
<P><TT><B>Exemplo:</B></TT>
<P>Este exemplo demonstra a funcionalidade de um
formul&aacute;rio, para improvisar um "Guest Book"
<P>&nbsp;
Nome, Nascimento: <BR>
<INPUT TYPE="text" NAME="nomidade" VALUE=" " SIZE=70><BR>
Endere&ccedil;o: <BR>
<INPUT TYPE="text" NAME="endereco" VALUE=" " SIZE=70><BR>
E-Mail: <BR>
<INPUT TYPE="text" NAME="email" VALUE=" "
SIZE=70><BR>
Sua Home-Page: <BR>
<INPUT TYPE="text" NAME="hp" VALUE=" " SIZE=70><BR>
IRC: <BR>
<INPUT TYPE="text" NAME="irc"
VALUE=" " SIZE=70><BR>
Sugest&otilde;es, etc.: <BR>
<TEXTAREA NAME="sujestao" ROWS=7 COLS=70></TEXTAREA>
<P><CENTER><INPUT TYPE="submit" NAME="Submit"
```

```

    VALUE="Enviar"> <INPUT TYPE="reset" VALUE="Limpar"></CENTER>
</FORM>
<CENTER>
<FORM>
<INPUT TYPE="button" VALUE="Página Anterior" onClick="history.go(-1)">
<IMG SRC="S177.gif" WIDTH=540 HEIGHT=46 ALIGN=bottom><BR>
<FONT SIZE="-2">P&aacute;gina desenvolvida por </FONT><FONT
SIZE="-2"><A HREF="mailto:esaex@canudos.ufba.br">Anderson Barros
Torres</A></FONT><FONT SIZE="-2">. Julho/97</FONT>
</FORM>
</CENTER>
</BODY>
</HTML>

```

## CheckBox

Este objeto como o próprio nome sugere, exibe uma caixa de checagem igual às que encontramos no Windows, o funcionamento também é o mesmo: a condição de selecionada ou não, é alternada quando clicamos o mouse sobre o objeto, ou seja, se clicarmos sobre um objeto checkbox já marcado ele será automaticamente desmarcado, ao passo que se clicarmos em um objeto checkbox desmarcado ele será automaticamente marcado.

Forma geral:

```

<FORM>
<INPUT TYPE="checkbox" NAME="NomeDoObjeto" [CHECKED] VALUE="Label"
onClick="Ação">
</FORM>

```

Onde:

**Type** - Nome do objeto;

**Name** - Nome dado pelo programador, para futuras referências ao objeto;

**CHECKED** - Se especificado a CheckBox já vai aparecer selecionada;

**Value** - Define um rótulo para a CheckBox.

**onClick** - Define o que fazer quando dá-se um clique na CheckBox, fazendo com que o objeto CheckBox funcione como um objeto Button.

Propriedades:

**name** - Nome do objeto CheckBox em forma de string, da mesma forma como definido no campo Name utilizado na criação da CheckBox.

NomeDoObjeto.name // equivale a string "NomeDoObjeto"

**value** - Armazena o conteúdo do campo VALUE, definido na TAG.

NomeDoObjeto.value

**checked** - Retorna um valor lógico que depende do estado do objeto CheckBox

NomeDoObjeto.checked

// equivale a True se o objeto selecionado e False caso contrário

**defaultChecked** - Informa/Altera o estado default de um objeto CheckBox. Com relação a alteração, somente os objetos CheckBox ainda não exibidos podem ter seu estado default alterado.

```
NomeDoObjeto.defaultChecked
// equivalerá a True, se a cláusula CHECKED estiver presente e a
False caso contrário
```

Métodos:

**click**: este método simula um clique do mouse no objeto *CheckBox*, ou seja, executa um procedimento associado a uma *CheckBox* como se estivéssemos clicado na *CheckBox* mas sem que o usuário tenha realmente clicado.

```
Select01.click() // executaria uma função
```

Eventos associados:

**onClick**: Define o que fazer quando clicamos no objeto *CheckBox*

Exemplo:

```
<HTML>
<HEAD>
<TITLE>Exemplo CheckBox</TITLE>
<SCRIPT>
function exemplo(p1,p2,p3,p4) {
  alert('Veja os conteúdos das propriedades:
\nName=' + p1 +
'\nValue=' + p2 +
'\nChecked=' + p3 +
'\nDefaultChecked=' + p4);
}
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
<H3>Exemplo do objeto CheckBox</H3>
<HR>
<FORM>
<INPUT TYPE="checkbox" NAME="chb" VALUE="QQ COISA" CHECKED
onClick="exemplo(chb.name,chb.value,chb.checked,chb.defaultChecked)
">
Tecle aqui...
</FORM>
<BR><HR><BR>
Tecle no CheckBox para observar o funcionamento!!!
Para retornar clique o mouse <A HREF="history.go(-1)">AQUI</A>
</CENTER>
</BODY>
</HTML>
```

## Document

Este objeto armazena todas as características da página HTML, como por exemplo: cor das letras, cor de fundo, figura que aparecerá como papel de parede, etc. Sempre que incluímos alguma declaração no `<body>` do documento, estamos alterando o objeto *Document*.

Forma geral:

```
<body [background="imagem"]
[bgcolor="#cordefundo"]
[fgcolor="#cordotexto"]
[link="#cordoslinks"]
[alink="#cordolinkativado"]
[vlink="#cordolinkvisitado"]
[onload="função"]
[onunload="funcao"]>
```

Onde:

**Imagen** = figura no formato GIF, que servirá como papel de parede para a Home Page;

**#Cor...** = número (hexadecimal), com seis dígitos, que corresponde à cor no formato RGB, o "#" é obrigatório. Os dois primeiros dígitos correspondem a R (red), os dois do meio a G (green) e os dois últimos a B (blue). A combinação dos três forma a cor no formato RGB.

**função** = Nome de uma função pré-definida, que será chamada quando o evento ocorrer.

Propriedades:

**alinkColor** - Determina a cor do *link* enquanto o botão do mouse estiver pressionado sobre ele.

Ex: `document.alinkColor="#FFFFFF"`

**anchors** - Vetor que armazena as âncoras definidas em uma página HTML com o comando `<A NAME="ancora">`. Esta propriedade é somente para leitura, não pode ser alterada.

Ex: `document.anchors[índice]`

**bgColor** - Determina a cor de fundo da página HTML.

Ex: `document.bgColor="#000000"`

**cookie** - Os *cookies* são pequenos arquivos que alguns sites da Web gravam no computador dos visitantes. A idéia é identificar o usuário, anotar quais caminhos ele já percorreu dentro do site e permitir um controle mais eficaz dos espectadores.

**fgColor** - Determina a cor das letras em uma página HTML. Esta propriedade não altera o que já está impresso na página HTML.

Ex: `document.fgColor="#0000FF"`

**forms** - Vetor que armazena as referências aos formulários existentes na página HTML.

Esta propriedade é somente para leitura, não pode ser alterada.

Ex: `document.forms[índice]`

**lastModified** - Obtém a data da última atualização da página HTML. Esta propriedade é somente para leitura, não pode ser alterada.

Ex: `document.lastModified`

**linkColor** - Determina a cor dos *links* que ainda não foram visitados pelo usuário.

Ex: `document.linkColor = "#00FF00"`

**links** - Vetor que armazena os *links* definidos em uma página HTML. Esta propriedade é somente para leitura, não pode ser alterada.

Ex: `document.links[índice]`

**location** - Armazena o endereço (URL) atual em forma de *string*. Esta propriedade é somente para leitura, não pode ser alterada.

**referrer** - Armazena o endereço (URL) de quem chamou a página HTML atual. Com essa propriedade você pode saber como usuário chegou à sua página. Esta propriedade é somente para leitura, não pode ser alterada.

Ex: `document.referrer`

**title** - Armazena uma *string* com o título da página HTML atual. Esta propriedade é somente para leitura, não pode ser alterada.

Ex: `document.title`

**vlinkColor** - Determina a cor que o link aparecerá após ser visitado.

Ex: `document.vlinkColor = "#80FF80"`

Métodos:

**clear** - limpa a tela da janela atual.

Ex: `document.clear()`

**open** - Abre um documento e envia (mas não exibe) a saída dos métodos write/writeln. Os dados enviados são exibidos, quando é encontrado o método close.

Ex: `document.open()`

**close** - Termina uma seqüência iniciada com o método open, exibindo o que tinha sido apenas enviado.

Ex: `document.close()`

**write** - Imprime informações na página HTML.

Ex: `document.write("Qualquer coisa" [,variável] [,..., expressão])`

**writeln** - Imprime informações na página HTML e passa para a próxima linha. Em meus testes, esse método não apresentou diferença com relação ao método write.

Ex: `document.writeln("Qualquer coisa" [,variável] [,..., expressão])`

Eventos:

**onLoad** - Ocorre assim que um browser carrega uma página *HTML* ou *frame*.

Ex: <BODY ... onLoad='alert("Oi!!!!")'>

**onUnload** - Ocorre quando se abandona uma página *HTML* ou *frame*.

Ex: <BODY ... onUnload='alert("Tchau!!!!")'>

## Date

Objeto muito útil que retorna a data e hora do sistema no seguinte formato: *Dia da semana, Nome do mês, Dia do mês, Hora:Minuto:Segundo e Ano*. Como todo objeto, podem ser criadas novas instâncias para este objeto, essa prática possibilita a utilização de quantos objetos data você precisar.

Forma geral:

```
NovoObjeto = NEW date( )
```

Onde:

**NovoObjeto** = Objeto definido pelo usuário, para manipular datas.

Métodos:

**getMonth** - Obtém o número do mês. Retornando um valor entre 0 e 11. ( janeiro=0)

Ex: Mes=NovoObjeto.getMonth( )

**setMonth** - Estabelece um novo valor para o mês. O valor deve estar entre 0..11

Ex: NovoObjeto.setMonth(NumerodoMes)

**getDate** - Obtém o número do dia, considerando-se o mês. Retornando um valor numérico entre 1..31.

Ex: dia = NovoObjeto.getDate( )

**setDate** - Estabelece um novo valor para o dia do mês. Este valor deve estar entre 1..31

Ex: NovoObjeto.setDate(NumerodoDia)

**getDay** - Obtém o número do dia, considerando-se a semana. Retornando um valor numérico entre 0..6. Lembre-se de que a semana começa no domingo, logo 0, corresponde ao domingo.

Ex: DiaDaSemana = NovoObjeto.getDay( )

**getHours** - Obtém um número correspondente a hora. Retornando um valor numérico entre 0..23

Ex: Hora = NovoObjeto.getHours( )

**setHours** - Estabelece um novo valor para a hora. O valor deve estar entre 0..23

Ex: NovoObjeto.setHours(NovaHora)

**getMinutes** - Obtém um número correspondente aos minutos. Retornando um valor numérico entre 0..59

Ex: Minutos = NovoObjeto.getMinutes( )

**setMinutes** - Estabelece um novo valor para os minutos. O valor deve estar entre 0..59  
 Ex: NovoObjeto.setMinutes (Minutos)

**getSeconds** - Obtém um número correspondente aos segundos. Retornando um valor numérico entre 0..59  
 Ex: Segundos = NovoObjeto.getSeconds ( )

**setSeconds** - Estabelece um novo valor para os segundos. O valor deve estar entre 0..59  
 Ex: NovoObjeto.setSeconds (Segundos)

**getTime** - Obtém o tempo decorrido desde 01/01/70 até o presente momento. O único inconveniente é que esta data é dada em milissegundos.

Ex: TempoDecorrido=NovoObjeto.getTime ( )

**setTime** - Estabelece uma nova data.

Ex: DataDeNascimento=New Date ("August 2, 1970")

uma outra forma para definir a data seria:

```
OutraForma = New Date( )
OutraForma.setTime(DataDeNascimento.getTime( ))
```

**getTimezoneOffset** - Obtém a diferença entre o horário local e o horário do meridiano central (Greenwich). Este tempo é dado em minutos, logo, para saber o fuso-horário, deve-se dividir o resultado obtido por esta função por 60.

Ex: FusoHorário=NovoObjeto.getTimezoneOffset( ) / 60

**getYear** - Obtém um valor numérico correspondente ao ano.

Ex: Ano=NovoObjeto.getYear ( )

**setYear** - Estabelece um novo valor ao ano. O valor deve ser maior ou igual a 1900.

Ex: NovoObjeto.setYear(1997)

**toGMTstring** - Converte um objeto data para uma string seguindo o padrão Internet GMT.

Ex: NovoObjeto.toGMTstring( )

**toLocaleString** - Converte uma data para uma string seguindo o padrão local.

Ex: NovoObjeto.toLocaleString( )

Exemplo:

```
<HTML>
<HEAD>
<TITLE>Exemplo - Objeto Date</TITLE>
<SCRIPT>
<!--
var timerID = null;
var timerRunning = false;
function startclock ()
{
stopclock();
time();
}
function stopclock ()
{
```

```

if(timerRunning)
clearTimeout(timerID);
timerRunning = false;
}
function time ()
{
var now = new Date();
var yr = now.getYear();
var mName = now.getMonth() + 1;
var dName = now.getDay() + 1;
var dayNr = ((now.getDate()<10) ? "0" : "")+ now.getDate();
var ampm = (now.getHours() >= 12) ? " P.M." : " A.M."
var hours = now.getHours();
hours = ((hours > 12) ? hours - 12 : hours);
var minutes = ((now.getMinutes() < 10) ? ":0" : ":") +
now.getMinutes();
var seconds = ((now.getSeconds() < 10) ? ":0" : ":") +
now.getSeconds();
if(dName==1) Day = "Domingo";
if(dName==2) Day = "Segunda";
if(dName==3) Day = "Terça";
if(dName==4) Day = "Quarta";
if(dName==5) Day = "Quinta";
if(dName==6) Day = "Sexta";
if(dName==7) Day = "Sabado";
if(mName==1) Month="Janeiro";
if(mName==2) Month="Fevereiro";
if(mName==3) Month="Março";
if(mName==4) Month="Abril";
if(mName==5) Month="Maio";
if(mName==6) Month="Junho";
if(mName==7) Month="Julho";
if(mName==8) Month="Augosto";
if(mName==9) Month="Setembro";
if(mName==10) Month="Outubro";
if(mName==11) Month="Novembro";
if(mName==12) Month="Dezembro";
var DayDateTime=( " "
+ Day
+ ", "
+ dayNr
+ " de "
+ Month
+ " de "
+ ""
+ "19"
+ yr
+ ". Agora são:"
+ hours
+ minutes
+ seconds
+ " "
+ ampm
);
window.status=DayDateTime;

```

```
timerID = setTimeout("time()",1000);
timerRunning = true;
}
function clearStatus()
{
if(timerRunning)
clearTimeout(timerID);
timerRunning = false;
window.status=" ";
}
//-->
</SCRIPT>
</head>
<BODY BACKGROUND="b190.gif" onLoad="startclock ()">
<H1>Exemplo:</H1>
Demonstração do objeto Date, conforme visto na página anterior.
Funcionamento: a data e hora
ficam sendo mostradas no rodapé do browser.
<FORM>
<CENTER>
<BR>
<INPUT TYPE="button" VALUE="Página Anterior" onClick="history.go(-1)">
</CENTER>
</FORM>
<CENTER>
<IMG SRC="S177.GIF"><BR>
<H6>Página desenvolvida por <A href="mailto:webmaster@areainicial.zzn.com"> Fernando
Dercoli </A>. Julho/97</H6>
</CENTER>
</BODY>
</HTML>
```

## History

Este objeto armazena todas as *URL* das páginas *HTML* por onde o usuário passou durante a sessão atual do browser. É uma cópia das informações armazenadas na opção *Go* da barra de menu do *Navigator*.

Forma geral:

```
history.propriedade  
history.método
```

Propriedades:

**length** - Informa a quantidade de páginas visitadas.

Ex: `history.length`

Métodos:

**back** - Retorna à página anterior, de acordo com a relação de páginas do objeto *history*. Equivale a clicar o botão *back* do *browser*.

Ex: `history.back()`

**forward** - Passa para a próxima página, de acordo com a relação de páginas do objeto *history*. Equivale a clicar o botão *forward* do *browser*.

Ex: `history.forward()`

**go** - Permite que qualquer URL que esteja presente na relação de páginas visitadas do objeto *history*, seja carregada.

Ex: `history.go(parâmetro)`

Existem duas possibilidades para "parâmetro":

1 - parâmetro é um número: Ao definir um número, este deve ser inteiro. Se for positivo, a página alvo está "parâmetro" páginas à frente. Ao passo que se for negativo, a página alvo está "parâmetro" páginas para traz.

2 - parâmetro é uma string: Neste caso, o alvo é a URL que mais se assemelhe ao valor da string definida por "parâmetro".

## Window

É o objeto que ocupa o topo do esquema hierárquico em *JavaScript*.

Propriedades:

**defaultStatus** - Determina o conteúdo padrão da barra de status do browser, quando nada de importante estiver acontecendo.

Ex: `window.defaultStatus='Qualquer coisa'`

**frames** - Vetor que armazena as referências para as *frames* da janela atual.

Ex: `parent.frames.length`

// obtém o número de frames da janela principal, assumindo que estamos em uma *frame*.

**parent** - Refere-se a janela pai da *frame* atual.

**self** - Refere-se a janela atual.

Ex: `self.defaultStatus='Qualquer coisa'`

**status** - Define uma mensagem que irá aparecer no rodapé do browser, em substituição por exemplo, a *URL* de um *link*, quando estivermos com o mouse sobre o *link*.

Ex: `window.status="qualquer texto"`

**top** - Refere-se a janela de nível mais alto do esquema hierárquico do *JavaScript*.

Ex: `top.close() // fecha a janela principal do browser`

**window** - Refere-se a janela atual. Funciona de modo análogo a **self**.

Ex: `window.status='Qualquer coisa'`

Métodos:

**alert** - Mostra uma caixa de alerta, seguido de um sinal sonoro e o botão de *OK*.

Ex: `alert('Esta é uma janela de alerta!')`

**close** - Termina a sessão atual do browser.

Ex: `top.close()`

**confirm** - Mostra uma caixa de diálogo, seguida de um sinal sonoro e os botões de *OK* e *Cancel*. Retorna um valor verdadeiro se o usuário escolher *OK*.

Ex: `retorno=confirm('Deseja prosseguir?')`

**open** - Abre uma nova sessão do *browser*, como se o usuário pressionasse <CTRL>+N

Ex: `window.open("URL", "Nome" [, "características"])`

Onde:

**URL**: endereço selecionado inicialmente quando da abertura da nova janela.

**Nome**: nome da nova janela, definido pelo programador;

**Características** - série de opções de configuração da nova janela, se especificados devem estar na mesma *string*, separados por vírgulas e sem conter espaços:

`toolbar=0 ou 1`  
`location=0 ou 1`  
`directories=0 ou 1`

```

status=0 ou 1
menubar=0 ou 1
scrollbars=0 ou 1
resizable=0 ou 1
width=valor inteiro positivo
height=valor inteiro positivo
Ex:window.open("http://www.supnet.com.br/solucoes/web/website.php?s
=105",
"NovaJanela","toolbar=1,location=1,directories=0,status=1,menubar=1
,
scrollbars=1,resizable=0,width=320,height=240")

```

**prompt** - Monta uma caixa de entrada de dados, de forma simplificada comparando-se com o objeto *text*.

Ex: prompt(label [,valor])

onde:

**label** : texto que aparece ao lado da caixa;

**valor** : é o conteúdo inicial da caixa;\

**setTimeout** - Faz com que uma expressão seja avaliada após um determinado tempo (em milissegundos).

Ex: ID=setTimeout(alert('você chegou aqui, a 10 segundos'),10000)

**ID** - identificador utilizado para o cancelamento de *setTimeOut*

**clearTimeout** - Cancela *setTimeOut*.

Ex: clearTimeout (ID)

Eventos:

**onLoad** : Ocorre assim que a página *HTML* termina de ser carregada.

**onUnload** : Ocorre assim que o usuário sai da página atual.

## Reset

Este objeto restaura os campos de um formulário, para seus valores iniciais.

Forma geral:

```
<INPUT TYPE="reset" NAME="nome" VALUE="label" onClick="ação">
```

onde:

**reset** : Tipo do objeto

**nome** : Nome definido pelo programador, para futuras referências;

**label** : String que será mostrada na face do botão.

**ação** : Define o que fazer (além de sua função normal) quando clicamos no botão *reset*.

Propriedades:

**name** - Armazena o nome que foi definido pelo usuário, no campo *NAME*, para o objeto *reset*.

Ex: document.form[0].element[0].name

**value** - Armazena o texto que aparece na face do botão *reset*. Definido no campo *VALUE*.

Ex: document.form[0].element[0].value

Métodos:

**click** : simula um clique de mouse no botão *reset*, executando todas as funções a ele associadas, sem que no entanto o usuário tenha realmente clicado neste botão.

Ex: resetName.click()

Eventos:

**onClick** - Ocorre quando clicamos o mouse sobre o botão *reset*. Permite que associemos outra função ao botão *reset*.

Exemplo:

```
<HTML>
<HEAD>
<TITLE>Tutorial JavaScript - Exemplo: ResetButton</TITLE>
</HEAD>
<BODY>
<P><CENTER> <FONT SIZE="+3" FACE="Britannic Bold"
COLOR="#0000AF">J</FONT><FONT SIZE="+2" FACE="Britannic Bold"
COLOR="#0000AF">ava</FONT><FONT SIZE="+3" FACE="Britannic Bold"
COLOR="#0000AF">S</FONT><FONT SIZE="+2" FACE="Britannic Bold"
COLOR="#0000AF">cript -
</FONT><FONT SIZE="+3" FACE="Britannic Bold"
COLOR="#0000AF">G</FONT><FONT SIZE="+2"
FACE="Britannic Bold" COLOR="#0000AF">uia de </FONT><FONT SIZE="+3"
FACE="Britannic Bold"
COLOR="#0000AF">R</FONT><FONT SIZE="+2" FACE="Britannic Bold"
COLOR="#0000AF">efer&ecirc;ncia</FONT><FONT SIZE="+2"
FACE="Britannic Bold"><BR>
</FONT>
```

```

<FONT SIZE="-1" FACE="Britannic Bold" COLOR="#000080">&copy; 1997
Anderson Barros
Torres</FONT><BR><BR>
<B><I><FONT SIZE=+4 color=#000000 >E</FONT><FONT SIZE=+3
color=#000000
>xemplo</FONT></I></B>
</CENTER>
<BR><BR>
<FORM action="" method="POST">
<P>Digite o seu estilo musical:
<P><INPUT TYPE="text" NAME="estilo" VALUE="" SIZE=30> <INPUT
TYPE="RESET"
NAME="apaga" VALUE="Limpa" >
</FORM>
<BR> <BR> <BR>
<CENTER>
<FORM>
<INPUT TYPE="button" VALUE="Página Anterior" onClick="history.go(-
1)">
</FORM>

```

## Link

HTML permite ligações de uma região de texto (ou imagem) à um outro documento. Nestas páginas, temos visto exemplos dessas ligações: o browser destaca essas regiões e imagens do texto, indicando que são ligações de hipertexto - também chamadas *hypertext links* ou *hiperlinks* ou simplesmente *links*.

Forma geral:

```
<A [ NAME="ancora" ]
HREF="URL" [TARGET="janela"] [onClick="ação"] [onMouseOver="ação"]>
Texto explicativo</A>
```

onde:

**URL** : É o documento destino da ligação hipertexto;

**âncora** : É o texto ou imagem que servirá de ligação hipertexto .

**janela** : Nome da janela onde a página será carregada, para o caso de estarmos trabalhando com frames:

**"\_top"** : Se estivermos trabalhando com frames, a página referenciada pelo link, substituirá a página que criou as frames, ou seja, a página atual, com todas as frames, dará lugar a nova página.

**"\_self"** : A nova página é carregada na mesma janela do link.

**"\_blank"** : Abre uma nova seção do *browse* para carregar a página.

**ação** :Código de resposta ao evento.

**Texto explicativo** : Texto definido pelo usuário, que aparece na tela de forma destacada.

Eventos:

**onClick** - Ocorre quando clicamos o mouse sobre o link.

Ex: <A HREF="URL qualquer" onClick="alert('você teclou no link!')">Texto</A>

**onMouseOver** - Ocorre quando o mouse passa por cima do link, sem ser clicado.

Ex: <A HREF="URL qualquer" onMouseOver="self.status='Este texto aparecerá na barra de status quando o mouse estiver posicionado sobre o link'"> Texto</A>

Caminhos para o documento destino:

1. **Caminho relativo** : O caminho relativo pode ser usado sempre que queremos fazer referência a um documento que esteja no mesmo servidor do documento atual. Para usar links com caminhos relativos é preciso, portanto, conhecer a estrutura do diretório do servidor no qual estamos trabalhando. Simplificando, é como acessarmos um arquivo que esteja no mesmo diretório, não sendo necessário acrescentar o path.

2. **Caminho absoluto** : Utilizamos caminho absoluto quando desejamos referenciar um documento que esteja em outro servidor. Com a mesma sintaxe, é possível escrever *links* para qualquer servidor WWW no mundo. Seria a aplicação do endereço completo da página em questão, como visto no ítem anterior.

3. **Ligações a trechos de documentos** - Além do atributo *href*, que indica um documento destino de uma ligação hipertexto, o elemento A possui um atributo *NAME* que permite indicar um trecho de documento como ponto de chegada de uma ligação hipertexto. Funciona tanto para documentos locais como para os armazenados em outro servidor. O trecho deve ser indicado com o símbolo "#".

Ex: JavaInde.htm

```
<HTML>
...
<A HREF = "partel.htm#2">Diferença entre Java e JavaScript</A>
...
</HTML>
Ex: parte1.htm
<HTML>
...
<A NAME="1"></A>O que é JavaScript
JavaScript é...
<A NAME="2"></A>Diferença entre Java e JavaScript
A diferença é...
...
</HTML>
```

No exemplo acima, o link de *JavaInde.htm* carrega a página "*partel.htm*" e automaticamente posiciona o trecho "2" no topo da janela do browser.

## Palavras reservadas

Existem várias palavras que são reservadas para o *JavaScript* as quais são listadas abaixo. Essas palavras não podem ser utilizadas para identificar variáveis ou funções.

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

## Tabela de cores

Ao invés de especificar códigos hexadecimais para utilizar cor em documentos *HTML*, você pode simplesmente utilizar um literal, que especifica o nome da cor para obter o mesmo resultado. A seguir serão mostrados os vários literais que você pode utilizar na especificação de cores:

Exemplo de utilização:

```
Definindo Background <BODY BGCOLOR="literal">
Definindo a cor do texto <BODY TEXT="literal">
Definindo a cor de trechos de textos <FONT
COLOR="literal"...</FONT>
Links, Followed Links, etc. <BODY ALINK="literal"> etc.
```

Nome da cor	Base 10	Base 16
Snow	255250250	#FFFFAFA
GhostWhite	248248255	#F8F8FFF
WhiteSmoke	245245245	#F5F5F5
Gainsboro	220220220	#DCDCDC
FloralWhite	255250240	#FFFCAF0
OldLace	253245230	#FDF5E6
Linen	250240230	#FAF0E6
AntiqueWhite	250235215	#FAEBD7
PapayaWhip	255239213	#FFEFDD
BlanchedAlmon	255235205	#FFEBCD
Bisque	255228196	#FFE4C4
PeachPuff	255218185	#FFDAB9
NavajoWhite	255222173	#FFDEAD
Moccasin	255228181	#FFE4B5

Cornsilk	255248220	#FFF8DC
Ivory	255255240	#FFFFFF
LemonChiffon	255250205	#FFFACD
Seashell	255245238	#FFF5EE
Honeydew	240255240	#F0FFF0
MintCream	245255250	#F5FFFA
Azure	240255255	#F0FFFF
AliceBlue	240248255	#F0F8FF
Lavender	230230250	#E6E6FA
LavenderBlush	255240245	#FFF0F5
MistyRose	255228225	#FFE4E1
White	255255255	#FFFFFF
Black	000	#000000
DarkSlateGray	477979	#2F4F4F
DimGray	105105105	#696969
SlateGray	112128144	#708090
LightSlateGray	119136153	#778899
Gray	190190190	#BEBEBE
LightGray	211211211	#D3D3D3
MidnightBlue	2525112	#191970
NavyBlue	00128	#000080
CornflowerBlue	100149237	#6495ED
DarkSlateBlue	7261139	#483D8B
SlateBlue	10690205	#6A5ACD
MediumSlateBlue	123104238	#7B68EE
LightSlateBlue	132112255	#8470FF
MediumBlue	00205	#0000CD
RoyalBlue	65105225	#4169E1
Blue	00255	#0000FF
DodgerBlue	30144255	#1E90FF
DeepSkyBlue	0191255	#00BFFF
SkyBlue	135206235	#87CEEB
LightSkyBlue	135206250	#87CEFA
SteelBlue	70130180	#4682B4
LightSteelBlue	176196222	#B0C4DE
LightBlue	173216230	#ADD8E6
PowderBlue	176224230	#B0E0E6
PaleTurquoise	175238238	#AFEEEE
DarkTurquoise		#00CED1
MediumTurquoise		#48D1CC
Turquoise	64224208	#40E0D0
Cyan	0255255	#00FFFF
LightCyan	224255255	#E0FFFF
CadetBlue	95158160	#5F9EA0
MediumAquamarine	102205170	#66CDAA
Aquamarine	127255212	#7FFFAD
DarkGreen	01000	#006400
DarkOliveGreen	8510747	#556B2F
DarkSeaGreen	143188143	#8FBBC8F
SeaGreen	4613987	#2E8B57
MediumSeaGreen	60179113	#3CB371

LightSeaGreen	32178170	#20B2AA
PaleGreen	152251152	#98FB98
SpringGreen	0255127	#00FF7F
LawnGreen	1242520	#7CFC00
Green	02550	#00FF00
Chartreuse	1272550	#7FFF00
MediumSpringGreen	0250154	#00FA9A
GreenYellow	17325547	#ADFF2F
LimeGreen	5020550	#32CD32
YellowGreen	15420550	#9ACD32
ForestGreen	3413934	#228B22
OliveDrab	10714235	#6B8E23
DarkKhaki	189183107	#BDB76B
PaleGoldenrod	238232170	#EEE8AA
LtGoldenrodYellow	250250210	#FAFAD2
LightYellow	255255224	#FFFFE0
Yellow	2552550	#FFFF00
Gold	2552150	#FFD700
LightGoldenrod	238221130	#EEDD82
Goldenrod	21816532	#DAA520
DarkGoldenrod	18413411	#B8860B
RosyBrown	188143143	#BC8F8F
IndianRed	2059292	#CD5C5C
SaddleBrown	1396919	#8B4513
Sienna	1608245	#A0522D
Peru	20513363	#CD853F
Burlywood	222184135	#DEB887
Beige	245245220	#F5F5DC
Wheat	245222179	#F5DEB3
SandyBrown	24416496	#F4A460
Tan	210180140	#D2B48C
Chocolate	21010530	#D2691E
Firebrick	1783434	#B22222
Brown	1654242	#A52A2A
DarkSalmon	233150122	#E9967A
Salmon	250128114	#FA8072
LightSalmon	255160122	#FFA07A
Orange	2551650	#FFA500
DarkOrange	2551400	#FF8C00
Coral	25512780	#FF7F50
LightCoral	240128128	#F08080
Tomato	2559971	#FF6347
OrangeRed	255690	#FF4500
Red	25500	#FF0000
HotPink	255105180	#FF69B4
DeepPink	25520147	#FF1493
Pink	255192203	#FFC0CB
LightPink	255182193	#FFB6C1
PaleVioletRed	219112147	#DB7093
Maroon	1764896	#B03060
MediumVioletRed	19921133	#C71585

VioletRed	20832144	#D02090
Magenta	2550255	#FF00FF
Violet	238130238	#EE82EE
Plum	221160221	#DDA0DD
Orchid	218112214	#DA70D6
MediumOrchid	18685211	#BA55D3
DarkOrchid	15350204	#9932CC
DarkViolet	1480211	#9400D3
BlueViolet	13843226	#8A2BE2
Purple	16032240	#A020F0
MediumPurple	147112219	#9370DB
Thistle	216191216	#D8BFDB
Snow1	255250250	#FFFFFA
Snow2	238233233	#EEE9E9
Snow3	205201201	#CDC9C9
Snow4	139137137	#8B8989
Seashell1	255245238	#FFF5EE
Seashell2	238229222	#EEE5DE
Seashell3	205197191	#CDC5BF
Seashell4	139134130	#8B8682
AntiqueWhite1	255239219	#8B8682
AntiqueWhite2	238223204	#EEDFCC
AntiqueWhite3	205192176	#CDC0B0
AntiqueWhite4	139131120	#8B8378
Bisque1	255228196	#FFE4C4
Bisque2	238213183	#EED5B7
Bisque3	205183158	#CDB79E
Bisque4	139125107	#8B7D6B
PeachPuff1	255218185	#FFDAB9
PeachPuff2	238203173	#EECBAD
PeachPuff3	205175149	#CDAF95
PeachPuff4	139119101	#8B7765
NavajoWhite1	255222173	#FFDEAD
NavajoWhite2	238207161	#EECFA1
NavajoWhite3	205179139	#CDB38B
NavajoWhite4	139121194	#8B795E
LemonChiffon1	255250205	#FFFACD
LemonChiffon2	238233191	#EEE9BF
LemonChiffon3	205201165	#CDC9A5
LemonChiffon4	139137112	#8B8970
Cornsilk1	255248220	#FFF8DC
Cornsilk2	238232205	#EEE8CD
Cornsilk3	205200177	#CDC8B1
Cornsilk4	139136120	#8B8878
Ivory1	255255240	#FFFFFF
Ivory2	238238224	#EEEEEE
Ivory3	205205193	#CDCDC1
Ivory4	139139131	#8B8B83
Honeydew1	240255240	#F0FFF0
Honeydew2	224238224	#E0EEEE
Honeydew3	193205193	#C1CDC1

Honeydew4	131139131	#838B83
LavenderBlush1	255240245	#FFF0F5
LavenderBlush2	238224229	#EEE0E5
LavenderBlush3	205193197	#CDC1C5
LavenderBlush4	139131134	#8B8386
MistyRose1	255228225	#FFE4E1
MistyRose2	238213210	#EED5D2
MistyRose3	205183181	#CDB7B5
MistyRose4	139125123	#8B7D7B
Azure1	240255255	#F0FFFF
Azure2	224238238	#E0EEEE
Azure3	193205205	#C1CDCC
Azure4	131139139	#838B8B
SlateBlue1	131111255	#836FFF
SlateBlue2	122103238	#7A67EE
SlateBlue3	10589205	#6959CD
SlateBlue4	7160139	#473C8B
RoyalBlue1	72118255	#4876FF
RoyalBlue2	67110238	#436EEE
RoyalBlue3	5895205	#3A5FC
RoyalBlue4	3964139	#27408B
Blue1	00255	#0000FF
Blue2	00238	#0000EE
Blue3	00205	#0000CD
Blue4	00139	#00008B
DodgerBlue1	30144255	#1E90FF
DodgerBlue2	28134238	#1C86EE
DodgerBlue3	24116205	#1874CD
DodgerBlue4	1678139	#104E8B
SteelBlue1	99184255	#63B8FF
SteelBlue2	92172238	#5CACCE
SteelBlue3	79148205	#4F94CD
SteelBlue4	54100139	#36648B
DeepSkyBlue1	0191255	#00BFFF
DeepSkyBlue2	0178238	#00B2EE
DeepSkyBlue3	0154205	#009ACD
DeepSkyBlue4	0104139	#00688B
SkyBlue1	135206255	#87CEFF
SkyBlue2	126192238	#7EC0EE
SkyBlue3	108166205	#6CA6CD
SkyBlue4	74112139	#4A708B
LightSkyBlue1	176226255	#B0E2FF
LightSkyBlue2	164211238	#A4D3EE
LightSkyBlue3	141182205	#8DB6CD
LightSkyBlue4	96123139	#607B8B
SlateGray1	198226255	#C6E2FF
SlateGray2	185211238	#B9D3EE
SlateGray3	159182205	#9FB6CD
SlateGray4	108123139	#6C7B8B
LightSteelBlue1	202225255	#CAE1FF
LightSteelBlue2	188210238	#BCD2EE

LightSteelBlue3	162181205	#A2B5CD
LightSteelBlue4	110123139	#6E7B8B
LightBlue1	191239255	#BFEFFF
LightBlue2	178223238	#B2DFEE
LightBlue3	154192205	#9AC0CD
LightBlue4	104131139	#68838B
LightCyan1	224255255	#E0FFFF
LightCyan2	209238238	#D1EEEE
LightCyan3	180205205	#B4CDCD
LightCyan4	122139139	#7A8B8B
PaleTurquoise1	187255255	#BBFFFF
PaleTurquoise2	174238238	#AEEEEEE
PaleTurquoise3	150205205	#96CDCD
PaleTurquoise4	102139139	#668B8B
CadetBlue1	152245255	#98F5FF
CadetBlue2	142229238	#8EE5EE
CadetBlue3	122197205	#7AC5CD
CadetBlue4	83134139	#53868B
Turquoise1	0245255	#00F5FF
Turquoise2	0229238	#00E5EE
Turquoise3	0197205	#00C5CD
Turquoise4	0134139	#00868B
Cyan1	0255255	#00FFFF
Cyan2	0238238	#00EEEE
Cyan3	0205205	#00CDCD
Cyan4	0139139	#008B8B
DarkSlateGray1	151255255	#97FFFF
DarkSlateGray2	141238238	#8DEEEE
DarkSlateGray3	121205205	#79CDCD
DarkSlateGray4	82139139	#528B8B
Aquamarine1	127255212	#7FFF4
Aquamarine2	118238198	#76EEC6
Aquamarine3	102205170	#66CDA
Aquamarine4	69139116	#458B74
DarkSeaGreen1	193255193	#C1FFC1
DarkSeaGreen2	180238180	#B4EEB4
DarkSeaGreen3	155205155	#9BCD9B
DarkSeaGreen4	105139105	#698B69
SeaGreen1	84255159	#54FF9F
SeaGreen2	78238148	#4EEE94
SeaGreen3	67205128	#43CD80
SeaGreen4	4613987	#2E8B57
PaleGreen1	154255154	#9AFF9A
PaleGreen2	144238144	#90EE90
PaleGreen3	124205124	#7CCD7C
PaleGreen4	8413984	#548B54
SpringGreen1	0255127	#00FF7F
SpringGreen2	0238118	#00EE76
SpringGreen3	0205102	#00CD66
SpringGreen4	013969	#008B45
Green1	02550	#00FF00

Green2	02380	#00EE00
Green3	02050	#00CD00
Green4	01390	#008B00
Chartreuse1	1272550	#7FFF00
Chartreuse2	1182380	#76EE00
Chartreuse3	1022050	#66CD00
Chartreuse4	691390	#458B00
OliveDrab1	19225562	#C0FF3E
OliveDrab2	17923858	#B3EE3A
OliveDrab3	15420550	#9ACD32
OliveDrab4	10513934	#698B22
DarkOliveGreen1		#CAFF70
DarkOliveGreen2		#BCEE68
DarkOliveGreen3		#A2CD5A
DarkOliveGreen4	11013961	#6E8B3D
Khaki1	255246143	#FFF68F
Khaki2	238230133	#EEE685
Khaki3	205198115	#CDC673
Khaki4	13913478	#8B864E
LightGoldenrod1	255236139	#FFEC8B
LightGoldenrod2	238220130	#EEDC82
LightGoldenrod3	205190112	#CDBE70
LightGoldenrod4	13912976	#8B814C
LightYellow1	255255224	#FFFFE0
LightYellow2	238238209	#EEEED1
LightYellow3	205205180	#CDCDB4
LightYellow4	139139122	#8B8B7A
Yellow1	2552550	#FFFF00
Yellow2	2382380	#EEEE00
Yellow3	2052050	#CDCD00
Yellow4	1391390	#8B8B00
Gold1	2552150	#FFD700
Gold2	2382010	#EEC900
Gold3	2051730	#CDAD00
Gold4	1391170	#8B7500
Goldenrod1	25519337	#FFC125
Goldenrod2	23818034	#EEB422
Goldenrod3	20515529	#CD9B1D
Goldenrod4	13910520	#8B6914
DarkGoldenrod1	25518515	#FFB90F
DarkGoldenrod2	23817314	#EEAD0E
DarkGoldenrod3	20514912	#CD950C
RosyBrown1	255193193	#FFC1C1
RosyBrown2	238180180	#EEB4B4
RosyBrown3	205155155	#CD9B9B
RosyBrown4	139105105	#8B6969
IndianRed1	255106106	#FF6A6A
IndianRed2	2389999	#EE6363
IndianRed3	2058585	#CD5555
IndianRed4	1395858	#8B3A3A
Sienna1	25513071	#FF8247

Sienna2	23812166	#EE7942
Sienna3	20510457	#CD6839
Sienna4	1397138	#8B4726
Burlywood1	255211155	#FFD39B
Burlywood2	238197145	#EEC591
Burlywood3	205170125	#CDAA7D
Burlywood4	13911585	#8B7355
Wheat1	255231186	#FFE7BA
Wheat2	238216174	#EED8AE
Wheat3	205186150	#CDBA96
Wheat4	139126102	#8B7E66
Tan1	25516579	#FFA54F
Tan2	23815473	#EE9A49
Tan3	20513363	#CD853F
Tan4	1399043	#8B5A2B
Chocolate1	25512736	#FF7F24
Chocolate2	23811833	#EE7621
Chocolate3	20510229	#CD661D
Chocolate4	1396919	#8B4513
Firebrick1	2554848	#FF3030
Firebrick2	2384444	#EE2C2C
Firebrick3	2053838	#CD2626
Firebrick4	1392626	#8B1A1A
Brown1	2556464	#FF4040
Brown2	2385959	#EE3B3B
Brown3	2055151	#CD3333
Brown4	1393535	#8B2323
Salmon1	255140105	#FF8C69
Salmon2	23813098	#EE8262
Salmon3	20511284	#CD7054
Salmon4	1397657	#8B4C39
LightSalmon1	255160122	#FFA07A
LightSalmon2	238149114	#EE9572
LightSalmon3	20512998	#CD8162
LightSalmon4	1398766	#8B5742
Orange1	2551650	#FFA500
Orange2	2381540	#EE9A00
Orange3	2051330	#CD8500
Orange4	139900	#8B5A00
DarkOrange1	2551270	#FF7F00
DarkOrange2	2381180	#EE7600
DarkOrange3	2051020	#CD6600
DarkOrange4	139690	#8B4500
Coral1	25511486	#FF7256
Coral2	23810680	#EE6A50
Coral3	2059169	#CD5B45
Coral4	1396247	#8B3E2F
Tomato1	2559971	#FF6347
Tomato2	2389266	#EE5C42
Tomato3	2057957	#CD4F39
Tomato4	1395438	#8B3626

OrangeRed1	255690	#FF4500
OrangeRed2	238640	#EE4000
OrangeRed3	205550	#CD3700
OrangeRed4	139370	#8B2500
Red1	25500	#FF0000
Red2	23800	#EE0000
Red3	20500	#CD0000
Red4	13900	#8B0000
DeepPink1	25520147	#FF1493
DeepPink2	23818137	#EE1289
DeepPink3	20516118	#CD1076
DeepPink4	1391080	#8B0A50
HotPink1	255110180	#FF6EB4
HotPink2	238106167	#EE6AA7
HotPink3	20596144	#CD6090
HotPink4	1395898	#8B3A62
Pink1	255181197	#FFB5C5
Pink2	238169184	#EEA9B8
Pink3	205145158	#CD919E
Pink4	13999108	#8B636C
Light Pink1	255174185	#FFAEB9
LightPink2	238162173	#EEA2AD
LightPink3	205140149	#CD8C95
LightPink4	13995101	#8B5F65
PaleVioletRed1	255130171	#FF82AB
PaleVioletRed2	238121159	#EE799F
PaleVioletRed3	205104137	#CD6889
PaleVioletRed4	1397193	#8B475D
Maroon1	25552179	#FF34B3
Maroon2	23848167	#EE30A7
Maroon3	20541144	#CD2990
Maroon4	1392898	#8B1C62
VioletRed1	25562150	#FF3E96
VioletRed2	23858140	#EE3A8C
VioletRed3	20550120	#CD3278
VioletRed4	1393482	#8B2252
Magenta1	2550255	#FF00FF
Magenta2	2380238	#EE00EE
Magenta3	2050205	#CD00CD
Magenta4	1390139	#8B008B
Orchid1	255131250	#FF83FA
Orchid2	238122233	#EE7AE9
Orchid3	205105201	#CD69C9
Orchid4	13971137	#8B4789
Plum1	255187255	#FFBBFF
Plum2	238174238	#EEAEEE
Plum3	205150205	#CD96CD
Plum4	139102139	#8B668B
MediumOrchid1	224102255	#E066FF
MediumOrchid2	20995238	#D15FEE
MediumOrchid3	18082205	#B452CD

MediumOrchid4	12255139	#7A378B
DarkOrchid1	19162255	#BF3EFF
DarkOrchid2	17858238	#B23AEE
DarkOrchid3	15450205	#9A32CD
DarkOrchid4	10434139	#68228B
Purple1	15548255	#9B30FF
Purple2	14544238	#912CEE
Purple3	12538205	#7D26CD
Purple4	8526139	#551A8B
MediumPurple1	171130255	#AB82FF
MediumPurple2	159121238	#9F79EE
MediumPurple3	137104205	#8968CD
MediumPurple4	9371139	#5D478B
Thistle1	255225255	#FFE1FF
Thistle2	238210238	#EED2EE
Thistle3	205181205	#CDB5CD
Thistle4	139123139	#8B7B8B
Gray11	282828	#1C1C1C
Gray21	545454	#363636
Gray31	797979	#4F4F4F
Gray41	105105105	#696969
Gray51	130130130	#828282
Gray61	156156156	#9C9C9C
Gray71	181181181	#B5B5B5
Gray81	207207207	#CFCFCF
Gray91	232232232	#E8E8E8
DarkGray	169169169	#A9A9A9
DarkBlue	00139	#00008B
DarkCyan	0139139	#008B8B
DarkMagenta	1390139	#8B008B
DarkRed	13900	#8B0000
LightGreen	144238144	#90EE90

## Referências

Consórcio World Wide Web (W3C), comunidade internacional que desenvolve padrões com o objetivo de garantir o crescimento da web.

<http://www.w3c.br/Home/WebHome>

## Tabela de cores

Universidade Federal do Pará

<http://www.ufpa.br/dicas/htm/htm-cor2.htm>