

JAVASCRIPT



ProWay
IT Training

Todos os direitos reservados.

Proibida a reprodução, mesmo parcial, por qualquer processo, seja mecânico, eletrônico, fotocópia, gravação, digitalização ou outros, sem a prévia autorização.



ProWay Informática Ltda

Rua 7 de Setembro, 1213 – CEP 89010-203

Shopping Center Neumarkt – Blumenau - SC

(47) 3322-3344

www.proway.com.br

Índice

Visão Geral	03
Lição 1	04
1. Conhecendo JavaScript	04
2. Sobre o curso	05
3. Primeiro Script	06
4. Escondendo o JavaScript	08
5. Revisão	09
Lição 2	10
1. Variáveis	10
2. Mostrando valores no corpo	12
3. Strings	13
4. Exemplo com variáveis e strings	17
5. Condições "If-then"	19
6. Exemplos com "If-then"	20
7. Eventos de link	24
8. Troca de Imagens	25
9. Revisão	26
Lição 3	27
1. Modelo de Documento de Objeto JavaScript, Janelas e Molduras	27
2. Introdução a manipulação de Janelas	28
3. Manipulação de Janelas utilizando JavaScript	29
4. Exemplos de janelas abertas com JavaScript	30
5. Características das Janelas	31
6. O Modelo de Documento de Objeto JavaScript	33
7. Modificando a Barra de Status	34
8. Comunicação entre Janelas	35
9. Hierarquia do MDO	36
10. Utilizando o MDO em outras janelas	37
11. Trabalhando com molduras	39
12. Janelas e Hierarquia de molduras	40
13. Revisão	41

Lição 4	42
1. Introdução a loops	43
2. Loops for	47
3. Loops aninhados	48
4. Exemplo de loops	49
5. Arrays	50
6. Arrays e loops	51
7. Arrays e MDO	53
8. Funções	54
9. Funções sem parâmetro	55
10. Parâmetros e valores retornados	57
11. Funções com mais de um parâmetro	59
12. Revisão	61

Lição 5	62
1. Introdução a formulários	62
2. Manipulando Valores de campos de texto	63
3. Eventos de campo de texto	65
4. Eventos de Formulários	67
5. Checkboxes	68
6. Botões de radio	70
7. Selects	71
8. Métodos de formulário onChange em selects	73
9. Revisão	75

Visão Geral

O Javascript é uma das peças mais importantes no desenvolvimento de um website consistente e interativo. Com ele, você pode executar dos comportamentos mais simples, como trocas de imagens ao passar do mouse, aos cálculos mais abrangentes, isso sem a necessidade de arquivos CGI externos.

Sua aceitação também é uma grande vantagem: os principais browsers já apartir da versão 3.0, interpretam JavaScript sem maiores problemas (até mesmo o Netscape 2.0 consegue interpretar alguma coisa).

Este curso foi desenvolvido para iniciantes, focando ensinar tudo que você precisa para começar sua carreira com esta linguagem. Já na primeira lição você será capaz de fazer seu primeiro script, bem como exemplos em todas as próximas.

Começaremos com uma visão geral sobre JavaScript, incluindo variáveis, indicações if - then (se - então), eventos de links, trocas de imagens, Modelo de documento de objeto (MDO), janelas e molduras, Sintáxe JavaScript com loops, arrays, funções e formulários.

Lição 1

1. Conhecendo JavaScript

Interação. A grande maioria dos sites que se julgam interativos na verdade apenas permitem que você clique em links de uma página nova para outra. Mesmo páginas com scripts CGI não parecem realmente interativas, com formulários onde você envia informações e tem que esperar por alguma resposta.

Para melhorar este quadro, nós temos o JavaScript, que oferece interações em tempo real, como imagens que trocam de acordo com ações do usuário, elementos que podem influenciar uns aos outros e cálculos que podem ser feitos sem CGIs externos. Tudo sem precisar esperar resposta de servidor.

Outra grande vantagem em trabalhar com JavaScript é que ele não exige de você uma configuração avançada para desenvolvimento. Tudo pode ser feito no seu computador, com um editor de textos simples (como o notepad do windows) e um browser, sem mesmo ter uma conexão.

Mas mesmo sendo simples para se trabalhar, o JavaScript ainda é uma linguagem de programação completa. Se você quiser depois estudar algum outro tipo de programação, como Perl, C, C++, ou Java, o JavaScript é uma introdução perfeita.

2. Sobre o curso

Este curso tem como objetivo o desenvolvimento de JavaScripts úteis imediatamente. Estas são algumas dicas que podemos oferecer para iniciantes:

- Antes de mais nada: JavaScript **não** é JAVA.
- Alguns browsers interpretam JavaScript de maneiras diferentes. Portanto, sempre que você estiver desenvolvendo site que utilizam JavaScript, teste ele no maior número de browsers e plataformas diferentes e de versões diferentes possíveis, para ver suas reações.
- Este tutorial não tem a pretensão de ensinar tudo sobre JavaScript, mas sim de fazer uma introdução completa e habilitar o estudante a entender sua sintaxe para estar apto a estudar JavaScript Avançado.
- Se você se perguntar como algum site faz alguma coisa, lembre-se que você pode sempre ver o código e tentar aprender com ele, já que JavaScript, ao contrário de scripts CGIs, acontece na máquina do usuário, não no servidor web.
- A melhor maneira de estudar é testando. Durante o curso você vai ter alguns exemplos para testar, mas sinta-se a vontade para experimentar outras idéias.

3. Primeiro Script

Para começar, experimente este código num HTML:

```
<HTML>
<head>
<title>Página de Teste utilizando JavaScript</title>

<script language="JavaScript">
  //Um Alert box para testar
  alert("Olá mundo!");
</script>

</head>
<body>

corpo do site

</body>
</html>
```

Com este código, execute o arquivo no browser e você receberá uma mensagem como esta:



Vamos analisar o código:

O JavaScript geralmente é colocado entre as tags de `</title>` e `</head>`, no cabeçalho dos HTMLs.

Assim como o HTML, JavaScript é apenas texto que pode ser digitado em qualquer processador de textos. Mesmo que a maioria dos JavaScripts estejam sempre no cabeçalho, você também pode trabalhar com ele no corpo das páginas.

No começo do código, nós temos a tag `<script language="JavaScript">`, indicando o começo do Script. A princípio, a linguagem padrão de Script para todos os browsers é JavaScript, e por esse motivo você não precisaria definir em `language="JavaScript"`, mas sempre existe a possibilidade de isso mudar, então acrescentar essa informação pode ser uma boa idéia.

Tudo que estiver entre `//` e o final da linha é **comentário**, e será ignorado pela interpretação do browser. A prática de comentar seus scripts é sempre muito bem-vinda, pois algum amigo/colega de trabalho/funcionário pode precisar entendê-los, ou até mesmo você, daqui a alguns meses, quando não se lembrar mais sobre o que eles tratam.

Para comentar blocos grande de texto, você pode utilizar seu texto dentro de `/*` e `*/`, assim:

```
/* este é um  
bloco grande texto  
comentado que eu  
precisei fazer */
```

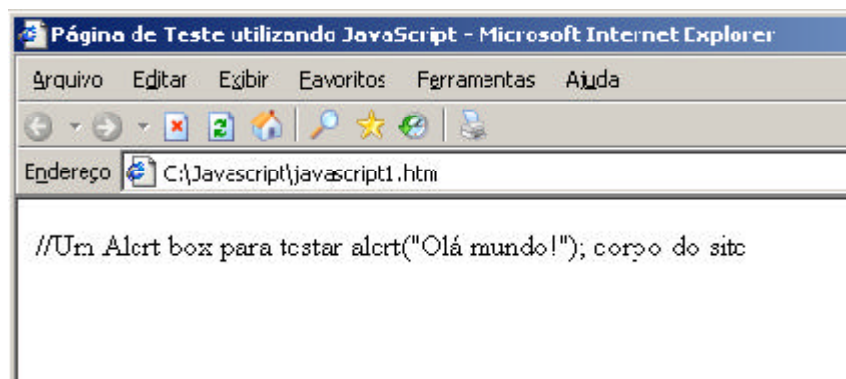
Na linha onde diz `alert("Olá mundo!");` você está chamando uma caixa de alerta simples do browser com o texto "Olá mundo!".

Este é o seu primeiro Script: O método de `alert`. Veremos mais sobre isso, mas por hora o que você precisa saber é que as linhas de comandos devem acabar sempre com um ponto e vírgula (`;`) e que texto deve estar sempre entre Aspas.

E o seu Script acaba, como um simples HTML, com uma tag de `</script>`.

4. Escondendo o JavaScript

O problema do exemplo anterior é que browsers antigos não entendem a tag `<script>`. Estes browsers antigos tratarão seu script como HTML comum e você terá uma página com o texto do script no corpo do browser, dessa forma:



Como esse não é o resultado esperado, nós temos que de alguma forma esconder o JavaScript destes browsers antigos para não haver interpretação errada. Para isso, nós podemos utilizar os comentários normais de HTML, desta forma:

```
<HTML>
<head>
<title>Página de Teste utilizando JavaScript</title>

<script language="JavaScript">
<!--
    //Um Alert box para testar
    alert("Olá mundo!");
-->
</script>

</head>
<body>

corpo do site

</body>
</html>
```

Dessa forma, os browsers antigos vão ignorar a existência deste código, enquanto os browsers novos, que sabem interpretar scripts, executarão ele e exibirão a caixa de diálogo com a mensagem de alerta.

5. Revisão

Nós vimos nesta lição:

- A importância do JavaScript
- Os problemas de compatibilidade de browsers
- Onde vai o JavaScript no HTML
- Os Comentários
- Como esconder seus Scripts de browsers antigos
- Como funciona uma caixa de diálogo de alerta

Lição 2

1. Variáveis

Agora que você já sabe onde o JavaScript vai no HTML e como ele se parece, vamos começar a entender a linguagem. Nessa Lição você vai aprender como o JavaScript armazena informações, faz decisões baseadas naquelas informações e como ele muda imagens baseado em interação com o usuário.

As **Variáveis** são a maneira simples que o JavaScript tem de armazenar informações. Por exemplo, se você escrever em algum lugar “ $x=2$ ”, x passa a ser a variável que contém o valor 2. Então se você disser que “ $y=x+3$ ”, você terá um valor 5 associado à “ y ”.

Aqui vai um exemplo de JavaScript que utiliza variáveis.

```
<script language="JavaScript">
<!--
```

Essas duas primeiras linhas você já conhece, que são o necessário para começar qualquer JavaScript.

```
// Carrega algumas variáveis
var segundos_por_minuto = 60;
var minutos_por_hora = 60;
var horas_por_dia = 24;
var dias_por_ano = 365;
```

A primeira linha aqui está comentada. Esse comentário pode parecer óbvio, mas é uma boa prática separar um bloco de variáveis do resto do JavaScript para organização.

As linhas seguintes são declarações de variáveis. Note que estamos utilizando a palavra `var` antes de declarar as variáveis. Isto não é estritamente necessário, mas veremos mais tarde por que é uma boa prática.

As variáveis devem começar sempre com uma letra ou um *underscore*. Se você iniciar com um número ou algum outro caractere, seu JavaScript provavelmente não vai funcionar.

Após o primeiro caractere (letra ou *underscore*), você pode adicionar números.

Variáveis são sensíveis a letras maiúsculas e minúsculas, ou seja, se você declarar que `Mins = 60`, isso não quer dizer que `MINS` ou `mins` também sejam iguais a 60. Por esse motivo, é uma boa idéia escolher uma regra de utilização para nomes de variáveis (por exemplo, sempre em minúsculas, ou sempre separadas por *underscores*) e sempre segui-la, para facilitar o desenvolvimento.

As variáveis devem descrever o que elas são. Isso quer dizer, se você for definir uma variável com o nome de “ x ”, fica difícil para quem quer entender o que ela faz descobrir seu significado. Não é necessário fazer variáveis longas a ponto de demorar para digitá-las, mas longas o bastante para serem intuitivas.

Você pode dar um valor à uma variável assim que a declara, ou você pode esperar até saber qual vai ser o valor.

Todas as linhas de comando devem acabar com ponto e vírgula. O Ponto e vírgula funciona como a pontuação do JavaScript, que diz onde um comando acaba para que outro possa começar, já que o JavaScript

ignora espaços e quebras de linhas. O layout que se aplica (colocando cada comando em uma linha diferente e alinhando eles com espaçamento) serve somente para facilitar a leitura e compreensão do código, pois não influencia em nada no seu funcionamento. Você poderia fazer todos os seus JavaScripts em uma única linha, com um comando depois do outro, mas seria praticamente impossível de lê-lo.

Para falar a verdade, existem casos em que você não precisa necessariamente adicionar um ponto e vírgula no final da linha. Você poderá ver casos em que ele não é utilizado. Mas é sempre uma boa prática aplicá-lo, pois além de deixar o programa mais legível, você corre menos riscos de adicionar algum código numa linha seguinte e estragar o programa.

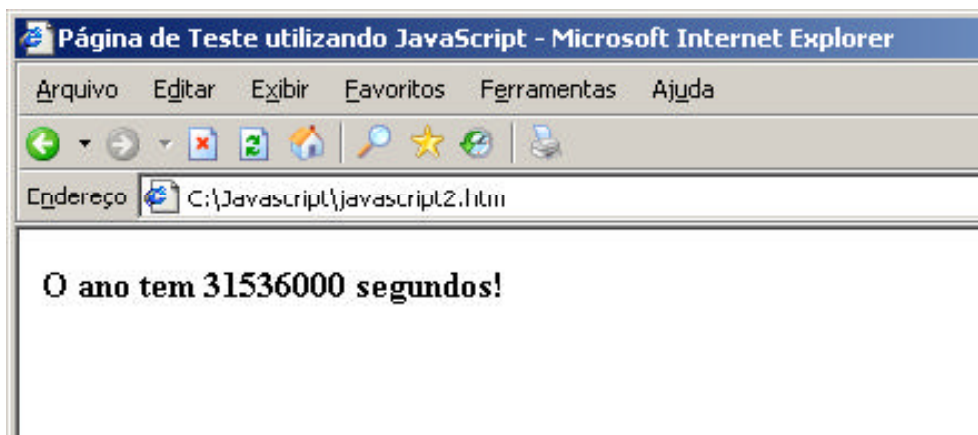
```
// faz alguns cálculos
var segundos_por_dia = segundos_por_minuto * minutos_por_hora * horas_por_dia;
var segundos_por_ano = segundos_por_dia * dias_por_ano;
```

Aqui nós podemos ver um pouco de matemática básica. Após o JavaScript executar essas definições, a variável `segundo_por_ano` vai conter o resultado da multiplicação de 60, 60, 24 e 365. De agora em diante, sempre que o JavaScript ver a variável `segundos_por_ano`, ela vai ser substituída por esse valor.

Finalize com

```
-->
</script>
```

Então agora que você já tem uma variáveis com valores atribuídos, Nós podemos utilizá-las para mostrar para o usuário, por exemplo, desta forma:



2. Mostrando valores no corpo

Agora que as variáveis estão definidas, vamos utilizá-las. Para demonstrar JavaScript no corpo do HTML, você usa exatamente da mesma forma que no cabeçalho:

```
<script language="JavaScript">
<!--
```

E aqui vai como se usa o JavaScript para escrever variáveis em uma página da Web:

```
// este é o código JavaScript dentro do corpo do HTML

document.writeln("<b>O ano tem ");
document.writeln(segundos_por_ano);
document.writeln(" segundos!</b>");
```

`document.writeln()` (*writeln* vem de “*write line*” - escrever linha) é o comando que escreve na página o que quer que esteja dentro dos parênteses.

Existem muitos detalhes e explicações sobre o comando `document.writeln()`. Por hora, lembre-se que você ainda está dentro de Tags de `<script></script>`, e por isso você vai precisar usar seus textos entre aspas para que eles sejam mostrados em sua página.

Caracteres entre aspas são impressos; caracteres que não estão entre aspas são considerados variáveis, e o JavaScript tenta interpretá-los. Faça a experiência: se você colocar seu texto sem aspas, não aparecerá nada (ou você receberá algum erro), pois o JavaScript acha que é uma variável sem valor definido. E se você tivesse escrito a segunda linha da forma `document.writeln("segundos_por_ano")` o JavaScript teria interpretado como texto e você receberia no browser a frase “O ano tem segundo_por_ano segundos!”.

Tudo que é colocado entre aspas é considerado uma **string** e por isso não é interpretado. Este exemplo utiliza aspas duplas (“), mas você também pode utilizar aspas simples ('). As duas funcionam.

Agora que você viu que o JavaScript pode ser colocado tanto no cabeçalho da página como no corpo, você deve estar se perguntando qual a diferença e por que se utiliza sempre no cabeçalho.

O motivo principal é que o cabeçalho é lido antes da página carregar. Então para que os valores das variáveis sejam mostrados na hora em que a página é exibida, o JavaScript já deve ter executado seus comandos e cálculos. Caso você tente definir suas variáveis depois de elas serem pedidas pelo browser, o JavaScript retornará um erro.

3. Strings

Como foi mencionado anteriormente, qualquer grupo de caracteres colocado entre aspas será chamado de **string**. Aspas duplas ou simples servem para isso. Mas assim como as variáveis podem armazenar números, elas também podem armazenar strings. Então podemos dizer que:

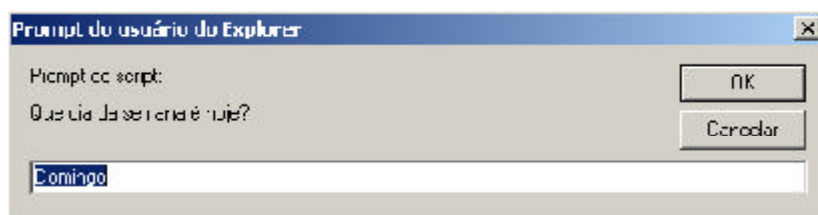
```
var ceu_azul = "Pelo jeito não vai chover hoje."  
var ceu_cinza = "Eu acho que hoje vai chover."
```

Com essas indicações, o JavaScript declara as variáveis `ceu_azul` e `ceu_cinza` e faz delas equivalentes as suas strings. Com isso você pode escrever a linha:

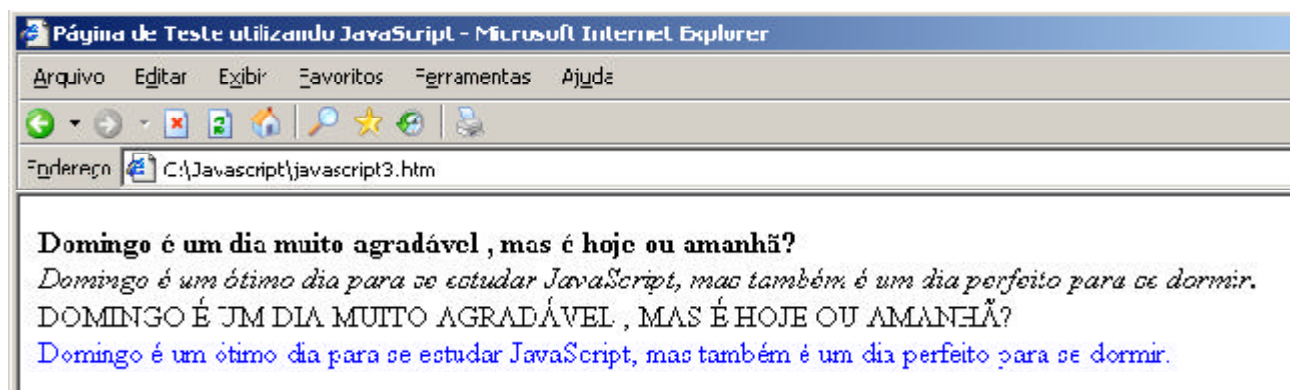
```
document.writeln(ceu_azul);
```

sempre que você precisar declarar uma mensagem longa com a sua opinião sobre o tempo hoje.

Aqui vai um exemplo de o que você pode fazer com strings.



Após entrar o dia da semana na caixa acima, eu recebo a seguinte página:



Vamos ver como isto funciona:

Este é o código completo do documento da página anterior:

```
<HTML>
<head>
<title>Página de Teste utilizando JavaScript</title>

<script language="JavaScript">
<!--

// Pede a variável para o usuário
var dia_hoje = prompt("Que dia da semana é hoje?", "Domingo");

// declara strings curtas
var agradavel = " é um dia muito agradável ";
var pergunta = ", mas é hoje ou amanhã? ";
var estudo = " é um ótimo dia para se estudar JavaScript";
var sono = ", mas também é um dia perfeito para se dormir.";

// constrói strings longas
var dia_agradavel = dia_hoje + agradavel + pergunta;
var dia_cansado = dia_hoje + estudo + sono;

// Modifica a aparência das strings
var agradavel_negrito = dia_agradavel.bold();
var sono_italico = dia_cansado.italics();
var agradavel_berrante = dia_agradavel.toUpperCase();
var sono_frio = dia_cansado.fontcolor('blue');

-->
</script>

</head>
<body>

<script language="JavaScript">
<!--

// este é o código JavaScript dentro do corpo do HTML
document.writeln(agradavel_negrito + "<br>");
document.writeln(sono_italico + "<br>");
document.writeln(agradavel_berrante + "<br>");
document.writeln(sono_frio + "<br>");

-->
</script>

</body>
</html>
```


Agora nós estamos vendo algo novo:

```
var dia_hoje = prompt("Que dia da semana é hoje?", "Domingo");
```

Aqui nós estamos utilizando o método `prompt`, para pedir para o usuário alguma informação, no caso, o valor da variável `dia_hoje`. Quando o método `prompt` é utilizado, ele nos traz uma janela de diálogo que pede por alguma informação do usuário. Quando o usuário pressiona “OK”, o `prompt` retorna o valor de o que quer que o usuário tenha digitado na caixa de diálogo.

Note que o método `prompt` pede dois parâmetros, no caso, duas Strings. O primeiro parâmetro é o que vai ser impresso acima do campo onde o usuário pode digitar sua informação na caixa de diálogo. Nesse caso é a pergunta “Que dia da semana é hoje?”. O segundo parâmetro, “Domingo”, é um exemplo de valor padrão pré-definido para o campo. Se você não quiser ter um valor padrão pré-definido, apenas deixe as duas aspas lá, mas sem nada entre elas, dessa forma:

```
var dia_hoje = prompt("Que dia da semana é hoje?", "");
```

As próximas linhas são indicações de variáveis, como nós já vimos antes. Após essas linhas, nós vemos:

```
var dia_agradavel = dia_hoje + agradavel + pergunta;
```

Essa linha nos introduz a um operador de strings: O sinal de mais. Quando você utiliza o sinal de mais entre duas strings ou duas variáveis que contêm strings, significa que você vai **concatenar** (não significa somar, nesse caso, é colocar uma no lado da outra). Então a linha acima cria uma nova variável chamada `dia_agradavel` que contém uma string feita do conteúdo das três variáveis citadas. Nesse caso, seria “(o dia que o usuário digitou)” + “é um dia muito agradável” + “, mas é hoje ou amanhã?”, em outras palavras...

```
var dia_agradavel = dia_hoje + agradavel + pergunta;
```

é igual a

```
var dia_agradavel = "Domingo é um dia muito agradável, mas é hoje ou amanhã?";
```

(isso caso o dia que o usuário definiu seja “Domingo”).

As próximas linhas mostram alguns efeitos que você pode aplicar em strings. Todos eles funcionam da mesma forma, então nós só vamos ver 3 deles.

```
var agradavel_negrito = dia_agradavel.bold();  
var agradavel_berrante = dia_agradavel.toUpperCase();  
var sono_frio = dia_cansado.fontcolor('blue');
```

A primeira linha, que aplica o atributo `.bold()` no final da variável `dia_agradavel` diz que o JavaScript deve adicionar tags de `` e `` antes e depois do valor da variável. Seria o mesmo que dizer:

```
var agradavel_negrito = "<b>" + dia_agradavel + "</b>";
```

Mas da forma `.bold()` você tem um código muito mais enxuto. As duas formas funcionam imprimindo o texto na tela em negrito.

A linha seguinte, que declara a variável `agradavel_berrante`, mostra uma string que você não consegue declarar em HTML (somente com CSS) que faz com que todas as letras fiquem em maiúsculas.

E a terceira linha mostra um exemplo de como trocar a propriedade de uma string. Todas as strings tem cor, e você pode mudá-la utilizando o comando `string.fontcolor('cor nova');`. Você também poderia ter feito isso:

```
var sono_frio = "<font color='blue'>" + dia_cansado + "</font>";
```

Mas é mais fácil até de ler se você usar dessa forma:

```
var sono_frio = dia_cansado.fontcolor('blue');
```

O resto todo do exemplo você já viu, menos a linha:

```
document.writeln(agradavel_negrito + "<br>");
```

Esta é apenas uma linha normal com o comando `document.writeln()`, só que ao invés de você imprimir a string somente, nós concatenamos duas strings e imprimimos o resultado. Você poderia ter feito isso em duas linhas, assim:

```
var negrito_quebra = agradavel_negrito + "<br>"  
document.writeln(negrito_quebra);
```

Mas dessa forma você criaria ainda mais uma variável e escreveria mais uma linha, quando não há necessidade.

4. Exemplo com variáveis e strings

Experimente o seguinte código:

```
<HTML>
<head>
<title>Página de Teste utilizando JavaScript</title>

<script language = "JavaScript">
<!--

var nome = prompt("Me diga um nome: ","");
var verbo = prompt("Me diga um verbo, conjugado no passado: ","");
var adjetivo= prompt("Me diga um adjetivo: ","");

var frase = nome + " " + verbo +
    " na sala, e era " + adjetivo + ".<p>";

-->
</script>

</head>
<body>

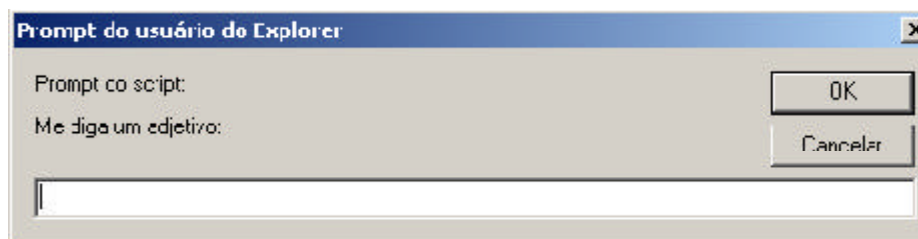
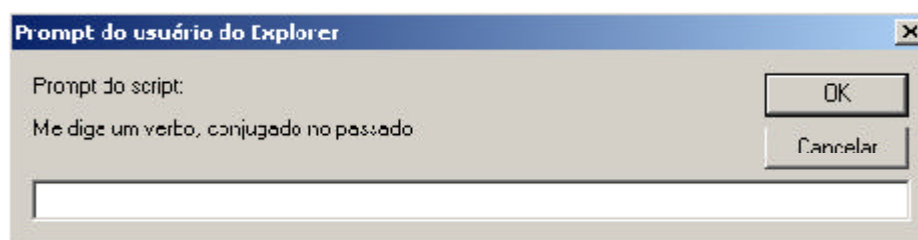
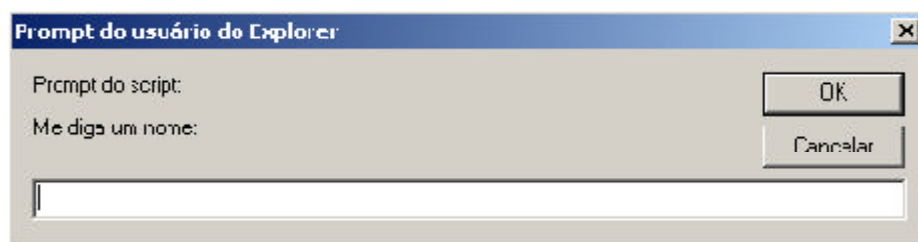
<script language="JavaScript">
<!--

// este é o código JavaScript dentro do corpo do HTML
document.writeln(frase);

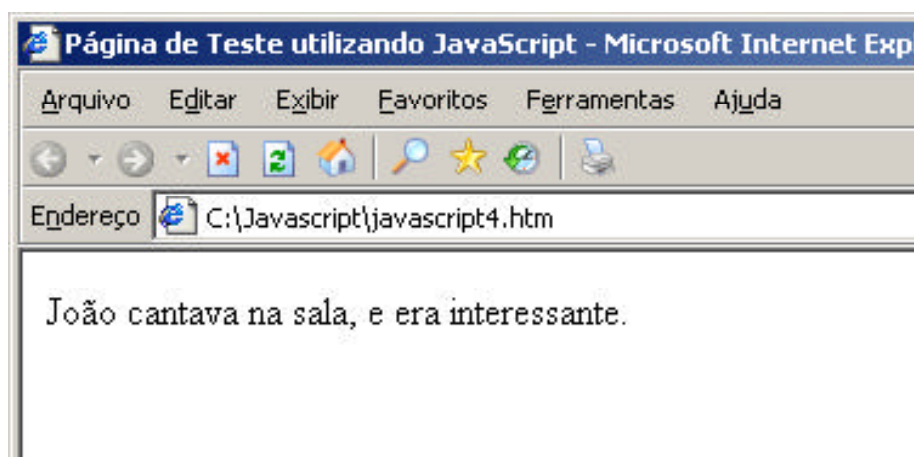
-->
</script>

</body>
</html>
```

O resultado desse arquivo deve ser uma página que vai lhe pedir uma sequencia de caixas de diálogos:



E com as suas respostas, vai montar uma frase:



Experimente modificar as strings e suas posições.

5. Condições “If - then” (“se - então”)

O que as indicações “if - then” fazem é permitir que o seu programa se comporte de maneira diferente, dependendo no que o usuário fizer. Por exemplo, você pode escrever um script que agiria diferente com você do que com outras pessoas.

O “if - then” funciona basicamente desta forma:

```
if (alguma condição é verdadeira)
{
    faz alguma coisa;
    faz alguma coisa;
    faz alguma coisa;
}
```

As partes mais importantes desta estrutura são:

- Começa com a palavra “if” (minúscula)
- Existe uma condição dentro dos parênteses que deve ser verdadeira ou falsa
- Existe um conjunto de indicações que devem ser executadas se a condição for verdadeira. Essas indicações estarão entre chaves.

Lembre-se que o espaçamento apenas existe para deixar o script mais legível. Você pode escrever seu script inteiro em apenas uma linha, mas será difícil de entendê-lo depois.

6. Exemplo "if-then"

Experimente este exemplo:

```
<HTML>
<head>
<title>Página de Teste utilizando JavaScript</title>

<script language = "JavaScript">
<!--

var alertas = prompt("Você gosta de janelas de alerta?", "Digite sim ou não.");
if (alertas == "sim")
{
alert("Bem-vindo! Eu sou uma janela de alerta!");
}

-->
</script>
</head>
<body>

corpo do html

</body>
</html>
```



E se você responder que sim....



Com esse exemplo, seu browser vai perguntar se você gosta de Janelas de alerta ou não. Vamos analisar o código:

```
var alertas = prompt("Você gosta de janelas de alerta?","Digite sim ou não.");
```

Esta primeira linha já não é mais estranha.

```
if (alertas == "sim")
```

Esta segunda linha já apresenta algo novo: uma condição. Esta condição diz que se a variável `alertas` for igual ao valor "sim", as indicações dentro das chaves devem ser executadas. Se a variável for igual a qualquer outra coisa, o script simplesmente será ignorado.

Note que a condição tem **dois** sinais de igual. Isso porque quando você utiliza apenas um sinal de igual, você está *agregando* um valor à variável, e não questionando seu valor. Alguns browsers quando encontram este erro (apenas um sinal de igual em uma condição "if") avisam sobre ele, mas não é necessário passar por isso.

Outras condições típicas são:

- `(variavel1 > variavel2)` - se é verdade que a variavel1 é maior que a variavel 2
- `(variavel1 < variavel2)` - se é verdade que a variavel1 é menor que a variavel2
- `(variavel1 <= variavel2)` - Se é verdade que a variavel1 é menor ou igual a variavel2
- `(variavel1 != variavel2)` - Se é verdade que a variavel1 é **diferente** da variavel2

Para deixar suas condições mais interessantes:

Se você quer conferir se duas coisas são verdadeiras antes de executar suas indicações dentro das chaves, faça o seguinte:

```
if ((idade < 18) && (bebe_alcool == "sim"))
{
document.writeln("Você é muito novo para beber!!");
}
```

Note os dois sinais de &. É desta forma que você diz "e" no JavaScript. Note também que toda a cláusula está entre sinais de parenteses, incluindo as duas sub-partes com as variáveis.

Se você quer testar se qualquer uma das duas condições é verdadeira antes de executar as indicações, use desta forma:

```
if ((variavel1 == "sorvete") || (variavel1 == "refrigerante"))
{
document.writeln("No verão eu gosto de " + variavel1);
}
```

As duas barras entre as duas definições de variáveis significam "ou". Ou seja, se o valor oferecido for sorvete **ou** refrigerante, o script dentro das chaves será executado. Se for qualquer outra coisa, ele não será executado.

Experimente o seguinte exercício:

```
<HTML>
<head>
<title>Página de Teste utilizando JavaScript</title>

<script language = "JavaScript">
<!--
var cerveja = prompt("Qual cerveja você prefere, Skol ou Kaiser? ","");
var adjetivo;
var cor;

if (cerveja == "Skol") {
    adjetivo = "redondo";
    cor = "orange"
} else if (cerveja == "Kaiser") {
    adjetivo = "quadrado";
    cor = "red"
} else {
    adjetivo = "confuso"
    cor = "blue"
}

var frase = "Se você gosta da cerveja " + cerveja + ", este curso será " +
    adjetivo + ".";

-->
</script>

</head>
<body>

<script language = "JavaScript">
<!--

document.writeln(frase.fontcolor(cor));

-->
</script>

</body>
</html>
```


Neste exercício nós vemos algo de novo:

```
} else if (cerveja == "Kaiser") {  
    adjetivo = "quadrado";  
    cor = "red"
```

A definição `else if` é a mesma coisa que dizer “então se”. No caso, a definição acontecerá caso a primeira cláusula não for verdadeira. No caso o script já chegou a conclusão de que o valor não é o primeiro comparado (não é Skol), então ele faz uma segunda comparação (se é Kaiser).

Se a comparação for verdadeira então, o script dentro das chaves correspondentes será executado.

Caso nenhuma das duas primeiras comparações for verdadeira, você pode colocar uma indicação de `else`, ou seja “então”.

```
} else {  
    adjetivo = "confuso"  
    cor = "blue"  
}
```

7. Eventos de link

Cada ação feita pelo usuário com o mouse é considerada pelo JavaScript como um Evento. Por exemplo, quando o usuário clica em um link, esse evento é chamado de `onClick` (ao clicar), ou quando ele passa o mouse sobre algum lugar, o evento correspondente é o `onMouseOver` (ao ter o mouse em cima).

Com os eventos, a primeira coisa a se notar é que as tags de `<script>` não são necessárias. Isso é porque qualquer coisa que esteja entre as aspas de um atributo `onClick` ou `onMouseOver` será considerado automaticamente JavaScript pelo browser.

Aqui vai um exemplo:

```
<a href="#" onClick="alert('Você me clicou!');">Clique aqui!</a>
```

Este é apenas um link normal de âncora, mas tem o elemento JavaScript `onClick=""` agregado como se fosse um atributo do link que diz *“Quando clicar em mim, executar o que diz dentro das aspas”*.

Os links feitos com a cerquilha (#) servem para que você não indique lugar nenhum para onde o browser deva ir. O problema deste link é que caso você tenha já nomeado alguma âncora na sua página (por exemplo, `#topo`), este link irá irremediavelmente, levar a página para o topo. Para evitar de isso acontecer, você pode escrever o código da seguinte forma:

```
<a href="#" onClick="alert('Você me clicou!');return false;">Clique aqui!</a>
```

Note o `return false;` logo após a indicação de `alert` dentro do `onClick`. Isso diz para o JavaScript impedir o browser de procurar o que está dentro do HREF.

Você ainda tem a opção de não utilizar o “#” desta forma:

```
<a href="Javascript:onClick(alert('Você me clicou!'))">Click on me!</a>
```

Agora experimente a seguinte linha:

```
<a href="#" onMouseOver="alert('Opa!');return false;">Passe o mouse aqui!</a>
```

Ela funciona da mesma forma que o `onClick`, só que o evento é diferente: ele dispara o JavaScript assim que o mouse passa por cima do link.

8. Troca de Imagens

Uma das características mais utilizadas do JavaScript é a habilidade de trocar imagens numa versão `onMouseOver`. Essa função só funciona no Netscape ou no Internet Explorer 4.0+.

Vamos ver um exemplo rápido de troca de imagens:

```
<HTML>
<head>
<title>Página de Teste utilizando JavaScript</title>

</head>
<body>


<br>
<a href="#" onMouseOver="document.imagem.src='imgs/b.gif';">muda!</a><br>
<a href="#" onMouseOver="document.imagem.src='imgs/c.gif';">muda!</a><br>

</body>
</html>
```

Vamos ver como ele funciona. A primeira linha que nos interessa é a seguinte:

```

```

Nesta linha nós temos uma tag normal de ``, exceto pelo fato de que nós demos um nome à esta imagem: "imagem". Este nome poderia ser qualquer coisa, seguindo as mesmas regras das variáveis já vistas no começo do curso.

A próxima linha interessante é a seguinte:

```
<a href="#" onMouseOver="document.imagem.src='imgs/b.gif';">muda!</a>
```

Aqui é onde acontece a troca. É apenas uma função `onMouseOver` como você já viu antes. A diferença é nas indicações. Onde diz `document.imagem.src` significa que a troca será feita na imagem nomeada "imagem", neste documento.

Note as aspas simples que contornam o endereço da imagem que será trocada. Para o browser, não vai fazer diferença de você utilizá-las ou não, mas você não poderá utilizar aspas duplas nesse caso, ou o JavaScript vai achar que as suas indicações acabam ali.

9. Revisão

Nesta lição, nós vimos:

- **Variáveis**

Variáveis podem conter números ou strings. Existem algumas restrições consideráveis sobre os nomes das variáveis.

- **Indicações**

Indicações acabam sempre com ponto e vírgula.

- **Strings**

Strings são sequências de caracteres dentro de aspas. Você pode usar tanto aspas duplas como simples. Strings podem ser concatenadas com um sinal de mais (+) com outras strings ou variáveis.

- **document.writeln()**

Utilizado para escrever texto e HTML em uma página.

- **prompt**

Serve para colher valores dos usuários, através de uma caixa de diálogo.

- **If - then - else**

Você pode utilizar as cláusulas de if - then - else para que o JavaScript se comporte de maneira diferente, dependendo das ações do usuário.

- **Eventos de links**

`onClick` e `onMouseOver` dentro de tags de links podem executar JavaScript para reagir de acordo com ações do usuário.

- **Troca de Imagem**

Nomeando uma imagem você pode fazer com que o JavaScript mude a imagem em exibição.

Lição 3

1. Modelo de Documento de Objeto JavaScript, Janelas e Molduras

O Modelo de Documento de Objeto JavaScript (MDO) é a maneira como o JavaScript descreve as páginas da WEB. Esta lição é destinada a explicar o funcionamento do MDO, que é a essência do JavaScript.

Para começar a entender o funcionamento do MDO, vamos ver como o browser interpreta suas janelas.

2. Introdução a manipulação de Janelas

Antes de entender como abrir uma janela com JavaScript, você já deve saber como funciona para abrir uma utilizando o próprio HTML.

Nos browsers mais recentes você pode abrir uma janela usando o atributo href. Por exemplo:

```
<a href="nova.html" target="minha_janela">clique aqui!</a>
```

Neste link você abriu o endereço nova.html em uma janela chamada “minha_janela” utilizando o atributo target, que significa “alvo”. Desta forma, a janela nova aberta apartir deste link está associada com o nome “minha_janela”. Então se eu tiver mais algum link na minha página que tenha o target=“minha_janela”, seu endereço contido no atributo href será aberto na mesma janela de browser do link anterior.

Com isso em mente, vamos ver como se abrem janelas utilizando JavaScript.

3. Manipulação de Janelas usando JavaScript

Abrir janelas usando HTML é muito fácil, mas também muito limitado. O browser é que controla como a janela deve ser. Você não tem controle sobre sua aparência ou tamanho.

Para ter este controle, você pode utilizar o JavaScript desta forma:

```
window.open("URL", "nome", "características");
```

Estas indicações abrem uma janela com o endereço que estiver no primeiro parâmetro do método. No caso acima, é onde está escrito “URL”. Ali você deve substituir pelo endereço real que você quer abrir na janela nova.

O segundo parâmetro é o método que chama pelo nome da janela. É o mesmo nome que nós vimos na página anterior. Se você abre uma janela e já existe uma outra aberta associada com o mesmo nome, o endereço novo abrirá nesta janela aberta.

O terceiro parâmetro, características, é uma lista de componentes diferentes que uma janela pode ter. É um parâmetro opcional, então vamos fazer alguns exemplos dos dois primeiros parâmetros antes de ver as características.

4. Exemplos de janelas abertas com JavaScript

Experimente fazer uma página com os seguinte link:

```
<a href="#" onClick="window.open('nova1.htm','minha_janela');">clique aqui!</a><br>
<a href="#" onClick="window.open('nova2.htm','outra_janela');">clique aqui!</a><br>
<a href="#" onClick="window.open('nova3.htm','minha_janela');">clique aqui!</a><br>
```

Clique nos três links sem fechar nenhuma das janelas abertas. Você vai notar que o último link faz abrir a página `nova3.htm` na mesma janela em que foi aberta a página `nova.htm`, pois ambas estão com o mesmo nome.

Como as características da janela (o terceiro parâmetro) são opcionais, você pode simplesmente não informá-los, e sua janela será como a última janela de browser fechada (padrão da plataforma windows).

Note que foi utilizado a função `onClick` para chamar a `window.open()`, mas isso não é necessário. Veremos mais tarde outras formas de se chamar o `window.open()` dentro de tags `<script>`.

5. Características das Janelas

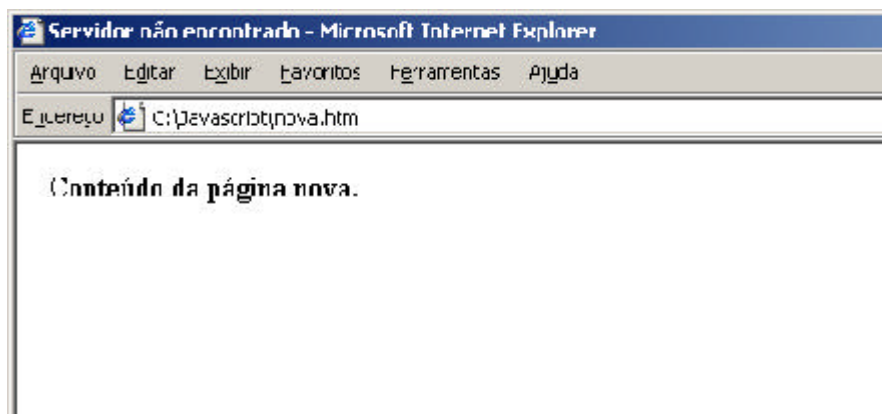
O terceiro parâmetro do método `window.open()` é uma lista de características que você define para a janela a ser aberta. Como já vimos anteriormente, se você não incluir este parâmetro, a janela aberta terá os padrões definidos pelo usuário.

Entretanto, se você definir qualquer característica neste parâmetro, somente esta definida será apresentada. A maneira de defini-las é listando-as, separadas por vírgulas.

Por exemplo, se você listar:

```
window.open ( "pagina.html", "nome", "location,menubar" );
```

... você terá uma janela que terá somente a barra de endereço do browser (`location`) e a barra de menu (onde tem as opções “arquivo”, “editar”...). Dessa forma:



Aqui você pode conferir outro exemplo:

```
window.open("pagina.html", "nome", "location,height=100,width=100");
```

Com isso você terá uma janela que tem a barra de endereços do browser, com 100 pixels de altura e largura. Note novamente que não existe nenhum espaço nesta string.

Se você quer especificar uma janela que tenha todas as características, MENOS uma ou duas, você pode defini-las nulas, apenas dizendo que elas são iguais a “não” (`no`). Desta forma:

```
window.open("url", "nome", "location=no,status=no");
```

Assim você especificou uma janela com todas as características normais, MENOS a barra de endereços e a barra de status do browser.

Vamos conferir uma lista de características que você pode utilizar neste parâmetro do método `window.open()`:

- **menubar**

É a barra de menu do browser, que contém, geralmente, as opções “Arquivo” (file), “Editar” (edit), “Exibir” (View), entre outras.

- **status**

Esta é a barra de mensagens que fica na base do browser. Quando você move o mouse por cima de um link HTML, o endereço de destino é exibido na barra de status. Você pode usar o JavaScript para esconder esta barra (para que o usuário não saiba o destino dos links ou não veja o processamento das páginas).

- **scrollbars**

Isso faz com que as barras de scroll apareçam se necessárias (lembre-se que se você não informar scrollbars, sua janela NÃO terá barras de rolagem, mesmo que tenha conteúdo para rolar).

- **resizable**

Se a opção resizable estiver na lista, o usuário vai poder redimensionar a janela.

- **width**

Esta é a definição de largura da página, para ser descrita em pixels.

- **height**

A altura, também em pixels.

- **toolbar**

Esta é a barra do browser que contém os botões de navegação, como o “voltar”, “avançar”, “atualizar” e “parar”.

- **location**

A barra onde você digita o endereço das páginas.

- **directories**

A barra onde ficam, no Netscape, as opções de “what’s new”, “what’s cool”, entre outras.



6. O Modelo de Documento de Objeto JavaScript

Agora que você já sabe como abrir uma janela com suas preferências, vamos ver como manipular as informações dentro dela. Para realmente ter controle sobre o conteúdo de uma janela, você tem que aprender sobre o Modelo de Documento de Objeto JavaScript (MDO). E antes de você aprender sobre o MDO, é bom você entender sobre programação orientada a objeto.

Uma breve visão geral

Programação orientada a objeto - especialmente a versão JavaScript - não é tão complicada de entender. A idéia principal é que a informação é armazenada em termos de objetos. O JavaScript é bem prático porque já vem com uma biblioteca interna de objetos. Por exemplo, uma janela (`window`) é um objeto.

Propriedades do Objeto

Os Objetos tem propriedades que os descrevem. Algumas das propriedades de um objeto Janela são seu nome, as palavras na barra de status, a URL do documento que ela exibe, e o próprio documento, o que inclui palavras, imagens e hyperlinks na janela.

Em JavaScript, você ganha um objeto padrão em uma janela de browser chamado `window`. Uma das propriedades do `window` é o que diz na barra de status. Para você ter controle da barra de status de uma janela, você pode definí-la desta forma:

```
var barra_status = window.status;
```

Isto diz: "Ache o que tem na barra de status do objeto `window`, e carregue o valor na variável `barra_status`". Agora, além de poder ler o que tem na barra de status, você também pode mudar o que diz lá.

```
window.status = "Eu sou uma mensagem na barra de status!"
```

7. Modificando a Barra de Status

Experimente o seguinte código:

```
<a href="#" onMouseOver="window.status='Tudo! e com você?';return true;">Olá! Tudo bem com você?</a>
```

Isso diz: “Quando o mouse passar por cima deste link, mude a mensagem do status bar”. Note o `return true;` dentro do `onMouseOver`. Se você não colocá-lo lá, o browser vai fazer o que ele sempre faz quando o mouse está em cima de um link: mostra seu endereço de destino.

Objetos tem métodos

Além das propriedades, objetos também tem métodos. Métodos são ações que um objeto sabe interpretar. Por exemplo, janelas sabem como abrir outras janelas.

```
window.open( "URL", "name", "features" ).
```

Isso diz para o JavaScript usar o método `open` (abrir) do objeto `window` para abrir uma janela nova.

Neste exemplo, o método é chamado da mesma forma que as propriedades: o nome do objeto, uma pausa, e então o método. A diferença principal é que os métodos são sempre seguidos de parênteses que contenham os parâmetros dos métodos. Mesmo que o método não tenha parâmetros, ele vai precisar dos parênteses, como por exemplo, algo que já vimos neste curso:

```
var dia_agradavel = agradavel_italico.italics();
```

Como você deve ter percebido, strings também são consideradas objetos, e neste caso, `italics()` é o método desta String.

8. Comunicação entre janelas

Experimente o seguinte código:

```
<a href="#" onClick="window.blur();return false;">Vou me esconder!</a>
```

Se você clicar neste link, você vai desfocar a sua janela (ou seja, jogar ela para baixo das outras janelas abertas). A princípio pode não fazer muito sentido você desfocar a janela em que está trabalhando, mas você pode com isso jogar outra janela importante para frente partindo daí. Para isso, você precisa saber o nome da janela, e chamá-la da seguinte forma:

```
<a href="#" onClick="window.focus();return false;">Mostrar janela escondida!</a>
```

9. Hierarquia do MDO

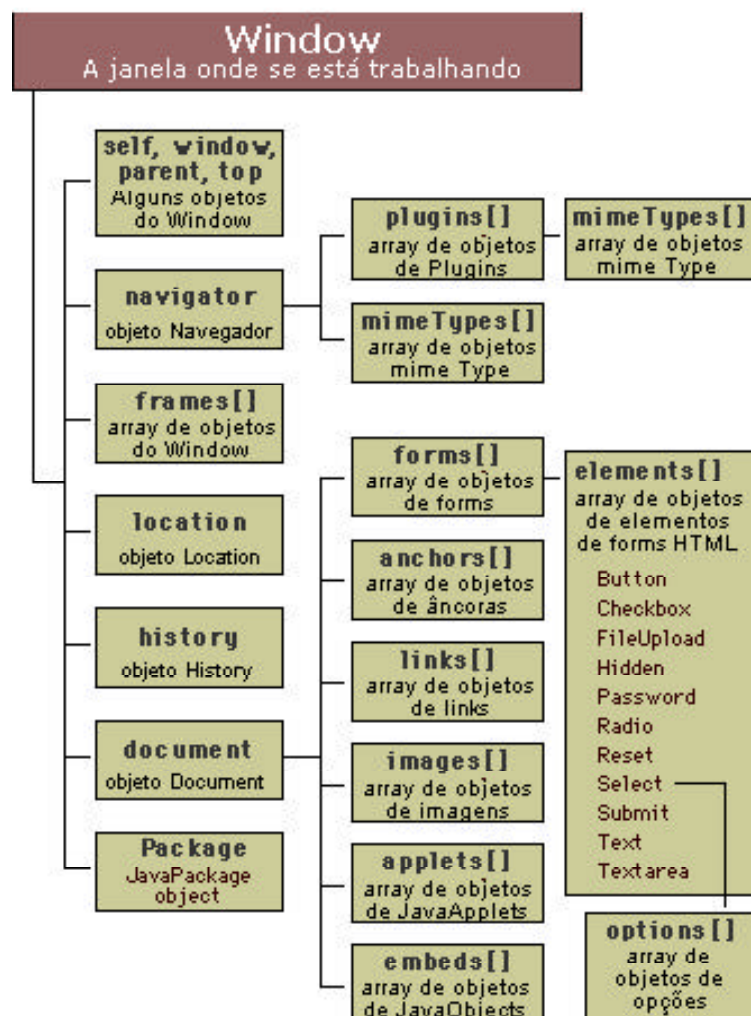
Até o momento nós já sabemos que JavaScript tem objetos padrões, como as janelas (`window`). Nós aprendemos que objetos tem propriedades que os descrevem, e métodos que descrevem o que os objetos sabem interpretar. Agora vamos aprofundar um pouco mais neste assunto.

Uma coisa interessante sobre os objetos é que as propriedades de um objeto também podem ser objetos. Por exemplo, `window` (janela) tem a propriedade `document` (documento), que é a própria página da janela. Esta propriedade é um objeto ela própria também, que tem seus próprios métodos. Nós vimos um exemplo que falava de trocas de imagens anteriormente. Vamos revê-lo:

```
<a href="#" onMouseOver="window.document.minha_imagem.src='botao.gif';">muda</a>
```

Esta string, `window.document.minha_imagem.src='botao.gif'`, significa: "Ache o `document`, que é propriedade de `window`, depois ache o `minha_imagem` que é propriedade do `document`, então ache o `src` que é propriedade do `minha_imagem`, e mude ele para `botao.gif`". Parece complicado? Isso tudo se dá porque `window` é um objeto, `document` é um objeto dentro de `window`, `minha_imagem` é um objeto dentro de `document` e `src` é um objeto dentro de `minha_imagem`.

Pode parecer muito complicado, mas não é. Vamos ver um gráfico da hierarquia dos objetos: (imagem ao lado)



10. Utilizando o MDO em outras janelas

Como nós já vimos, uma troca de imagem pode ser feita com uma linha assim:

```
window.document.minha_imagem.src="botao.gif";
```

Isso funciona dizendo para o JavaScript olhar em `window`, encontrar o seu `document`, então encontrar uma coisa chamada `minha_imagem` lá dentro. Assim que o JavaScript encontra a `minha_imagem`, ele pode mudar para qualquer outra imagem que eu quiser.

Pode ser útil você ter um link em uma janela que troca a imagem de outra. Imagine uma apresentação de slides em que uma janela mostra as reduções dos slides em uma pequena janela para que sejam clicados e mostrados grandes em uma janela maior.

Vamos fazer uma experiência:

crie um arquivo com o seguinte código:

```
<html>
<head>
<title>Página de teste de JavaScript</title>
<script language="JavaScript">
<!--

var janela_grande = window.open("slides.htm","janela_grande");window.focus();

-->
</script>
</head>
<body>

<a href="#" onClick="janela_grande.document.slide.src='imgs/a.gif';return
false;"></a><br>
<a href="#" onClick="janela_grande.document.slide.src='imgs/b.gif';return
false;"></a><br>
<a href="#" onClick="janela_grande.document.slide.src='imgs/c.gif';return
false;"></a><br>

</body>
</html>
```

Logo após, crie uma outra página chamada “slides.htm” com o seguinte código (salve os dois arquivos numa mesma pasta de seu computador):

```
<html>
<head>
<title>Slides</title>

</head>
<body>



</body>
</html>
```

Para testar, abra o primeiro arquivo (o da página anterior).

Quando ele carregar, ele já vai abrir em uma outra janela o arquivo slides.htm, contendo o que seria a versão natural dos slides (definidos aqui com 400px de altura/largura).

No caso, o seu primeiro arquivo é o *controle remoto* dos slides. Se você clicar nas imagens pequenas deles, a imagem grande do arquivo slides.htm irá mudar.

Isso é possível porque quando nós abrimos o primeiro documento, ele já abriu o slides.htm e nomeou sua janela como “janela_grande”. Feito isso, nós podemos encontrar esta janela e modificar seus objetos.

Como a janela dos slides é aberta depois do controle remoto, é bem possível que ela abra por cima e esconda o controle. Para resolver isso, nós adicionamos aquela indicação `window.focus()`; no final do comando que abre os slides.

Agora, para modificar as imagens do slides.htm, nós precisamos saber, além do nome da sua janela (janela_grande), o nome da imagem que será mudada. No caso, o nome colocado é “slide”. Sabendo isso, nós podemos entender como os links do controle remoto conseguem modificar as imagens do slides.htm:

```
onClick="janela_grande.document.slide.src='imgs/c.gif'
```

Nesta linha, nós dizemos: “encontre o objeto janela_grande, dentro dele, encontre o document, e dentro dele, encontre o objeto slide, no objeto slide, encontre seu src, e mude para imgs/c.gif”.

11. Trabalhando com Molduras

Todas as propriedades restantes das janelas nós vamos ver agora nessa lição relacionada a molduras (frames). Nós assumimos aqui que você sabe trabalhar com molduras de HTML.

Para testar, primeiro, vamos criar um *frameset* (conjunto de molduras):

```
<html>
<head>
<title>Slides</title>

</head>

<frameset rows="25%,*">
<frame src="moldura_controle.html" name="m_controle">
<frame src="moldura_conteudo.html" name="m_alvo">
</frameset>

</html>
```

Este conjunto de molduras vai abrir um conjunto com duas molduras, uma para o controle e outra para o resultado (conteúdo). Em JavaScript, os frames são tratados da mesma forma que as janelas.

A primeira moldura (moldura_controle.html) é a que contém JavaScript. A segunda (moldura_conteudo.html) não contém nada.

Na primeira moldura (moldura_controle.html), adicione o seguinte código:

```
<html>
<head>
<title>Controle</title>

</head>
<body>

<a href="#" onClick="top.m_alvo.document.writeln('Alô, sou eu!<br>');">Alô, quem
fala?</a>

</body>
</html>
```

Teste o link “Alô, quem fala?”. Clicando nele, o JavaScript vai escrever a mensagem “Alô, sou eu!” na moldura inferior, que foi nomeada de `m_alvo` no frameset.

O `onClick` da moldura superior diz que é para o JavaScript procurar no `top`, ou seja, no objeto mais acima de todos, no caso, o próprio browser, e dentro dele, o objeto chamado `m_alvo`, que é a nossa moldura inferior. Dentro dela, ele procura por um objeto `document` e nele, escreve a linha (`writeln`) “Alô, sou eu!”.

12. Janelas e Hierarquia de molduras

Nós vimos algumas instâncias de variáveis internas do JavaScript sendo utilizadas nos últimos exemplos. O objeto `window` se refere a qualquer janela em que você esteja trabalhando. Se você estiver em uma moldura e utilizar o comando `window.document.writeln()`, o `writeln` vai aparecer nesta mesma moldura. Outra utilização para o mesmo efeito seria usar o `self` (no caso seria `self.document.writeln()`). Ambos são funcionais.

No exemplo anterior, você foi apresentado à variável interna `top`. Esta variável vai sempre se referir à janela mais acima de todas. No caso se você tiver um conjunto de molduras dentro do outro, você pode dar comandos ao primeiro conjunto utilizando-se do `top`.

Outra variável seria a `parent` (pai), que se refere a janela que contém o conjunto de molduras em que você está trabalhando. Se você estiver trabalhando em um conjunto de molduras que fica dentro de outro, você pode escolher o conjunto originário da sua moldura utilizando-se do `parent`.

13. Revisão

Nós vimos 4 tópicos principais nesta lição: janelas, molduras, MDO e programação orientada a objeto. Agora você já deve conhecer sobre:

- **Abrir janelas**

Como abrir janelas e fazer com que elas tenham a aparência que você escolhe

- **Comunicação entre browsers**

Como fazer com que o JavaScript de uma janela afete a outra.

- **Comunicação entre molduras**

Como fazer com que o JavaScript afete outras molduras

- **Objetos**

Objetos tem propriedades e métodos

- **Modelo de objeto de documento**

Uma janela de browser é um objeto que contém outros objetos. Se você sabe a hierarquia dos objetos do MDO, você saberá afetar diferentes aspectos de uma página HTML.

Lição 4

Existem duas partes principais na sintaxe do JavaScript e em suas bibliotecas. Na lição 2, que introduzia a sintaxe, nós vimos variáveis, indicações e cláusulas `if - then`, que fazem parte de qualquer linguagem de programação. Agora está na hora de aprender o resto da sintaxe.

Só existem mais três aspectos importantes ainda para serem vistos da sintaxe do JavaScript: `loops`, `arrays` e `functions` (funções).

1. Introdução a Loops

Pode acontecer de você precisar fazer a mesma coisa mais de uma vez. Por exemplo, você quer que um usuário lhe forneça uma senha, mas você quer tentar pedir a senha para ele até que ele acerte. Se você quisesse lhe dar apenas duas tentativas, você poderia fazer algo desse tipo:

```
<html>
<head>
<title>Página de teste de JavaScript</title>
<script language="JavaScript">
<!--

var minha_senha = "com licença";
var resposta = prompt("Como é que se diz?","");
if (resposta != minha_senha) {
    resposta = prompt("Não, seja mal-educado e peça licença!","");
    if (resposta != minha_senha) {
        document.write("Seu mal-educado!<p>");
    } else {
        document.write("Claro, fique a vontade!<p>");
    }
} else {
    document.write("Fique a vontade!<p>");
}
-->
</script>
</head>
<body>

</body>
</html>
```

Infelizmente, esse tipo de coisa não funciona se você quiser continuar perguntando até que ele acerte. E se você resolvesse perguntar mais uma vez você teria 3 níveis de cláusulas if-then, o que não é boa coisa.

A melhor maneira de se fazer coisas iguais mais de uma vez é utilizando o loop. Neste caso, você pode utilizar o loop para continuar pedindo a senha até que a pessoa acerte, ou desista.

Vamos ver um exemplo de utilização do loop:

```
<html>
<head>
<title>Página de teste de JavaScript</title>
<script language="JavaScript">
<!--

// Este é um script simples que demonstra
// o funcionamento do loop pedindo uma senha

var senha="por favor";
var resposta="";

while (resposta != senha)
{
    resposta = prompt("Só deixo você entrar se você pedir por favor.", "");
}

-->
</script>
</head>
<body>

</body>
</html>
```

Vamos analisar o exemplo linha por linha:

A primeira coisa importante a se notar é que nós começamos o script declarando as variáveis necessárias para se saber a senha: senha e resposta.

```
var senha="por favor";
var resposta="";
```

A variável senha é a senha real, e a variável resposta é a senha fornecida pelo usuário. A Variável resposta inicia em branco o script, pois o usuário ainda não forneceu seu valor.

E então, o loop:

```
while (resposta != senha)
{
    resposta = prompt("Só deixo você entrar se você pedir por favor.", "");
}
```

Esse é um loop while (enquanto). O loop while funciona basicamente desta forma:

```
enquanto (testa se alguma coisa é verdadeira)
{
    executa as indicações dentro das chaves
}
```

Então isso significa que “enquanto a resposta não for igual a senha, execute o prompt”. O loop vai continuar executando as indicações dentro das chaves até que o teste dê verdadeiro. Neste caso, o teste somente será verdadeiro quando o usuário digitar a senha correspondente a string da variável senha (que nesse caso é “por favor”).

A variável resposta deve ser declarada no começo para que o script funcione em todos os browsers. Alguns browsers mais antigos respondem com um erro se você fizer uma comparação resposta != senha, sem ter declarado a resposta e a senha antes. Por esse motivo, começamos definindo a variável resposta com o valor “”.

Vamos ver mais um exemplo de loop:

```
<html>
<head>
<title>Página de teste de JavaScript</title>

<script language="JavaScript">
<!--

var quantos = prompt("Quantos X's você quer ver? (de 1 a 10 está de bom
tamanho)", "5");
var xises="";
var loop = 0;
while (loop < quantos)
{
    xises = xises + "x";
    loop=loop+1;
}

alert(xises);

-->
</script>
</head>
<body>

</body>
</html>
```

Com esse exemplo, você deve ter recebido uma janela perguntando quantos X's você queria, e deve ter recebido em seguida uma janela alert com o número de X's pedidos. Vamos analisar este script:

A primeira coisa feita é perguntar quantos X's o usuário quer:

```
var quantos = prompt("Quantos X's você quer ver? (de 1 a 10 está de bom
tamanho)", "5");
```

Depois, nós declaramos algumas variáveis que serão necessárias:

```
var xises="";
var loop = 0;
```

E então, o loop em si:

```
while (loop < quantos)
{
    xises = xises + "x";
    loop=loop+1;
}
```

Esse trecho do script diz: “enquanto a variável loop é menor que o número de X's que o usuário pediu, acrescente outro x na variável xises e mais um na variável loop (para que ele possa contar quantas vezes deve acontecer)”.

Este loop vai continuar adicionando X's para a variável xises até que ela tenha o mesmo número de X's que a variável quantos, que é a quantidade que o usuário especificou.

Aqui temos uma linha do tempo do que acontece se uma pessoa pedir por 2 X's:

Primeiro tempo:

- `xises = ""` (pois essa variável foi declarada inicialmente como "")
- `loop=0` (porque foi declarada inicialmente como 0)
- `quantos=2` (porque foi o que o usuário pediu)
- 0 é menor que 2 então
- `xises = xises + "X"`, o que faz de `xises = "X"` agora.
- `loop=loop+1`, o que faz dele então `loop = "1"`

Voltamos para o loop, segundo tempo:

- `loop=1`
- `quantos=2`
- `xises = "X"`
- 1 ainda é menor que 2 então
- `xises = xises + "X"`, que faz dele = "XX"
- `loop=loop+1`, então `loop = 2`

Voltando para o loop: terceiro tempo:

- `loop=2`
- `quantos=2`
- `xises = "XX"`
- 2 não é mais menor que 2 então
- pulamos o loop e caímos no que procede o loop

E o que procede o loop é

```
alert(xises);
```

Que é uma caixa de alerta com o valor da variável `xises`.

Esse tipo de loop é tão comum que os programadores desenvolveram alguns atalhos. Utilizando os atalhos, este loop `while` poderia ter sido escrito dessa forma:

```
while (loop < quantos)
{
    xises += "X";
    loop++;
}
```

A primeira linha, `xises += "X"`, diz "adicione mais um X em mim mesmo". Estes atalhos também funcionam com números. Se você tem um `numero = 5`, e você escrever `numero += 3`, é a mesma coisa que escrever `numero = numero + 3`.

A próxima linha, `loop++`, diz "adicione um em mim mesmo". Então, se o `loop++` é a mesma coisa que `loop=loop+1`, também poderia ter sido escrito como `loop+=1`. Qualquer um desses funciona perfeitamente, fica a sua escolha.

Assim como existe mais de uma maneira de dizer como adicionar +1 em um número, também existe mais de uma maneira de fazer loops. Os loops `while` não são os únicos disponíveis. Outro loop bem utilizado é o `for`.

2. Loop For

O loop `while` do último exemplo poderia ter sido escrito da seguinte forma:

```
var xises="";
for (var loop=0; loop < quantos; loop++)
{
    xises = xises + "x";
}
```

Os loops For funcionam assim:

```
for (valor inicial; teste; incrementação)
{
    fazer isso;
}
```

Então, o loop acima diz que `loop = 0`, e diz que você deve continuar adicionando 1 ao `loop` enquanto ele for menor que `quantos` (`loop < quantos`). É a mesma coisa que o loop anterior, só que em menos linhas. Ambos dizem “adicione um X ao xises no numero quantos”

3. Loops aninhados

Vamos tentar outro exemplo:

```
<html>
<head>
<title>Página de teste de JavaScript</title>

<script language="JavaScript">
<!--

var altura = prompt("Qual a altura que você quer da grade? (de 1 a 10 está bom)", "10");
var largura = prompt("Qual a largura que você quer para a grade? (de 1 a 10 está bom)", "10");
var xises;

window.document.writeln("<h3>Grade!</h3>");

for (altura_loop = 0; altura_loop < altura; altura_loop++)
{
    xises = "";
    for (largura_loop = 0; largura_loop < largura; largura_loop++)
    {
        xises+="x";
    }
    window.document.writeln(xises + "<br>");
}

-->
</script>
</head>
<body>
</body>
</html>
```

Vamos ao script analisar o script:

Depois de perguntar ao usuário pela largura e altura de uma grade, nós vamos entrar em um loop.

O primeiro loop for diz que xises = "". Tente fazer o exemplo sem essa linha e veja o que acontece. Após iniciar o xises, o script entra em outro loop que constrói a largura pedida em X's. Isso vai funcionar quantas vezes o usuário tiver pedido.

4. Exemplo de Loop

Vamos tentar este exemplo:

```
<html>
<head>
<title>Página de teste de JavaScript</title>

<script language="JavaScript">
<!--

var meu_numero = prompt("Quantas palavras? (até umas 5 tá bom)?", "4");
var minha_string = "";
var minha_palavra;

for (loop = 0; loop < meu_numero; loop++)
{
    linha_numero = loop + 1;
    minha_palavra = prompt("Qual vai ser a palavra número " + linha_numero +
"?", "");
    minha_string = "palavra " + linha_numero + ": " + minha_palavra + "<br>" +
minha_string;
}

window.document.writeln("<h3>Ao contrário!</h3>");
window.document.writeln(minha_string);

-->
</script>
</head>
<body>

</body>
</html>
```

Sinta-se a vontade para modificar e testar esse exemplo.

5. Arrays

Nós vimos que variáveis podem conter números, strings e referências à objetos. Mas existe mais um tipo de informação que o JavaScript entende: **Arrays**.

Arrays são listas. Você deve ter uma lista de endereços que você quer visitar, uma lista de nomes que quer lembrar, ou uma lista de cores que você quer que o texto seja exibido. Todas essas listas podem ser armazenadas em arrays.

Veja aqui como criar uma array de cores:

```
var cores = new Array("vermelho","azul","verde");
```

O interessante das arrays é que você pode acessá-las através de números. O primeiro elemento é o elemento [0], e pode ser acessado assim:

```
var meu_elemento = cores[0];
```

Após você executar esta linha, a variável `meu_elemento` vai ter a string “vermelho”. Como você pode ver, você acessou o primeiro elemento de uma array chamando pelo nome da array e pelo número de sua posição entre colchetes. O segundo elemento da lista seria o elemento [1].

Agora que você já criou a array, você pode adicionar e modificar seus valores. Se você decidir mudar o primeiro elemento da array de cores para roxo, ao invés de vermelho, você pode fazer assim:

```
cores[0] = "roxo";
```

Arrays são muito utilizadas em loops, como veremos em seguida.

6. Arrays e Loops

O interessante do uso de arrays com loops é que você pode utilizar cada um dos elementos de uma array para formar um loop. Vamos ver um exemplo:

```
<html><head><title>Página de teste de JavaScript</title>

<script language="JavaScript">
<!--

var array_url = new Array ("proway.com.br", "macromedia.com", "google.com");
var uma_url;

for (loop = 0; loop < array_url.length; loop++)
{

    // monta os endereços das URLs
    uma_url = "http://www." + array_url[loop];

    // abre uma janela
    var nova_janela=open(uma_url,"nova_janela","width=400,height=400");

    // espera pelo clique para a troca
    alert("Clique ok para ver o próximo site");
}

-->
</script>
</head><body></body></html>
```

Vamos analisar este código:

A primeira coisa que nós fazemos é declarar as variáveis:

```
var array_url = new Array ("proway.com.br", "macromedia.com", "google.com");
var uma_url;
```

Em seguida, nós vamos fazer um loop pela array, abrindo cada uma das URLs e esperando que o usuário clique no OK da caixa de alerta.

```
for (loop = 0; loop < array_url.length; loop++)
{

    // monta os endereços das URLs
    uma_url = "http://www." + array_url[loop];

    // abre uma janela
    var nova_janela=open(uma_url,"nova_janela","width=400,height=400");

    // espera pelo clique para a troca
    alert("Clique ok para ver o próximo site");
}
```

Existem algumas coisas interessantes sobre este loop. Primeiro, note que o loop vai de 0 para alguma coisa chamada `array_url.length`. Colocando o `.length` depois do nome da array você tem o número de elementos da array. Lembre-se que este número não é o mesmo número de indexação de elementos, por exemplo, se você tiver 3 elementos numa array, seu último elemento não vai ser [3], e sim [2], pois o primeiro elemento é na verdade o [0]. Se você receber um erro de “objeto não encontrado” e tiver uma array no seu código, tem uma grande chance de você ter confundido o número de indexação de elementos pelo número de elementos da array.

Note que o `.length` no final da array é uma propriedade de um objeto, bem como em um `window.open()` o `.open` é um objeto do `window`.

Note também que para criar uma array utilizou-se a indicação “`new Array`”

```
var array_url = new Array ("proway.com.br", "macromedia.com", "google.com");
```

Esta indicação “`new (alguma coisa)`” é a maneira de se criar novas instâncias de objetos. Nós não entraremos muito no assunto de objetos neste curso, mas lembre-se que sempre que você ver a palavra “`new`” alicada em um script, é porque um objeto novo está sendo criado.

A primeira linha do loop apenas cria uma variável que vai conter uma string.

```
uma_url = "http://www." + array_url[loop];
```

Na primeira vez que o loop for executado, o valor da variável do loop é 0. O primeiro elemento da `array_url` é a string “`proway.com.br`”. Então, durante a primeira execução do loop, a variável `uma_url` vai ser igual a string “`http://www.proway.com.br`”.

A próxima linha do loop abre uma janela com aquela URL:

```
var nova_janela=open(uma_url,"nova_janela","width=400,height=400");
```

Já que a cada vez que ela abre uma janela, ela abre com o mesmo nome (`janela_nova`), as URLs serão abertas sempre na mesma janela, nos poupando de janelas múltiplas abertas.

A terceira linha do loop simplesmente nos mostra uma caixa de alerta e espera que o usuário clique em OK para dar continuidade com a próxima rodada do loop.

```
alert("Clique ok para ver o próximo site");
```

Nesse caso, se esse `alert` não existisse, as 3 URLs seriam mostradas uma depois da outra instantaneamente, e você acabaria somente conseguindo ver o Google aberto.

7. Arrays e o Modelo de Objeto de Documento

Vamos testar este código:

```
<html>
<head>
<title>Página de teste de JavaScript</title>
</head>
<body>

 
<br>
<a href="#"
onClick="var muda=prompt('Qual imagem devo mudar? (0 ou 1)?','');
window.document.images[muda].src='imgs/c.gif';">muda</a>

</body>
</html>
```

Neste exemplo, você terá duas imagens e um link para escolher qual delas deve ser trocada. Quando você clica no link “muda”, uma caixa de diálogo lhe pergunta qual delas você vai querer trocar, 0 ou 1.

A primeira linha do script pega o número do usuário, e a segunda faz a mudança. Antes nós fazíamos trocas de imagens da seguinte forma:

```
document.nome_da_imagem.src = 'outra_imagem.gif';
```

Mas para isso, cada imagem deveria ter um nome. Se você não sabe o nome da imagem que você quer trocar, mas você sabe em que posição ela está na ordem da página HTML, você pode se referir à ela pelo número MDO. A primeira imagem de um documento é a `document.images[0]`. a segunda é `document.images[1]`, e assim por diante.

Se você quiser saber quantas imagens existe em um documento, você pode descobrir utilizando a propriedade `.length` do array de imagens: `document.images.length`. Nesse caso, se você quiser trocar todas as imagens de um site por uma imagem de espaçamento, você poderia fazer isso:

```
for (var loop = 0; loop < document.images.length; loop++)
{
    document.images[loop].src = 'spacer.gif';
}
```

8. Funções

As funções são a última matéria importante que você deve aprender sobre JavaScript antes de começar a fazer scripts. Todas as linguagens de programação tem funções. As funções, ou *sub-rotinas*, como também são chamadas, são pedacinhos de JavaScript que você pode chamar sem ter que reescrevê-las sempre.

Digamos que você quer cronometrar o seu tempo de leitura com um link no começo e um no final que lhe diz o seu tempo.

```
<a href="#" onClick="
  var a_data = new Date();
  var a_hora = a_data.getHours();
  var o_minuto = a_data.getMinutes();
  var o_segundo = a_data.getSeconds();
  var o_horario = a_hora + ':' + o_minuto + ':' + o_segundo;
  alert('Agora são: ' + o_horario);">que horas são?</a>
```

Os detalhes de como esse pequeno javascript funciona não são importantes agora. O que importa é que você não queria que este link ficasse tão grande, pois se você quiser aplicá-lo várias vezes na página, você vai ter que copiar e colar esse bloco grande de código, e caso resolva editá-lo algum dia, terá que achá-lo em suas várias cópias pela página.

Ao invéz disso, você poderia criar uma função com este script e chamá-la sempre que for preciso.

9. Funções sem parâmetro

Vamos ver como ficaria a função de ver as horas:

```
<html>
<head>
<title>Página de teste de JavaScript</title>

<script language="JavaScript">
<!--

function queHorasSao()
{
    var a_data = new Date();
    var a_hora = a_data.getHours();
    var o_minuto = a_data.getMinutes();
    var o_segundo = a_data.getSeconds();
    var o_horario = a_hora + ':' + o_minuto + ':' + o_segundo;
    alert('Agora são: ' + o_horario);
}

-->
</script>

</head>
<body>

<a href="#" onClick="queHorasSao(); return false;">Que horas são?</a>

</body>
</html>
```

Neste código, nós criamos uma função chamada `queHorasSao()`. Para chamá-la de um link, foi utilizado um `onClick`, que executa e mostra a caixa de alerta com o horário.

Você viu algo semelhante no começo do curso quando fizemos este exemplo:

```
<a href="#" onClick="alert('Olá!'); return false;">Oi, tudo bem?</a>
```

Isso chamava o método `alert` de um link. A função é exatamente como um método, a diferença é que o método é ligado a um objeto. Neste caso, o `alert` é um objeto do `window`.

Mas vamos analisar então como funciona uma função:

Para definir uma função em javascript, utilize o seguinte formato:

```
function NomeDaFunção(lista de parâmetros)
{
    indicações
}
```

As regras para nomear uma função são as mesmas para as variáveis. O primeiro caractere deve ser uma letra ou um underscore, nunca um número. O resto dos caracteres podem ser letras, números ou traços. Além disso, você tem que se certificar de não nomear uma função com o mesmo nome de uma variável. É interessante nesse caso utilizar outro tipo de nomeação para funções. Por exemplo, estamos usando underscores para nomear variáveis, então vamos neste curso utilizar letras maiúsculas para funções.

Após o nome da função, nós teríamos uma listagem de parâmetros. Esta função em especial não tem parâmetros, então veremos isso somente no próximo exemplo.

Logo após os parâmetros vem o corpo da função. Aqui vai a lista de indicações que você quer executar sempre que a função for chamada.

Como nós vamos utilizar esta função de horário para os próximos exemplos, vamos ver como ela funciona:

```
var a_data = new Date();
```

Esta primeira linha pega um novo objeto `Date()`. Assim como você deve criar uma array nova utilizando o `new` array, você precisa especificar um `Date()` novo antes de descobrir que horas são.

E para conseguir as informações necessárias, você deve utilizar as propriedades deste objeto:

```
var a_hora = a_data.getHours();  
var o_minuto = a_data.getMinutes();  
var o_segundo = a_data.getSeconds();
```

Se você estiver se perguntando como descobrir quais os métodos/propriedades de objetos que existem e como você chegaria a conclusão de que poderia montar um script desses, você deve acessar uma biblioteca JavaScript. Nós vamos explicar quantos objetos forem possíveis neste curso, mas saiba que existem muitos mais. Um lugar em que você pode procurá-los é neste link:

<http://devedge.netscape.com/library/manuals/2000/javascript/1.5/reference/ix.html>

O resto da função já é bem clara para você agora. Ela monta os números colhidos pelas variáveis, os transforma em uma string e chama uma caixa de alerta com a string. Note que você pode chamar funções de dentro de funções. veremos mais disso.

Agora, se você já brincou bastante com o link, talvez você tenha percebido que, as vezes, o horário aparece como, por exemplo, 12:14:4. O problema aqui é que o método `getSeconds()` retorna um número. E se forem 12:14:04, o `getSeconds` vai retornar o valor 4. Então quando nós montamos a concatenação da string `o_minuto + ":" + o_segundo`, nós temos 14:4, ao invés de 14:04. Existe uma maneira simples de resolver isso, e seria fazendo uma nova função que concerta os minutos e os segundos, se eles precisarem ser concertados. Essa nova função vai demonstrar parâmetros e valores retornados, duas partes cruciais das funções.

10. Parâmetros e valores retornados

No exemplo anterior, nós tivemos um problema quando os métodos `getMinutes` e o `getSeconds` do objeto `Date()` retornavam valores menores que 10. Nós queríamos que eles fossem mostrados como 04, e não apenas 4.

Uma solução seria:

```
var the_minute = the_date.getMinutes();
if (the_minute < 10)
{
    the_minute = "0" + the_minute;
}

var the_second = the_date.getSeconds();
if (the_second < 10)
{
    the_second = "0" + the_second;
}
```

Isso funcionaria perfeitamente. Note que no caso, nós estamos digitando o mesmo código duas vezes.

Seria mais prático se você escrevesse uma função para isso, para poupar tempo de programação:

```
function arrumaNumero(o_numero)
{
    if (o_numero < 10)
    {
        o_numero = "0" + o_numero;
    }
    return o_numero;
}
```

A função `arrumaNumero` tem apenas um parâmetro, chamado "`o_numero`". Um parâmetro é apenas uma variável que é configurada quando a função é chamada. Neste caso, nós chamaríamos a função desta forma:

```
var variavel_arrumada = arrumaNumero(4);
```

O parâmetro `o_numero` é configurado para 4 na função. O corpo do `arrumaNumero` deve agora fazer sentido. Ele diz "se a variável `o_numero` é menor que 10, coloque um 0 na frente". A única diferença é o comando que retorna: `return o_numero`. O comando `return` é utilizado apenas quando você diz algo como

```
var uma_variavel = umaFuncao();
```

Aqui, a variável `uma_variavel` recebe o valor retornado pela função `umaFuncao()`. Na função `arrumaNumero`, eu escrevi: `return o_numero`. Isso faz com que eu saia da função definindo o valor de `o_numero`. Então, se eu escrever:

```
var variavel_certa = arrumaNumero(4);
```

...o número vai ser inicialmente configurado para 4, mas como a função diz que se o número for menor que 10, deve ter um 0 na frente, o valor da `variavel_certa` será 04.

Para incluir o `arrumaNumero` na função original `queHorasSao()`, faça o seguinte:

```
function queHorasSao()
{
    var a_data = new Date();
    var a_hora = a_data.getHours();
    var o_minuto = a_data.getMinutes();
    var minuto_certo = arrumaNumero(o_minuto);
    var o_segundo = a_data.getSeconds();
    var segundo_certo = arrumaNumero(o_segundo);
    var o_horario = a_hora + ':' + minuto_certo + ':' + segundo_certo;
    alert('Agora são: ' + o_horario);
}
```

E dessa forma, você terá um horário certo mostrado na caixa de alerta. Para aplicar esta função em sua página de teste, não esqueça de colocar a função `arrumaNumero` logo acima, assim:

```
<html>
<head>
<title>Página de teste de JavaScript</title>

<script language="JavaScript">
<!--

function arrumaNumero(o_numero)
{
    if (o_numero < 10)
    {
        o_numero = "0" + o_numero;
    }
    return o_numero;
}

function queHorasSao()
{
    var a_data = new Date();
    var a_hora = a_data.getHours();
    var o_minuto = a_data.getMinutes();
    var minuto_certo = arrumaNumero(o_minuto);
    var o_segundo = a_data.getSeconds();
    var segundo_certo = arrumaNumero(o_segundo);
    var o_horario = a_hora + ':' + minuto_certo + ':' + segundo_certo;
    alert('Agora são: ' + o_horario);
}

-->
</script>

</head>
<body>

<a href="#" onClick="queHorasSao(); return false;">Que horas são?</a>

</body>
</html>
```

11. Funções com mais de um parâmetro

Vamos testar o seguinte código:

```
<html>
<head>
<title>Página de teste de JavaScript</title>

<script language="JavaScript">
<!--

var programador = new Array("Joao","Maria","Cristiano","Flavio","Alan","Kelvin");
var gosto_de_cafe = new Array("Maria","Roberto","Ronei","Jonas","Alan","Kelvin");
var gosto_de_dancar = new Array("João","John Travolta","Maria");
var feliz = new Array("Joaquim","Maria","Alexandra","Alan");
var triste = new Array("Kelvin","Joao","Flavio");

function interseccaoArray(nome_interceccao, array_1, array_2)
{
    var a_lista = "";
    for (loop_1 = 0; loop_1 < array_1.length; loop_1++)
    {
        for (loop_2 = 0; loop_2 < array_2.length; loop_2++)
        {
            if (array_1[loop_1] == array_2[loop_2])
            {
                a_lista = a_lista + array_1[loop_1] + ", ";
            }
        }
    }
    alert("Lista de" + nome_interceccao + ": " + a_lista);
}
-->
</script>

</head>
<body>

<a href="#" onClick="interseccaoArray('programadores que bebem
café',programador,gosto_de_cafe); return false;">Lista de programadores que gostam
de café.</a>

</body>
</html>
```

No começo do script nós definimos uma sequência de arrays com nomes de programadores, de pessoas que gostam de café, que gostam de dançar, que estão felizes e que estão tristes. Logo abaixo nós temos uma função que faz a intersecção entre estas arrays para compará-las. Agora fica bem fácil definir quem são os programadores que gostam de café.

Esta é uma função com 3 parâmetros: uma string que representa qual vai ser o nome da intersecção, a primeira array e a segunda array. Para descobrir o nome dos programadores que gostam de dançar é igualmente fácil.

```
<a href="#" onClick="interseccaoArray('programadores que gostam de
dançar',programador,gosto_de_dancar); return false;">Lista de programadores que
gostam de dançar.</a>
```

No caso, note que por mais que o John Travolta goste de dançar, ele não é programador, por isso não aparece na lista.

Vamos fazer uma breve análise da função de intersecção de arrays:

```
function interseccaoArray(nome_interseccao, array_1, array_2)
```

Isto define a função chamada de `interseccaoArray` com três parâmetros. Exatamente como no exemplo anterior. Por isso, quando a função precisa ser chamada, chamamos assim:

```
interseccaoArray('programadores que sabem dançar',programadores,gosto_de_dancar);
```

Com estas indicações, nós definimos que:

- `nome_interseccao` = `programadores que sabem dançar`
- `array_1` - `programadores`
- `array_2` - `gosto_de_dancar`

12. Revisão

Com isso, nós terminamos com a sintaxe do JavaScript. Estes foram os tópicos abordados:

- **while loops**

Como fazer um loop `while` executar indefinidamente ou um número específico de vezes.

- **for loops**

Como funcionam os loops `for`

- **Arrays**

O que são arrays e como iniciá-las e configurá-las

- **Arrays and loops**

Como utilizar loops com arrays utilizando a propriedade `.length`

- **Arrays em MDO**

Como utilizar arrays para afetar imagens em sua página

- **Functions**

Como funcionam as funções e como escrever e chamar uma

- **Parameters e return values**

Funções ficam mais interessantes se você pode mudar seus comportamentos baseados em resultados de outros scripts/interações com o usuário.

Lição 5

1. Introdução a Formulários

Formulários são partes da especificação HTML 1.0. Sabe-se pouco sobre formulários, pois normalmente se acha que a sua única utilização é para uso de scripts de CGI do lado do servidor. Aqui veremos que você pode aplicar JavaScript em formulários sem a necessidade de CGIs do lado do servidor.

Este é um curso de JavaScript. Nós assumimos aqui que você sabe trabalhar com formulários HTML.

A primeira coisa que você deve saber sobre formulários e JavaScript é que os formulários, assim como as imagens, são armazenadas como uma array de objetos, bem como as imagens. Ou seja, assim como você consegue se referir a uma imagem chamando-a por `window.document.images[0]`, você também pode referir-se a um formulário usando `window.document.forms[0]`.

E da mesma forma que você pode nomear imagens para chamá-las, você também pode nomear formulários. Como este:

```
<form name="meu_form">
<input type="text" name="campo_texto" size="30"
value="Eu tenho nome!">
</form>
```

Que resultaria nisso:

Eu tenho nome!

Você pode se referir a um formulário com qualquer uma dessas opções:

```
var meu_form = window.document.forms[0];

var o_mesmo_form = window.document.meu_form;
```

Mesmo que achar um formulário seja útil, você pode precisar chamar por um elemento dentro do formulário, como aquele campo de texto do exemplo anterior.

2. Manipulando valores de campos de texto

Vamos ver um exemplo. Adicione este código no corpo de uma página HTML:

```
<form name="meu_form">
<input type="text" name="meu_texto" value="Você está feliz?">
</form>

<a href="#" onMouseOver="window.document.meu_form.meu_texto.value=
'Então bata palmas!';">Sim!</a> &nbsp; &nbsp; &nbsp;
<a href="#" onMouseOver="window.document.meu_form.meu_texto.value=
'Pobre criatura.';">Não</a>
```

Você deve receber uma página com o seguinte conteúdo:

[Sim!](#) [Não](#)

Agora, se você passar o mouse por cima dos links, você vai ver o valor do campo acima mudando para as respostas “Então bata palmas!” ou “Pobre criatura.”, dependendo de em qual link você passar o mouse.

Note que o JavaScript disto é um evento `onMouseOver` normal:

```
onMouseOver="window.document.meu_form.meu_texto.value='Então bata palmas!';"
```

Ali ele diz: “procure nesta janela, neste documento, alguma coisa chamada `meu_form`, e dentro disso algo chamado `meu_texto`, e mude o seu valor para ‘Então bata palmas!’”. Da mesma forma que você muda o `src` de uma imagem.

Outros elementos de formulário também podem ser mudados, por exemplo:

[Primeira](#) [Segunda](#)

Vamos ver o código deste formulário na página seguinte:

```
<html>
<head>
<title>Página de teste de JavaScript</title>
</head>

<script language=JavaScript>
<!--

var primeira = "We get some rules to follow\nThat and this\nThese and those\nNo one
knows"

var segunda = "We get these pills to swallow\nHow they stick\nIn your THROAT\nTaste
like gold

-->
</script>

<body>

<form name="outro_form">
<textarea name="minha_caixa" rows=4 cols=40>
Se você passar o mouse por cima dos links, você verá dois trechos da música "No one
Knows" da banda Queens of Stone Age:
</textarea>
</form>

<a href="#" onMouseOver="window.document.outro_form.minha_caixa.value=
primeira;">Primeira</a>
<a href="#" onMouseOver="window.document.outro_form.minha_caixa.value=
segunda;">Segunda</a>

</body>
</html>
```

Note que o formulário tem um nome, `outro_form`, e a caixa de texto também tem um, `minha_caixa`. Os links são basicamente os mesmo do exemplo anterior, mas desta vez, nós armazenamos as strings dentro de tags `<script>` no cabeçalho da página em variáveis.

Note nas variáveis no cabeçalho que foram adicionados alguns “\n” durante a string. Esta é a forma mais básica de quebra de linha utilizada em programação.

O resultado de passar o mouse por cima dos links então seria esse:



3. Eventos de campos de texto

Os campos de texto entendem os eventos `onBlur` (ao desfocar), `onFocus` (ao focar) e `onChange` (ao mudar). O evento `onFocus` ocorre quando alguém clica dentro de um campo de texto. O `onBlur` acontece quando alguém sai de um campo de texto, seja clicando em algum outro campo como clicando fora dele ou pressionando a tecla "tab". E o evento `onChange` acontece quando alguém muda o que tem dentro de um campo de texto e logo depois sai dele.

Por exemplo:

```
<html><head><title>Página de teste de JavaScript</title>

<script language="JavaScript">
<!--

function writeIt(a_palavra)
{
    var palavra_que_retorna = a_palavra + "\n";
    window.document.form.textarea.value += palavra_que_retorna;
}

-->
</script>

</head>
<body>

<form name="form" onSubmit="return false;">
<input type="text" name="text" onFocus="writeIt('focus');"
onBlur="writeIt('blur');" onChange="writeIt('change');"><br><br>

<textarea name="textarea" rows=10 cols=30>

</textarea>
</form>

</body></html>
```

Isso resultaria em dois campos de texto. Experimente executar este código e clicar de um campo para o outro.

Você deve estar reconhecendo todos estes comandos agora. Na primeira linha do script no cabeçalho nós criamos uma função chamada `writeIt` (escreve) com um parâmetro, que é o "a_palavra". Logo em seguida nós declaramos a variável `palavra_que_retorna = a_palavra + "\n"`. Isso significa que o valor da variável `palavra_que_retorna` terá uma quebra de linha no final.

O truque está na linha seguinte:

```
window.document.form.textarea.value += palavra_que_retorna;
```

Aqui é dito que a `palavra_que_retorna` vai ter o valor do campo de texto MAIS a nova variável. Seria o mesmo que dizer:

```
window.document.form.textarea.value = window.document.form.textarea.value +  
palavra_que_retorna;
```

Depois nós definimos dentro da caixa de texto o que será escrito nela, de acordo com o comportamento do usuário com os campos deste formulário.

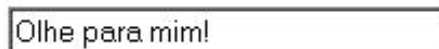
```
<input type="text" name="text"  
onFocus="writeIt('focus');"  
onBlur="writeIt('blur');"  
onChange="writeIt('change');">
```

Vamos ver agora os métodos que podemos utilizar com os campos de formulários: `blur()`, `focus()`, e `select()`.

Aqui vai um exemplo (coloque no corpo de um código HTML normal):

```
<form name="form">  
<input type="text" name="campo" size=30 value="Olhe para mim!">  
</form>  
  
<a href="#" onClick="window.document.form.campo.focus();">Clique em mim para  
focar</a> &nbsp;   <a href="#" onClick="window.document.form.campo.select();">Clique em mim para  
selecionar</a>
```

Este código resultará em uma página parecida com isso:



[Clique em mim para focar](#) [Clique em mim para selecionar](#)

No caso, se você clicar no primeiro link, o campo de texto acima estará em foco (ou seja, o cursor para digitação estará lá), e se você clicar no segundo link, o texto do campo será selecionado. Isto é feito com a função `onClick` contida dentro dos links:

```
onClick="window.document.form.campo.focus();"
```

Ali diz “ao clicar, ache dentro de `window`, dentro do seu documento, algo chamado `form`, e dentro disso algo chamado `campo`, e foque nele”. O segundo link é a mesma coisa, só que ao invés de `focus`, é `select`.

4. Eventos de Formulário

Formulários são objetos. Eles tem seus próprios métodos, propriedades e eventos.

O evento de formulário `onSubmit`, por exemplo, é chamado em duas situações: se o usuário clica em um botão de envio, ou se ele teclar “enter” em um campo de texto. Se essas ações não estiverem definidas em algum lugar, o seu JavaScript pode se comportar de uma maneira não esperada.

Caso você queira, por exemplo, utilizar um botão com alguma ação, normalmente o botão quando clicado recarrega a página. Para que a página não seja recarregada, você pode utilizá-la desta forma:

```
<form onSubmit="return false;">
<input type="submit" value="Submit">
</form>
```

O `return false` é a maneira que o JavaScript tem de impedir o browser de fazer o que ele geralmente faz.

Por exemplo, vamos testar este código:

```
<html>
<head>
<title>Página de teste de JavaScript</title>
<script>
<!--
function sabeJavaScript()
{
    var quem_e = window.document.campo_de_entrada.sabe_js.value;
    quem_e = quem_e + ' sabe JavaScript!';
    window.document.campo_de_entrada.sabe_js.value = quem_e;
}
-->
</script>
</head>
<body>

<form name="campo_de_entrada" onSubmit="sabeJavaScript(); return false;">
<b>Quem sabe JavaScript? </b>
<input type="text" name="sabe_js" size="30">
</form>

</body>
</html>
```

Quando você digitar algum nome no campo desta página e teclar “enter”, o evento `onSubmit` é chamado. Ele executa a função “`sabeJavaScript`”, que muda o valor do campo de texto, mas não recarrega a página por causa do `return false` do formulário.

Experimente tirar o `return false` e ver o que acontece. A página será recarregada e os parâmetros do script serão passados para a URL.

5. Checkboxes

Os Checkboxes tem uma propriedade principal que nos interessa: `checked` (marcado).

- ☒ Este checkbox está marcado
- ☐ Este Checkbox não está marcado

Se você tem um formulário chamado "form1" e um checkbox chamado "check01" você pode conferir se ele está marcado ou não assim:

```
var esta_marcado = window.document.form01.check01.checked;
if (esta_marcado == true)
{
    alert("Sim, está marcado!");
} else {
    alert("Não, não está marcado.");
}
```

Como você pode ver, se um checkbox está marcado, a propriedade será `true` (verdadeira). Se você quisesse conferir se o checkbox NAO está marcado, você usaria `false`.

Assim como você pode conferir se está marcado, você também pode mudar esta propriedade.

```
<form name="form_1">
<input type="checkbox" name="check_1">Checkbox 1
</form>

<a href="#" onClick="window.document.form_1.check_1.checked=true;
return false;">Clique para marcar</a><br>

<a href="#" onClick="window.document.form_1.check_1.checked=false;
return false;">Clique para desmarcar</a><br>

<a href="#" onClick="alert(window.document.form_1.check_1.checked);
return false;">Clique para saber o valor</a>
```

Vamos ver um outro exemplo:

```
<html>
<head>
<title>Página de teste de JavaScript</title>
<script>
<!--

function interruptor()
{
    var a_caixa = window.document.form_2.check_1;
    var o_botao = "";
    if (a_caixa.checked == false) {
        document.bgColor='black';
        alert("Ficou escuro! Acende a luz!!");
    } else {
        document.bgColor='white';
        alert("Obrigado!");
    }
}

-->
</script>

</head>
<body>

<form name="form_2">
<input type="checkbox" name ="check_1" checked="true"
onClick="interruptor();">O interruptor de luz.
</form>

</body>
</html>
```

Você deve ter entendido como essa função funciona.

Ao clicar no checkbox, a função `interruptor()` que está no evento `onClick` é chamada, e o `if` da função se encarrega de ver o que deve fazer com o valor fornecido pelo checkbox.

6. Botões rádio

Botões de radio são basicamente a mesma coisa que checkboxes, a diferença é que você não consegue marcar/desmarcar. Para isso você precisa escolher outra opção, para que a anterior se desmarque.

Com isso em mente, nós podemos repensar aquele interruptor de luz:

```
<html><head><title>Página de teste de JavaScript</title>
<script>
<!--

function desligaLuz()
{
    var o_radio = window.document.form_1.radio_1;
    if (o_radio.checked == true)
    {
        window.document.form_1.radio_2.checked = false;
        document.bgColor='black';
        alert("Ficou escuro, acende a luz!");
    }
}

function ligaLuz()
{
    var o_radio = window.document.form_1.radio_2;
    if (o_radio.checked == true) {
        window.document.form_1.radio_1.checked = false;
        document.bgColor='white';
        alert("Obrigado!");
    }
}

-->
</script>

</head>
<body>

<form name="form_1">
<input type="radio" name ="radio_1" onClick="desligaLuz();">Desliga a luz
<input type="radio" name ="radio_2" onClick="ligaLuz();" checked>Liga a luz
</form>

</body>
</html>
```

A grande diferença aqui é na linha `window.document.form_1.radio_1.checked =false`, que diz que o radio button que estava ligado antes deve ser desligado. Na verdade, o botão de rádio será automaticamente desligado, mas se essa linha não for adicionada, a variável `o_radio` não vai ficar sabendo disso.

7. Selects

Os Selects são basicamente de dois tipos: Pulldown e List:

```
<form name="form_1">
<b>Select tipo Pulldown:</b><br>
<select name="pulldown_1" size=1>
<option>azul</option>
<option>vermelho</option>
<option>amarelo</option>
<option>verde</option>
<option>rosa</option>
<option>lilás</option>
</select>

<br><br>
<b>Select tipo List:</b><br>
<select name="list_1" size=3>
<option>azul</option>
<option>vermelho</option>
<option>amarelo</option>
<option>verde</option>
<option>rosa</option>
<option>lilás</option>
</select>
</form>
```

Este código gera algo com esta aparência:

Select tipo Pulldown:



Select tipo List:



Note que a única diferença no código para montar os dois é o número de linhas. O Menu pulldown só é pulldown porque tem apenas uma linha.

O próprio select pode ser nomeado como qualquer elemento de formulário. No nosso caso, eles foram nomeados de pulldown_1 e list_1. Mas as opções não são nomeadas, o que pode dificultar um pouco a aplicação de JavaScript.

Neste caso, nós vamos encarar os Selects como arrays. Cada opção passa a ser um item de uma array, numeradas a partir do número [0], como já havíamos visto neste curso. No caso, se você quisesse mudar o texto da segunda opção do select você usaria isto:

```
window.document.form_1.pulldown_1.options[1].text = 'laranja';
```

Além da propriedade `options` (opções) do `select`, existe também uma propriedade chamada `selectedIndex`. No caso, quando uma das opções do `select` estiver selecionada, o valor do `selectedIndex` será o mesmo valor do seu item na array.

Experimente adicionar este link abaixo dos seus `selects` (da página anterior):

```
<a href="#" onClick= "alert(window.document.form_1.list_1.selectedIndex);return false;">Quem é o index?</a>
```

Antes de clicar, selecione um dos itens do `select` (o segundo, que está em lista). Você receberá uma caixa de alerta com o valor da opção selecionada.

Você também pode utilizar a mesma idéia para selecionar algum dos itens, por exemplo:

```
<a href="#" onClick= "window.document.form_1.list_1.selectedIndex = 3;return false;">Selecione a 4ª opção.</a>
```

Com este link, eu terei a 4ª opção do `select` selecionada. No caso do exemplo que estamos usando, a cor verde seria selecionada.

8. Método de formulário onChange em Selects

Vamos ao exemplo:

```
<html>
<head>
<title>Página de teste de JavaScript</title>
<script>
<!--

var quentes = new Array("vermelho","laranja","amarelo");
var frias = new Array("azul", "verde", "lilás");
var neutras = new Array("cinza", "branco", "preto");

function mudaOpcao(nome_array)
{
    var numbers_select = window.document.o_form.as_cores;
    var a_array = eval(nome_array);
    mudaTexto(window.document.o_form.as_cores, a_array);
}

function mudaTexto(cores, a_array)
{
    for (loop=0; loop < cores.options.length; loop++)
    {
        cores.options[loop].text = a_array[loop];
    }
}

-->
</script>

</head>
<body>

<form name="o_form">
<select name="cores"
onChange="mudaOpcao(window.document.o_form.cores.options[selectedIndex].text);return
false;">
<option selected>quentes</option>
<option>frias</option>
<option>neutras</option>
</select>

<select name="as_cores" multiple>
<option>vermelho</option>
<option>laranja</option>
<option>amarelo</option>
</select>
</form>

</body>
</html>
```

Este é um JavaScript um pouco mais complicado. Vamos analisá-lo

No corpo do código, você tem dois selects: um pulldown e um list. O pulldown tem um método `onChange` que chama a função `mudaOpcao`. Esta função que está definida no cabeçalho tem somente um parâmetro: o tipo de cor selecionada (no caso, quente, fria ou neutra).

No cabeçalho, a primeira que foi feita foi definir as arrays dos valores de cara tipo de cor escolhida.

```
var quentes = new Array("vermelho","laranja","amarelo");
var frias = new Array("azul", "verde", "lilás");
var neutras = new Array("cinza", "branco", "preto");
```

As arrays estão nomeadas com o mesmo nome que as opções do select, logo veremos por quê.

Agora, vamos ver a primeira função:

```
function mudaOpcao(nome_array)
{
    var numbers_select = window.document.o_form.as_cores;
    var a_array = eval(nome_array);
    mudaTexto(window.document.o_form.as_cores, a_array);
}
```

Esta função tem apenas um parâmetro: `home_array`. Se você escolher no menu pulldown a opção “neutras”, este será o valor da variável `home_array`.

A segunda linha configura uma variável para se referir ao segundo elemento do form, o select list.

Nesta linha nós somos apresentados a algo novo: `eval()`. Este método vai igualar a variável nova, `a_array`, com a variável da escolha do usuário, `nome_array`.

Na terceira linha desta função nós chamamos outra função chamada `mudaTexto()`. Esta função faz mais uma sequência de indicações que de fato muda os textos do segundo select.

```
function mudaTexto(cores, a_array)
{
    for (loop=0; loop < cores.options.length; loop++)
    {
        cores.options[loop].text = a_array[loop];
    }
}
```

Esta função pede dois parâmetros: uma referência ao select e uma array. No caso, `cores` é o nome do select e `a_array` é a variável definida na função anterior que o usuário escolheu, igualada a array de mesmo nome.

Vamos ver como esse loop funciona:

Na primeira rodada do loop, a variável `loop` é igual a 0.

O corpo do loop então lê: `cores.options[0].text` - `a_array[0]`; ou seja, ele defini que a primeira opção do select será igual ao primeiro elemento da array.

Este processo se repetirá pelo número de opções que existem no select, ou seja, três vezes.

9. revisao

Nesta lição os seguintes assuntos foram abordados:

- **Formulários e seus elementos podem ser nomeados**

Você pode se referir a elementos de formulários desta forma:
`window.document.form_name.element_name`

- **Campos de texto e caixas de texto tem a propriedade value (valor)**

Para se referir ao seus valores, usamos isto: `window.document.form_name.element_name.value;` e podemos configurar o valor usando os eventos `focus()`, `blur()` e `change()`.

Quando você clica em um campo ou caixa de texto, o evento **focus** é ativado. Clicando fora de um destes elementos, o evento **blur** é ativado. Se você teclar enter ou utilizar a tecla tab para mudar de campo depois de alterar os dados dele, o evento **change** é ativado.

- **Formulários tem o evento onSubmit()**

Para impedir que um form recarregue a página quando você clica em um botão de envio, utilize `onSubmit="return false;"` dentro da tag `<form>`.

- **Checkboxes e radio buttons tem a propriedade checked.**

Você pode marcar e desmarcar checkboxes e botões de radio mudando sua propriedade `checked`.

- **Checkboxes e radio buttons tem o evento onClick**

Assim como os links, você pode aplicar funções quando o usuário clica em um checkbox ou radiobox.

- **Os Selects tem a propriedade option que é uma array das opções dele.**

Bem como uma array, você pode definir a quantidade de opções de um select por este método:

```
window.document.nome_do_form.nome_do_select.options.length
```

- **Pode-se mudar o texto de uma opção modificando a sua propriedade.**

Para mudar o texto de uma opção, você diz

```
window.document.nome_do_form.nome_do_select.options[0].text='Texto novo';
```

- Você pode descobrir qual opção foi selecionada utilizando a propriedade `selectedIndex` de um elemento select, desta forma:

```
window.document.nome_do_form.nome_do_select.selectedIndex
```

O valor retornado é a posição da opção no select, contando a partir de [0], como uma array. Sabendo o número, você pode utilizar o seguinte código para descobrir o que existe nesta posição:

```
window.document.nome_do_form.nome_do_select.options[aquele_numero].text
```

- **Selects tem o evento onChange**

Se alguém selecionar alguma outra opção depois de já ter uma selecionada, o evento `onChange` é chamado.

- **Você pode mudar a URL de uma janela utilizando**

```
window.location.replace('nova URL');
```

(ok, nós não vimos isso, mas experimente fazer um select que muda a página em que você está!)