

**Universidade do Sul de Santa Catarina**

# **Linguagens de Programação III**

**Disciplina na modalidade a distância**

**UnisulVirtual**  
A sua universidade a distância

Universidade do Sul de Santa Catarina

# **Linguagens de Programação III**

Disciplina na modalidade a distância

Palhoça  
UnisulVirtual  
2011

# Créditos

## Universidade do Sul de Santa Catarina – Campus UnisulVirtual – Educação Superior a Distância

Avenida dos Lagos, 41 – Cidade Universitária Pedra Branca | Palhoça – SC | 88137-900 | Fone/fax: (48) 3279-1242 e 3279-1271 | E-mail: cursovirtual@unisul.br | Site: www.unisul.br/unisulvirtual

### Reitor Unisul

Ailton Nazareno Soares

### Vice-Reitor

Sebastião Salésio Heerdt

### Chefe de Gabinete da Reitoria

Willian Máximo

### Pró-Reitora Acadêmica

Miriam de Fátima Bora Rosa

### Pró-Reitor de Administração

Fabian Martins de Castro

### Pró-Reitor de Ensino

Mauri Luiz Heerdt

### Campus Universitário de Tubarão

Diretora

Milene Pacheco Kindermann

### Campus Universitário da Grande Florianópolis

Diretor

Hércules Nunes de Araújo

### Campus Universitário UnisulVirtual

Diretora

Jucimara Roesler

### Equipe UnisulVirtual

#### Diretora Adjunta

Patrícia Alberton

#### Secretaria Executiva e Cerimonial

Jackson Schuelter Wiggers (Coord.)  
Marcelo Fraiberg Machado  
Tenille Catarina  
Vanessa Guimaraes Franceschi

#### Assessoria de Assuntos Internacionais

Murilo Matos Mendonça

#### Assessoria DAD - Disciplinas a Distância

Patrícia da Silva Meneghel (Coord.)  
Carlos Alberto Areias  
Franciele Arruda Rampelotti  
Luiz Fernando Meneghel

#### Assessoria de Inovação e Qualidade da EaD

Dênia Falcão de Bittencourt (Coord.)  
Rafael Bavaresco Bongiolo

#### Assessoria de Relação com Poder Público e Forças Armadas

Adenir Siqueira Viana

#### Assessoria de Tecnologia

Osmar de Oliveira Braz Júnior (Coord.)  
Felipe Jacson de Freitas  
Jefferson Amorim Oliveira  
José Olímpio Schmidt  
Marcelo Neri da Silva  
Phelipe Luiz Winter da Silva

### Priscila da Silva

Rodrigo Battistotti Pimpão

### Coordenação dos Cursos

#### Assistente das Coordenações

Maria de Fátima Martins

#### Auxiliares das coordenações

Fabiâna Lange Patrício  
Tânia Regina Goularte Waltemann

#### Coordenadores Graduação

Adriana Santos Rammé

Adriano Sérgio da Cunha

Aloísio José Rodrigues

Ana Luisa Mühlert

Ana Paula R. Pacheco

Artur Beck Neto

Bernardino José da Silva

Carmen Maria C. Pandini

Catia Melissa S. Rodrigues

Charles Cesconetto

Diva Marília Flemming

Eduardo Aquino Hübner

Eliza B. D. Locks

Fábio Ceretta

Horácio Dutra Mello

Itamar Pedro Beviláqua

Jairo Afonso Henkes

Janaína Baeta Neves

Jardel Mendes Vieira

Joel Irineu Lohn

Jorge Alexandre N. Cardoso

José Carlos N. Oliveira

José Gabriel da Silva

José Humberto D. Toledo

Joseane Borges de Miranda

Luciana Manfroi

Luiz Guilherme B. Figueiredo

Marceli Evangelista Catâneo

Maria Cristina Veit

Maria da Graça Poyer

Mauro Faccioni Filho

Moacir Fogaça

Nélio Herzmann

Onei Tadeu Dutra

Patrícia Fontanella

Raulino Jacó Brüning

Roberto Iunskovski

Rodrigo Nunes Lunardelli

Rogério Santos da Costa

Rosa Beatriz M. Pinheiro

Rose Clér Beche

Sérgio Sell

Tatiana Lee Marques

Thiago Coelho Soares

Valnei Campos Denardin

Victor Henrique Moreira Ferreira

#### Coordenadores Pós-Graduação

Aloísio Rodrigues

Anelise Leal Vieira Cubas

Bernardino José da Silva

Carmen Maria Cipriani Pandini

Daniela Ernani Monteiro Will

Giovani de Paula

Karla Leonora Nunes

Luiz Otávio Botelho Lento

Thiago Coelho Soares

Vera Regina N. Schuhmacher

#### Gerência Administração Acadêmica

Angelita Marçal Flores (Gerente)

Fernanda Farias

#### Gestão Documental

Lamuniê Souza (Coord.)

Clair Maria Cardoso

Janaina Stuart da Costa

Josiane Leal

Marilia Locks Fernandes

Ricardo Mello Platt

### Secretaria de Ensino a Distância

Karine Augusta Zanoni

(Secretária de Ensino)

Giane dos Passos

(Secretária Acadêmica)

Alessandro Alves da Silva

Andréa Luci Mandira

Cristina Mara Shauffert

Djeime Sammer Bortolotti

Douglas Silveira

Fabiano Silva Michels

Felipe Wronski Henrique

Janaína Conceição

Jean Martins

Luana Borges da Silva

Luana Tarsila Hellmann

Maria José Rossetti

Miguel Rodrigues da Silveira Junior

Monique Tayse da Silva

Patrícia A. Pereira de Carvalho

Patrícia Nunes Martins

Paulo Lisboa Cordeiro

Rafaela Fusieger

Rosângela Mara Siegel

Silvana Henrique Silva

Vanilda Liordina Heerdt

### Gerência Administrativa e Financeira

Renato André Luz (Gerente)

Naiara Jeremias da Rocha

Valmir Venício Inácio

#### Financeiro Acadêmico

Marlene Schauffer

Rafael Back

Vilmar Isaurino Vidal

### Gerência de Ensino, Pesquisa e Extensão

Moacir Heerdt (Gerente)

Aracelli Araldi

#### Elaboração de Projeto e Reconhecimento de Curso

Diane Dal Magno

Vanderlei Brasil

#### Extensão

Maria Cristina Veit (Coord.)

#### Pesquisa

Daniela E. M. Will

(Coord. PUIP, PUIC, PIBIC)

Mauro Faccioni Filho

(Coord. Nuvem)

#### Pós-Graduação

Clarissa Carneiro Mussi (Coord.)

#### Biblioteca

Soraya Arruda Waltrick (Coord.)

Paula Sanhudo da Silva

Renan Felipe Cascaes

Rodrigo Martins da Silva

#### Capacitação e Assessoria ao Docente

Enzo de Oliveira Moreira (Coord.)

Adriana Silveira

Alexandre Wagner da Rocha

Cláudia Behr Valente

Elaine Cristiane Surian

Juliana Cardoso Esmeraldino

Simone Perroni da Silva Zigunovas

#### Monitoria e Suporte

Enzo de Oliveira Moreira (Coord.)

Anderson da Silveira

Angélica Cristina Gollo

Bruno Augusto Zunino

Claudia Noemi Nascimento

Débora Cristina Silveira

Elaine Cristina Silveira

Flávia Cristina Silveira

Flávia Cristina Silveira

### Ednéia Araujo Alberto

Francine Cardoso da Silva

Karla F. Wisniewski Desengrini

Maria Eugênia Ferreira Celeghin

Maria Lina Moratelli Prado

Mayara de Oliveira Bastos

Patrícia de Souza Amorim

Poliana Morgana Simão

Priscila Machado

### Relacionamento com o Mercado

Eliza Bianchini Dallanhol Locks

Walter Félix Cardoso Júnior

### Design Visual

Pedro Paulo Alves Teixeira (Coord.)

Adriana Ferreira dos Santos

Alex Sandro Xavier

Alice Demaria Silva

Anne Cristyne Pereira

Diogo Rafael da Silva

Edison Rodrigo Valim

Frederico Trilha

Higor Ghisi Luciano

Jordana Paula Schulka

Nelson Rosa

Patrícia Fragnani de Moraes

### Portal

Rafael Pessi (Coord.)

Luiz Felipe Buchmann Figueiredo

### Comunicação

Marcelo Barcelos

Andreia Drewes

### e-OLA

Carla Fabiana F. Raimundo (Coord.)

Vinicius Ritta de Moura

### Produção Industrial

Francisco Asp (Coord.)

Ana Paula Pereira

Marcelo Bittencourt

### Gerência Serviço de Atenção Integral ao Acadêmico

James Marcel Silva Ribeiro (Gerente)

### Atendimento

Maria Isabel Aragon (Coord.)

Andiara Clara Ferreira

André Luiz Portes

Bruno Ataide Martins

Holdrin Milet Brandao

Jenniffer Camargo

Maurício dos Santos Augusto

Maycon de Sousa Candido

Sabrina Mari Kawano Gonçalves

Vanessa Trindade

Orivaldo Carli da Silva Junior

### Estágio

Jonatas Collaço de Souza (Coord.)

Juliana Cardoso da Silva

Micheli Maria Lino de Medeiros

Priscilla Geovana Pagani

### Prouni

Tatiane Crestani Trentin (Coord.)

Gisele Ierezinha Cardoso Ferreira

Scheila Cristina Martins

Taize Muller

### Gerência de Marketing

Fabiano Céretta (Gerente)

Alex Fabiano Wehrle

Márcia Luz de Oliveira

Sheyla Fabiana Batista Guerrer

Victor Henrique M. Ferreira (África)

**Edson Orivaldo Lessa Junior**

# **Linguagens de Programação III**

Livro didático

Design instrucional  
Flavia Lumi Matuzawa

Palhoça  
UnisulVirtual  
2011

Copyright © UnisulVirtual 2011

Nenhuma parte desta publicação pode ser reproduzida por qualquer meio sem a prévia autorização desta instituição.

### **Edição – Livro Didático**

**Professor Conteudista**  
Edson Orivaldo Lessa Junior

**Design Instrucional**  
Flavia Lumi Matuzawa

**Projeto Gráfico e Capa**  
Equipe UnisulVirtual

**Diagramação**  
Anne Cristyne Pereira  
Daiana Ferreira

**Revisão**  
Letra de Forma

741.6

L63      Lessa Junior, Edson Orivaldo  
             Linguagens de programação III : livro didático / Edson Orivaldo Lessa  
Junior ; design instrucional Flavia Lumi Matuzawa. – Palhoça :  
UnisulVirtual, 2011.  
        462 p. : il. ; 28 cm.

Inclui bibliografia.

1. Desenho gráfico. 2. Artes gráficas. 3. Desenho comercial. I. Matuzawa,  
Flavia Lumi. II. Título.

# Sumário

Apresentação .....	7
Palavras do professor.....	9
Plano de estudo .....	11
<b>UNIDADE 1 - Aspectos do desenvolvimento de aplicações Web .....</b>	<b>17</b>
<b>UNIDADE 2 - JavaScript.....</b>	<b>79</b>
<b>UNIDADE 3 - PHP .....</b>	<b>149</b>
<b>UNIDADE 4 - Sistemas Web com banco de dados (PHP + MYSQL).....</b>	<b>203</b>
<b>UNIDADE 5 - Extensible Markup Language (XML).....</b>	<b>291</b>
<b>UNIDADE 6 - Asynchronous Javascript and XML (AJAX) .....</b>	<b>347</b>
Para concluir o estudo.....	395
Referências .....	397
Sobre o professor conteudista.....	399
Respostas e comentários das atividades de autoavaliação .....	401
Biblioteca Virtual.....	461



# **Apresentação**

Este livro didático corresponde à disciplina **Linguagens de Programação III**.

O material foi elaborado visando a uma aprendizagem autônoma e aborda conteúdos especialmente selecionados e relacionados à sua área de formação. Ao adotar uma linguagem didática e dialógica, objetivamos facilitar seu estudo a distância, proporcionando condições favoráveis às múltiplas interações e a um aprendizado contextualizado e eficaz.

Lembre-se que sua caminhada, nesta disciplina, será acompanhada e monitorada constantemente pelo Sistema Tutorial da UnisulVirtual, por isso a “distância” fica caracterizada somente na modalidade de ensino que você optou para sua formação, pois, na relação de aprendizagem, professores e instituição estarão sempre conectados com você.

Então, sempre que sentir necessidade, entre em contato, pois você tem à disposição diversas ferramentas e canais de acesso, como: telefone, e-mail e o Espaço Unisul Virtual de Aprendizagem, que é o canal mais recomendado, pois tudo o que for enviado e recebido fica registrado para seu maior controle e sua comodidade. Nossa equipe técnica e pedagógica terá o maior prazer em atendê-lo, pois sua aprendizagem é o nosso principal objetivo.

Bom estudo e sucesso!

Equipe UnisulVirtual.



## Palavras do professor



Caro desbravador do conhecimento,

O mundo tecnológico é imenso permanecendo em constante expansão e evolução. Por essa razão, o conhecimento e as técnicas de desenvolvimento de *software* seguem essa linha. Linguagens nascem, envelhecem, algumas morrem outras reestruturam-se e com novas técnicas são reutilizadas. Assim o aprendizado se torna constante, de tal forma que força o desenvolvedor a sempre procurar atualizar-se. A técnica permanece, mas a forma de escrita se altera.

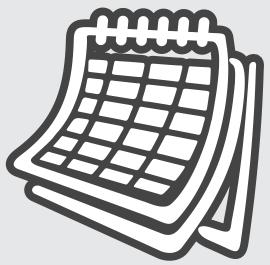
A linguagem de programação nada mais é que um dialeto para que a máquina possa entender quais são as ordens que nós damos. Assim como na vida real, uma maçã é uma maçã em qualquer língua falada pelos homens. A programação não é diferente, já que uma variável sempre será uma variável. A forma de escrever o comando pode e deve se alterar, mas o conceito é o mesmo.

O desenvolvimento Web possui uma característica maravilhosa que nos leva sempre a utilizar diversas técnicas e linguagens para alcançar o melhor resultado. Nesta disciplina, você verá a ponta do *iceberg*. Você será tentado a desbravar esse universo da programação Web. Desvendar seus segredos submersos será algo inesquecível e onde você poderá construir uma sólida carreira em tecnologia.

Bom estudo!

Professor Edson O. Lessa Jr.





## Plano de estudo

O plano de estudos visa a orientá-lo no desenvolvimento da disciplina. Ele possui elementos que o ajudarão a conhecer o contexto da disciplina e a organizar o seu tempo de estudos.

O processo de ensino e aprendizagem na UnisulVirtual leva em conta instrumentos que se articulam e se complementam, portanto, a construção de competências se dá sobre a articulação de metodologias e por meio das diversas formas de ação/mediação.

São elementos desse processo:

- O livro didático.
- O Espaço UnisulVirtual de Aprendizagem (EVA).
- As atividades de avaliação (a distância, presenciais e de autoavaliação).
- O Sistema Tutorial.

## Ementa

Páginas dinâmicas. Conceitos de páginas do lado servidor e cliente. Recursos do Active Server Page (ASP). Comandos básicos SQL. Introdução a PHP. Configuração de servidores Web. Enquetes, fórum, mecanismo de busca. Filtro e totalização de dados. Formulários. Desenvolvimento de aplicações. Implementação de *sites* dinâmicos e seus respectivos testes, avaliação e manutenção.

## Objetivos

### Geral:

O desenvolvimento para Web requer uma ampla gama de conhecimentos em ferramentas necessárias para aperfeiçoar o resultado. Assim, podemos estudar técnicas de desenvolvimento para Web utilizando diversas ferramentas para alcançar o melhor resultado possível.

### Específicos:

- Identificar e caracterizar as aplicações Web sob a visão do desenvolvedor e do usuário.
- Distinguir o uso de linguagens Client-side e Server-side.
- Compreender a dinâmica do uso de banco de dados nas aplicações Web.
- Identificar os recursos das linguagens de marcação para transporte e armazenamento de informações.
- Integrar tecnologias a fim de montar uma solução computacional voltada à Web.

## Carga Horária

A carga horária total da disciplina é 120 horas/aula.

## Conteúdo programático/objetivos

Veja, a seguir, as unidades que compõem o livro didático desta disciplina e os seus respectivos objetivos. Estes se referem aos resultados que você deverá alcançar ao final de uma etapa de estudo. Os objetivos de cada unidade definem o conjunto de

conhecimentos que você deverá possuir para o desenvolvimento de habilidades e competências necessárias à sua formação.

## **Unidades de estudo: 6**

### **Unidade 1 - Aspectos do desenvolvimento de aplicações Web – 20h/a**

A primeira unidade você poderá identificar as utilitárias que existem por traz de um desenvolvimento Web. Esta unidade contém as definições necessárias para identificar as principais características de cada forma de programar, assim como reconhecer as principais vantagens de se desenvolver em plataforma Web.

### **Unidade 2 - Javascript – 20h/a**

O JavaScript hoje se torna essencial para o desenvolvimento Web. Esta linguagem tem como o seu principal uso em validações rápidas e máscaras de campos. A linguagem JavaScript pode ainda ser usada como um recurso de melhoramento visual. Assim você poderá ser capaz de iniciar sua utilização no universo extenso desta linguagem.

### **Unidade 3 - PHP – 20h/a**

Nesta unidade você conhecerá o fantástico mundo do PHP. Conhecerá como utilizá-la de forma a aumentar a sua aplicabilidade. Saberá algumas bibliotecas ou APIs que possibilitam uma programação mais eficiente.

### **Unidade 4 - Sistemas Web com banco de dados (PHP + MYSQL) – 20h/a**

Nesta unidade você aprenderá como trabalhar em conjunto com a linguagem e um banco de dados. Desenvolverá uma aplicação em que utilizará vários níveis do desenvolvimento Web. Perceberá como a escolha correta das linguagens a serem desenvolvidas com

o banco de dados faz diferença na eficiência e na velocidade de desenvolvimento de uma aplicação.

### **Unidade 5 - Extensible Markup Language (XML) – 20h/a**

No XML você será capaz de identificar as características que levaram esta linguagem a se tornar base de diversas tecnologias. Você conhecerá as principais aplicabilidades e os detalhes mais relevantes do Extensible Markup Language (XML).

### **Unidade 6 - Asynchronous Javascript and XML (AJAX) – 20h/a**

Na Unidade 6 você condensará todas as unidades anteriores em uma única solução. Identificará como o uso em conjunto de diversas linguagens podem resultar em um ganho de performance. Você compreenderá o conceito do AJAX, seus benefícios e suas limitações. Assim, identificará quando ela é mais aconselhável em seu uso.



## Agenda de atividades/ Cronograma

- Verifique com atenção o EVA, organize-se para acessar periodicamente a sala da disciplina. O sucesso nos seus estudos depende da priorização do tempo para a leitura, da realização de análises e sínteses do conteúdo e da interação com os seus colegas e tutor.
- Não perca os prazos das atividades. Registre no espaço a seguir as datas com base no cronograma da disciplina disponibilizado no EVA.
- Use o quadro para agendar e programar as atividades relativas ao desenvolvimento da disciplina.

Atividades obrigatórias	
Demais atividades (registro pessoal)	



## Aspectos do desenvolvimento de aplicações Web



### Objetivos de aprendizagem

Ao final do estudo desta unidade, é importante que você:

- Identifique as principais características do desenvolvimento Web.
- Conheça os métodos de comunicação entre as linguagens.
- Conheça as principais tecnologias disponíveis.



### Seções de estudo

**Seção 1** O ambiente

**Seção 2** Conceitos envolvidos no desenvolvimento de aplicações Web

**Seção 3** Características

**Seção 4** Integração de linguagens

**Seção 5** Tecnologias



## Para início de estudo

Ao navegar um pouco na Web, você pode identificar as vantagens do desenvolvimento nesta tecnologia e perceber como as atualizações dos sistemas são rápidas e amplamente difundidas. Devido a essa velocidade, é de se esperar uma redução significativa de custos das versões comparada com as aplicações *desktops*. Além da redução de custo, a Web não exige de um usuário uma determinada configuração de máquina para que possa usá-la.

Uma aplicação Web possui compatibilidade comprovada com diversos sistemas operacionais disponíveis no mercado visto que a aplicação não é executada pelo seu equipamento, mas interpretada por um navegador. Essa facilidade a difere das aplicações *desktops*, pois estas necessitam de versões diferentes para cada sistema operacional.

Você verá como funciona a interpretação das regras de negócio e das validações dos campos e estudará os níveis de comunicação entre os métodos.

Esses conceitos são extremamente importantes, mas vale lembrar que as linguagem *desktop* não são o alvo do presente estudo. Esses conceitos são de grande auxílio para sempre buscar a melhor solução para cada tipo de necessidade dos clientes. Assim você terá um conhecimento mais refinado para poder optar em determinada linguagem e ou que tipo de tecnologia de desenvolvimento (Web ou *desktop*) terá um melhor resultado.

Embora não seja o alvo desta unidade, conhecer as linguagens *desktop* é de grande auxilio para a busca da melhor solução, de acordo com a demanda dos clientes. Conhecer linguagens lhe dará mais subsídios para optar pela tecnologia de desenvolvimento (web ou *desktop*) mais adequada para proporcionar o melhor resultado para a situação em questão.

## Seção 1 – O ambiente

Todo o desenvolvimento Web é realizado no ambiente da internet, inclusive as aplicações que são construídas para serem utilizadas em redes privadas. Todas são regidas sobre as vantagens e desvantagens que a internet proporciona.

A internet, nada mais é que um grande conjunto de redes de computadores interligadas pelo mundo inteiro, basicamente, um conjunto de documentos. Ela foi criada por um grupo de cientistas do CERN (*European Laboratory for Particle Physics*), na Suíça, com o intuito de facilitar a comunicação interna e externa.

Inicialmente, essa rede integrada continha dados os quais cada equipamento deveria interpretá-los individualmente. Ou seja, não existia uma ferramenta que transformasse os dados em informação padronizada. O primeiro a realizar essa tarefa foi o Mosaic (primeiro navegador de grande aceitação no mercado).

O objetivo inicial era centralizar, em uma única ferramenta, as várias tarefas necessárias para se obter as informações disponíveis pela internet de forma integrada e visando à conectividade independentemente do tipo de máquina a ser utilizada. Para manter essa multicompatibilidade, foi utilizado um conjunto de protocolos e serviços em comum, permitindo assim que os usuários a ela conectados pudessem usufruir os serviços de informação de alcance mundial.

O HTML (*Hyper Text Markup Language*) é a linguagem padrão da internet interpretada pelos *browsers*. Ela igualmente foi criada pelo CERN tendo como seu principal responsável Tim Berners-Lee. A motivação de Tim foi encontrar mais praticidade na hora de escrever suas páginas. O HTML proporcionou algumas vantagens, entre elas:

- **Padronização:** as páginas estritas em HTML tem a aparência semelhante nas diversas plataformas de trabalho.

- **Diversidade:** o HTML permite uma singularidade a diversos elementos do documento, ou seja, ele permite que alguns elementos possam ser modificados para torná-los únicos. Por exemplo, padrão da letra, cores etc.
- **Compactação:** as páginas escritas em HTML possuem um tamanho reduzido, a fim de economizar tempo de transmissão por meio da internet.

O HTML é construído para que o documento seja um arquivo texto com marcações que controlam a forma de exibição do documento. Este documento, página ou arquivo pode ser construído utilizando um *framework* elaborado, um *framework* simples ou ainda simplesmente um editor de texto.

Uma das principais vantagens dessa linguagem é que, se um *browser* não entender determinado marcador, ele simplesmente o ignora, evitando qualquer tipo de mensagem de erro (em tempo de execução) e que pouco afeta o resto do documento. Essa nota técnica foi elaborada com base na versão 3.0 do HTML. Logo que a produção desta foi concluída, surgiu o HTML 4.0.

O HTML faz metade do trabalho. Basicamente organiza os dados em um formato padrão. A outra metade é realizada pelos *browsers*, como sucessores do Mosaic. Os *browsers* interpretam esses dados organizados pelo HTML.

Durante a história, já foi utilizado diversos *browsers*. Alguns permaneceram e evoluíram. Outros não conseguiram se adequar às novas tendências do mercado, como foi o caso do Netscape. Esse *browser* foi largamente utilizado na década de 1990, porém não consegui manter sua preferência na década seguinte.

Nesta época, existiam muitas diferenças entre a interpretação do HTML. Tanto que muitas vezes existiam páginas em que o usuário deveria informar qual *browser* estava utilizando. Desta forma, o usuário era encaminhado a seu conjunto de páginas devidamente formatado. Atualmente, esse tipo de problema é mais restrito. Desta forma, a escolha do *browser* acaba muitas vezes por motivos visuais. Vejamos alguns *browsers*.

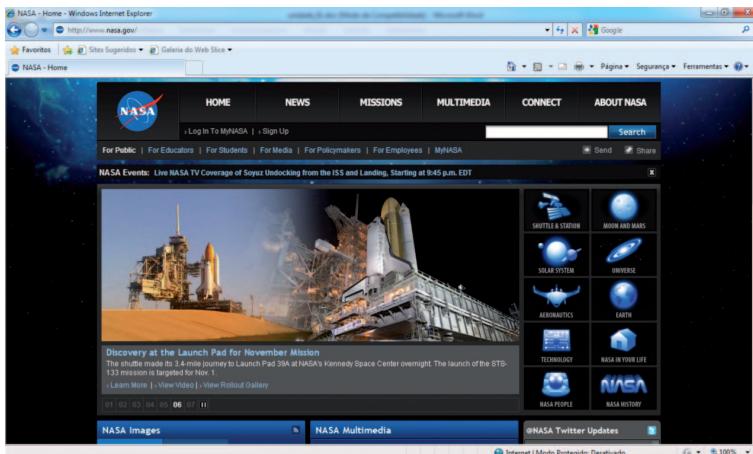


Figura 1.1 - Internet Explorer – desenvolvido pela Microsoft.

Fonte: Elaborada pelo autor (2010).



Figura 1.2 - Chrome – desenvolvido pela Google.

Fonte: Elaborada pelo autor (2010).



Figura 1.3 - Safari – desenvolvido pela Apple.

Fonte: Elaborada pelo autor (2010).



Figura 1.4 - Firefox – desenvolvido pela Mozilla.

Fonte: Elaborada pelo autor (2010).

Você pode perceber que praticamente não há diferença na navegação entre os *browsers*. Porém, eventualmente, é possível que você se depare com diferenças muito mais perceptíveis.

O HTML está presente em todas as páginas da internet, inclusive nas páginas que possuem sons, imagens, vídeos ou animações. Essas interações multimídia são exibidas sobre o HTML.

A sua estrutura se baseia em *tags*. O documento sempre é inicializado pela *tag* <html> e logo após vem o cabeçalho

<head>, o título <title> e o corpo <body>.



Lembre-se que o HTML não é *case-sensitive*, ou seja, ele não distingue <html>, <HTML>, <HtMl>, <HTml>. Todas as *tags* devem ser sempre fechadas, pois desta forma mantém o padrão e não há problemas com outras linguagens que podemos vir a usar. As *tags* de fechamento possuem uma barra "/". Além dos fechamentos, devemos sempre manter a hierarquia do bloco, isto é, as *tags* sejam sempre fechadas do último para o primeiro, deste modo: <head> <title> documento 1 </title> </head>.

Vejamos um exemplo de HTML:

```
<html>
<head>
<title>Primeiro teste</title>
</head>
<body>
Esse é um teste
</body>
</html>
```

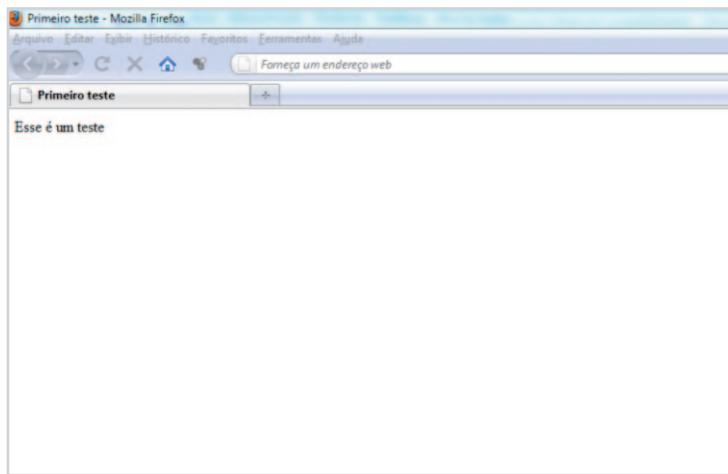


Figura 1.5 - Exemplo de página html.

Fonte: Elaborada pelo autor (2010).

As principais *tags* utilizadas no desenvolvimento Web são as que compõem o formulário HTML (*form*). Eventualmente, para contemplarmos uma solução mais completa, podemos utilizar de outras *tags* para nos auxiliar.

## Seção 2 – Desenvolvimento de aplicações Web

Um dos principais pontos a se destacar quando falamos em desenvolver aplicação para Web é que ela possui algumas vantagens que superam desenvolvimento de aplicação *desktop*. Vamos apresentar alguns conceitos envolvidos no desenvolvimento de aplicações Web.

O desenvolvimento de *software* para a Web mantém um princípio básico: mantenha-o simples; mantenha-o pequeno. Na Conferência de Tecnologia Emergente do ano de 2006 houve vários temas em discussão. Rael Dornfest, ex-chefe executivo da O'Reilly, explanou: “grandes negócios serão criados oferecendo menos a você”. Perceba como essa afirmação se torna importante quando pensamos na **economia da atenção**. Isso quer dizer que não podemos ter mais informações que o tempo que podemos ler.

Uma das vantagens no trabalho do desenvolvedor Web diz respeito às operações de manutenção: ele trabalha com uma aplicação centralizada. Ou seja, a aplicação permanece em um único equipamento onde são acessadas pelos clientes. Desta forma, as **operações de manutenção** da aplicação decorrem de forma muito dinâmica, onde qualquer alteração na aplicação se reflete instantaneamente a todos os usuários conectados. Além disso, as operações de *backup* e integração com outras aplicações são mais bem gerenciadas.

A centralização permite que as estações *desktops* utilizem menos recursos. Essa diminuição favorece na diminuição de custos de um projeto de implementação. Porém, o desenvolvimento Web requer uma metodologia de desenvolvimento mais organizada.

Quando se fala de centralização, não significa que seja uma obrigatoriedade que exista apenas um servidor. As tecnologias atuais permitem conter equipamentos em diversas camadas. Por exemplo, podemos ter um equipamento com apenas o servidor Web; um segundo equipamento com uma linguagem de interpretação como o ASP; e, por último, um equipamento apenas com o banco de dados.

A forma de organizar a infraestrutura depende da estratégia adotada no projeto de desenvolvimento. Muitas variáveis devem

ser levadas em considerações, como: número de acessos, carga de processamento, volume de consultas, entre outras. Apesar de esta análise não ser o foco de estudo desta disciplina, torna-se importante você ter consciência do processo como um todo.

Quanto aos recursos necessários no *desktop* para que uma aplicação Web seja executada, são bem resumidos, uma vez que basta ter acesso a um servidor Web por meio de um *browser* para que a aplicação mais básica passe a funcionar. As aplicações mais robustas em grandes empresas também não perdem muito essa característica. O servidor Web deverá ser mais bem equipado, porém, para alguém passar a utilizar a aplicação, basta um *browser* e uma conexão com o servidor para iniciar o seu trabalho.

Os recursos adicionais que por ventura possam ser necessários, normalmente se tratam de complementos para o *browser* executar uma animação, um vídeo ou outra forma de mídia. Esses tipos de complementos são nomeados como *plug-ins*.

As aplicações Web necessitam de uma conexão com um servidor Web. Esse servidor pode estar instalado no próprio *desktop* do usuário ou em outro equipamento em qualquer distância. Essa **conexão com o servidor** pode ocorrer de duas formas: pela **intranet** ou pela **extranet**. A intranet se caracteriza pela rede **LAN** (rede local) onde o ambiente é muito bem conhecido e controlado. A extranet é a internet propriamente dita, assim a aplicação está aberta a qualquer navegador que tenha acesso a ela.



Lembre-se que, com uma aplicação centralizada, o processamento e o volume de dados trafegam em um único equipamento. Essa preocupação deve estar presente no início do projeto de desenvolvimento do *software*.

**LAN (Local Area Network):**  
sigla para representar  
uma rede local, a rede de  
computadores à qual sua  
máquina está conectada.

Uma grande facilidade para o uso das aplicações Web é que a plataforma já se encontra pronta. Isto é, a Web utiliza de todos os equipamentos de redes já existentes, uma vez que tudo é tratado pelo **protocolo TCP/IP**.

Assim, os serviços de requisições utilizam o seguinte procedimento:

Protocolos TCP/IP é um conjunto de protocolos de comunicação entre computadores em rede.

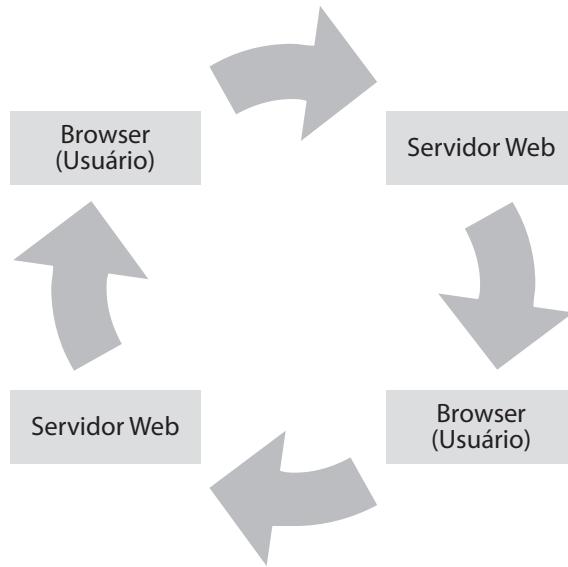


Figura 1.6 - Diagrama de interação *browser x servidor*.

Fonte: Elaborada pelo autor (2010).

Esse diagrama representa de forma básica como há a interação entre *browser* e servidor. Utilizam a rede (internet ou intranet) como plataforma de comunicação.

Uma outra forma de exemplificar a dinâmica das aplicações Web é utilizar um diagrama com o ASP sendo uma linguagem de interpretação de código. Assim como foi utilizado o ASP, poderia ser qualquer outra linguagem, como PHP, JSP, entre outras.

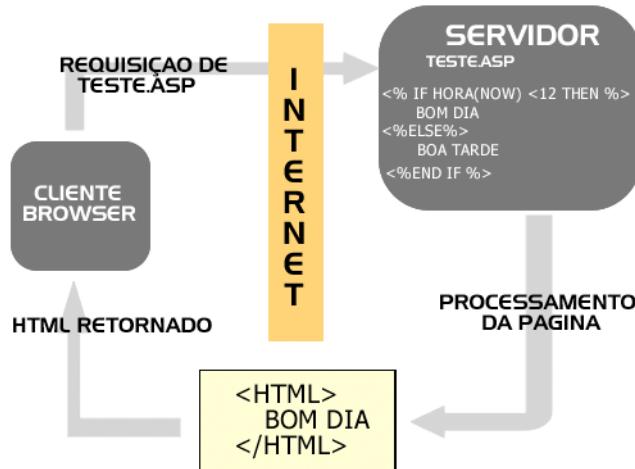


Figura 1.7 - Diagrama em que o ASP é uma linguagem de interpretação de código.

Fonte: <<http://blog.computero.com.br/tutoriais-asp-01-o-que-e-asp/>>. Acesso em 06 jan. 2011.

Perceba que, mesmo utilizando uma linguagem para interpretar uma programação existente, a dinâmica da aplicação Web não foi alterada.

Logicamente esse desenvolvimento não nasceu do dia para a noite. As linguagens vêm evoluindo para que os desenvolvedores possam ter um esforço menor durante seus desenvolvimentos. O esquema a seguir representa a evolução das principais linguagens Web.

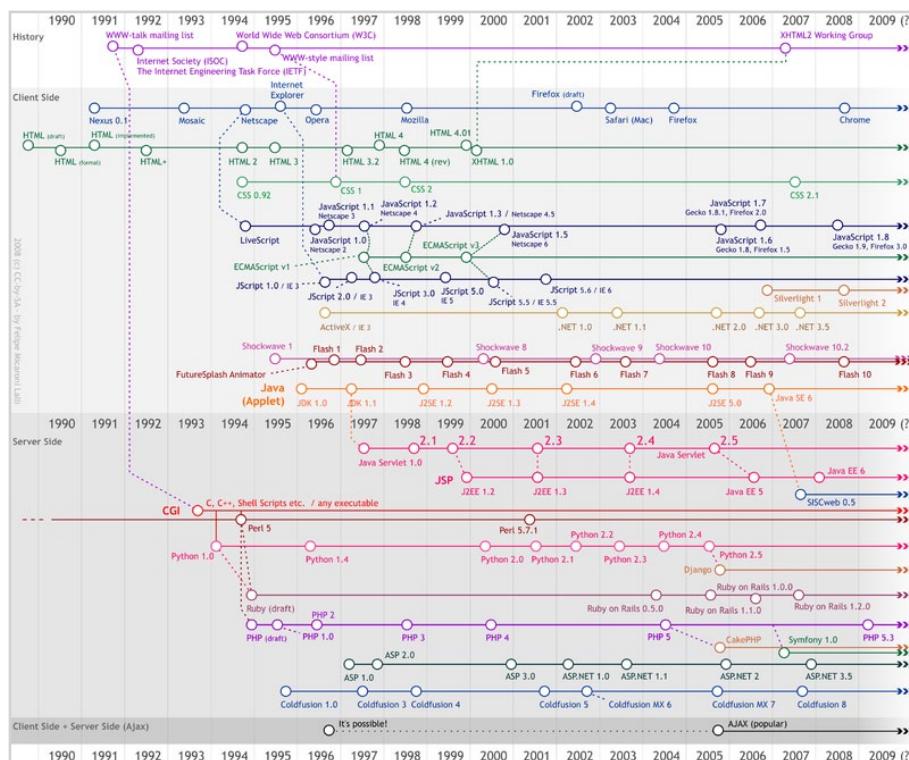


Figura 1.8 - Diagrama de evolução de linguagens de programação.

Fonte: <[www.deccanpro.com/deccanpro\\_web\\_development.php](http://www.deccanpro.com/deccanpro_web_development.php)>. Acesso em 06 jan. 2011.

A possibilidade de uso das linguagens interpretadas no servidor Web tornou-se possível criar páginas em que o resultado depende de um **tipo de requisição**. Ou seja, uma consulta a um banco de dados ou outra forma de armazenamento de dados pode retornar informações diferentes dependendo dos parâmetros para uma mesma página. Assim, as páginas que outrora eram estáticas passam a ser dinâmicas e um universo novo se abre para os desenvolvedores.



A definição da Aplicação Web está entrelaçado com a definição de páginas dinâmicas. Assim, o principal fluxo de informação que trafega nas aplicações Web é o envio (inserção ou manipulação de dados) e o resgate (consulta) de informações de um servidor.

---

Vale ressaltar que o *browser* possui um **tempo interno para aguardar uma resposta** da requisição realizada. Em outras palavras: o *browser* envia alguns dados para o servidor. Este, por sua vez, realiza uma consulta sobre esses dados. Caso essa consulta seja demasiadamente lenta, o *browser* entenderá que a conexão com o servidor não existe mais e retornará uma mensagem de erro. Portanto, qualquer requisição nas aplicações Web deve conter um retorno rápido, de forma a evitar esse problema.

Além do **retorno** das consultas que deve ser levado em consideração, o **envio das informações** também é importante.



Imagine você enviar um arquivo enorme para o servidor. Se um arquivo muito grande (400 Mb, por exemplo) for enviado para o servidor e se a aplicação não estiver preparada para suportar esse tráfego, muito provavelmente toda a aplicação irá parar enquanto processa a sua requisição.

---

Essas considerações dependem do **link com o servidor**. O link, neste caso, refere-se à velocidade de uma conexão com o servidor. Quanto maior a velocidade, maior o volume de dados possíveis de trafegar na rede.

Outro aspecto que é importante ser levado em consideração no projeto de uma aplicação Web é o cuidado com a **usabilidade**. Lembre-se sempre que o custo para diversos *browsers* interpretarem as páginas Web de forma semelhante é a simplicidade do código HTML. Esses elementos gráficos não possuem qualidades semelhantes aos elementos de uma aplicação *desktop*.

O crescente uso da Web como plataforma de sistemas acaba por forçar a utilização de mecanismos que melhorem o visual. Tornando-os mais atrativos para os usuários.

Atualmente existem recursos para minimizar esse problema, como CSS, JavaScript, entre outros. Mas aqui está se referenciando aos elementos básicos do HTML como os elementos de formulário. O uso das **folhas de estilos (CSS)** criam um visual mais amigável e harmônico. O **JavaScript**, além de poder realizar algumas tarefas do CSS, auxilia a interatividade com o usuário como máscaras e validações de campos.

Assim como qualquer aplicação *desktop*, o desenvolvimento de aplicações para Web requer um **projeto** conciso em que possa abordar as diferentes tecnologias que serão utilizadas. Diferente do *desktop*, onde uma linguagem é adotada para o projeto, a aplicação para Web requer, hoje, o uso de várias linguagens e tecnologias para atingir um melhor resultado. Por isso, o uso de **frameworks** auxilia e muito o desenvolvimento em equipes, uma vez que essas ferramentas procuram manter um padrão de desenvolvimento uniforme.



### **Atenção!**

Todo o projeto de *software*, seja de pequeno ou grande porte, requer uma fase de análise rigorosa. Nesta etapa começa-se a identificar possíveis problemas e riscos ao desenvolvimento. Ainda nesta fase, é possível direcionar e identificar as tecnologias utilizadas no desenvolvimento. Execute sempre após o planejamento baseado na análise do projeto.  
**Aprenda o problema, a solução já existe!**

---

Apesar de se tratar de um desenvolvimento de *software*, as aplicações Web possuem algumas características que a diferencia.

- a) ubiquidade;
- b) infraestrutura tecnológica imprevisível;
- c) alta volatilidade dos requisitos de negócio;
- d) equipes multidisciplinares.

### a) Ubiquidade

A ubiquidade nada mais é que a característica de acessar a aplicação de diversos locais para diversos dispositivos. Portanto, para ter essa característica comprovada, a aplicação Web necessita estar disponível à rede. Entenda que, quando é mencionado o acesso à rede, trata-se de rede doméstica, corporativa ou mundial. Hoje, com uso de dispositivos menores para acessar a rede, é possível criar visões específicas para um determinado tipo de dispositivo sem trocar a aplicação.

### b) Infraestrutura tecnológica imprevisível

O que você como desenvolvedor deve sempre manter em mente é que qualquer pessoa conectada à rede pode ter acesso a sua aplicação. Não estamos tratando aqui os fatores de segurança, mas sim o número de acesso que sua aplicação pode ter.

Assim, um usuário que deseja acessar sua aplicação pode conter uma infinidade de configurações diferentes em seu equipamento. Podendo assim variar o sistema operacional, a memória, a velocidade de conexão, o tamanho de tela, a resolução, entre outros.

Todas essas variáveis podem afetar diretamente a forma que o usuário poderá visualizar a sua aplicação. Os *browsers* ainda permitem que os usuários alterem suas configurações livremente desabilitando funcionalidades importantes que sua aplicação pode estar utilizando.

As melhores aplicações Web são aquelas que preveem o maior número de variações de equipamentos possíveis que os usuários possam ter. Um exemplo típico é que os usuários podem desabilitar o uso da linguagem JavaScript. Essa linguagem, como veremos, é largamente utilizada em validações de formulários. Caso sua aplicação apenas valide o formulário via o JavaScript, pode sofrer por usuários mal intencionados que estejam acessando sua aplicação.

Outro exemplo seria a dimensão que você pode estar utilizando. Ao dimensionar sua página Web de tal forma que alguns

usuários tenham de mover a barra de rolagem em diversas direções, leva a uma insatisfação por parte dos usuários. Sem falar que podem perder informações importantes que não estejam sendo mostradas adequadamente.

### c) Alta volatilidade dos requisitos de negócio

Lembre-se que em aplicações Web a regra de negócio fica no servidor Web. Essa característica permite que uma alteração se replique para todos os clientes instantaneamente. Essa ação faz com que seja comum nessas aplicações um índice maior na alteração das regras de negócio se comparadas com aplicações *desktops*.

Aplicações para Web não possuem em sua maioria um controle de *releases*. Contudo, existem algumas aplicações em que esse controle é feito e normalmente são aplicações Web para uso genérico. Nesses casos, você adquire uma cópia, instala e passa a usar. Com a evolução tecnológica ou com a ocorrência de relatos de *bugs*, uma nova *release* é lançada.

### d) Equipes multidisciplinares

Os projetos Web são concebidos, normalmente, por profissionais com conhecimentos abrangentes. Um projeto pequeno desenvolvido por um único profissional acaba por forçar este a aprimorar seus conhecimentos em *design*, banco de dados, linguagem server-side e em linguagem client-side.

Assim, as aplicações para Web formam profissionais com visões abrangentes sobre a solução em que está se pretendendo alcançar.

Você verá adiante mais detalhes sobre a linguagem *server-side* e *client-side*.



#### **Atenção!**

Essa última característica pode se tornar um ponto fraco, pois muitos profissionais acabam por manter o foco em uma generalização tamanha que lhes acaba faltando, algumas vezes, conhecimentos mais aprimorados sobre uma tecnologia.

Já vimos até o momento diversas informações que mostram como o desenvolvimento para a Web é benéfico, mas necessita de alguns cuidados. Chegou a hora de desmitificar uma aplicação Web e um *website*.

A aplicação Web é formada por *websites* que possibilitam aos usuários interagirem com a aplicação. Toda a aplicação para Web é composta por *websites* que propiciam o serviço aos usuários.

A ênfase também é diferente: os *websites* têm em seu ponto forte os aspectos plásticos do sistema (apresentação, movimento, navegação, informação). As aplicações para Web, por sua vez, preocupam-se com os aspectos funcionais da solução (velocidade de resposta, funcionalidade, interatividade, aplicabilidade). Todo esse conjunto é facilitado pelo ambiente que a aplicação está sendo desenvolvida, a internet.

O ambiente da internet possui diversas aplicabilidades e finalidades que podem ser atendidas por uma aplicação. Por causa disso, existem diversas tecnologias que auxiliam a alcançar os objetivos.

Muitas dessas tecnologias são definidas por diversas características como: aparência, interatividade, disponibilidade no servidor Web, experiência do desenvolvedor ou por uma escolha pessoal do desenvolvedor por uma linguagem específica.



### **Atenção!**

Cuidado com a escolha de uma linguagem, pois a escolha errada pode causar fracasso no projeto. Procure nunca utilizar o lado pessoal durante esse processo. Lembre-se que o objetivo de um desenvolvimento é oferecer a melhor solução para um problema.

---

Existem inúmeras linguagens que podem ser trabalhadas no desenvolvimento Web e é possível interpretar linguagens tipicamente *desktops* em servidores Web. Lembre-se que, para

uma linguagem ser interpretada, basta que o servidor Web entenda os comandos provenientes desta.

Vejamos mais detalhadamente duas linguagens amplamente disseminadas: (a) *Client-Sides* e (b) *Server-Sides*.

### a) Linguagens *Client-Sides*

*Client-sides* caracterizam as linguagens que são interpretadas pelo *browser*. Normalmente não fazem parte da regra de negócio composta pelo projeto e possuem uma aplicabilidade de interação imediata.



Linguagens *client-sides* são mais úteis no realce do aspecto visual da aplicação ou quando precisamos de processos simples e rápidos sendo executados na interface, como a validação dos dados digitados pelo usuário.

Hoje, esse tipo de linguagem tem ganhado força com a usabilidade. As aplicações atuais possuem um nível muito grande de personalização e interação. Por esse motivo, as linguagens como CSS, JavaScript e DHTML são cada vez mais trabalhadas e novos profissionais especialistas surgem no mercado.

As linguagens *client-sides* mais comuns são:

- CSS
- HTML
- XHTML
- Javascript
- AJAX
- Flash – Action Script
- Microsoft Silverlight

### b) Linguagens *Server-Sides*

Este tipo de linguagem mantém em seu escopo as regras de negócio da aplicação. Pelo fato de as linguagens *Server-sides* serem executadas no servidor, todo o processamento da aplicação recai sobre ela. Nessa camada se fazem as chamadas a banco de dados e a arquivos de comunicação.



---

As linguagens *server-sides* já proveem mecanismos para a implementação da parte lógica da aplicação, como os algoritmos que executam os processos da camada de negócio ou recursos auxiliares, como acesso a banco de dados.

---

As linguagens mais usadas atualmente são:

- PHP
- ASP
- .NET
- CGI, Perl
- Java, J2EE, WebObjects
- SSJS, Aptana Jaxer, Mozilla Rhino
- Python, Django
- Ruby, Ruby on Rails
- Smalltalk Seaside
- ColdFusion
- Lotus Domino
- Websphere

As linguagens *server-sides* dependem diretamente do servidor Web. Esse servidor necessita estar configurado para interpretar as linguagens. Em outras palavras, para que uma linguagem

se adéque a um desenvolvimento Web e se torne *Server-side*, basta que o servidor Web interprete o seu código. Mesmo linguagens tipicamente *desktop* como o C++ pode ser utilizada no desenvolvimento Web. Apenas o servidor tem de identificar seu código e encaminhar para o compilado C.

*Software* de aplicação (*Software as a Service – SaaS*) são aplicações executadas em um servidor Web ao invés de serem instaladas em um equipamento cliente. Esse tipo de aplicação possui vários tipos de benefícios, como reduzir a demanda dos departamentos internos de TI. O SaaS ainda pode aumentar a acessibilidade para o uso fora dos escritórios facilitando a sua manutenção. O usuário acessa o *software* como um serviço, desta forma sempre está atualizado. Quaisquer atualizações efetuadas no servidor causam diretamente impacto a todos os equipamentos dos usuários. Esse tipo de aplicação vem sendo utilizada pelas empresas para facilitar a atualização dos seus *softwares*, além de adotar uma padronização.

A Google utiliza esse conceito em vários de seus *softwares*. Um dos mais usados é o Google Docs, que se baseia em um conjunto de aplicativos para escritórios, como um editor de textos e uma planilha eletrônica.

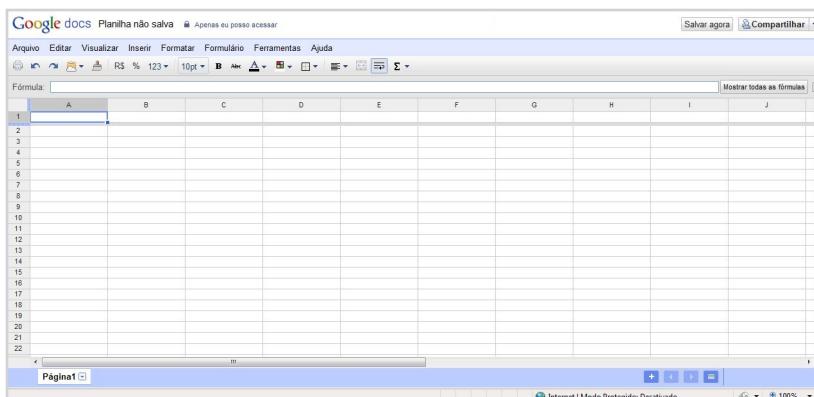


Figura 1.9 - Planilha eletrônica do Google Docs.

Fonte: <<http://docs.google.com>>.

A colaboração de projetos torna-se simplificada. Uma vez que o “arquivo” se encontra no servidor, não é necessário enviar para um colega. Simplesmente o usuário necessita compartilhar o arquivo com os colegas que deseja.

Outra forma de aplicativo que vem ganhando muito espaço e atenção são os **Webtop**. Um Webtop é um *desktop* Web, isto é, um ambiente de sistema operacional disponível pelo *browser*. Esse tipo de aplicação possui um potencial impressionante, já que o usuário poderá trocar de equipamento sem perder as suas configurações e os seus arquivos.

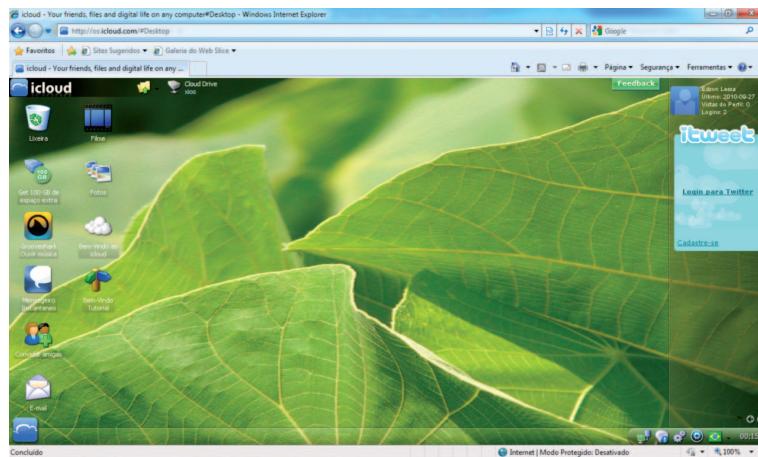


Figura 1.10 - Webtop, uma forma de trabalhar de forma virtual.

Fonte: <http://icloud.com/pt>. Acesso em 06 jan. 2011

Como você pôde notar, o desenvolvimento Web possui inúmeras aplicabilidades e está longe de encontrar o seu limite. Porém, somente desenvolver não é tudo. Conhecer o usuário e procurar técnicas para melhorar o uso das aplicações são fatores fundamentais. Essa tentativa de facilitar o uso das aplicações e potencializar o seu uso chama-se de usabilidade.

### Seção 3 – Usabilidade

Já foi mencionado aqui sobre usabilidade, como ela vem ganhando espaço e cada vez mais é utilizada em aplicações Web. Mas, afinal, o que é a usabilidade?

Usabilidade é uma característica daquilo que é utilizável, funcional. É tornar óbvio o óbvio, tendo em conta as necessidades do utilizador e o contexto em que este está inserido. Podemos relacionar usabilidade com os seguintes verbos: simplificar, rentabilizar, otimizar, facilitar, melhorar, acelerar. Mais usabilidade é sinônimo de maior flexibilidade e de maior interação com os usuários.

A usabilidade está em todo o lugar, praticamente em todos os produtos que o ser humano desenvolveu é possível alterar para melhor. Porém, nem sempre mudar significa que irá melhorar.

Qualquer pessoa que já tenha operado uma tevê poderá utilizar qualquer outra. Essa afirmação é verdadeira, porém nem sempre poderá utilizar no máximo de seus recursos. Não são todos os detalhes que o usuário terá conhecimento, como uma função *sleep*, *mute*, canal anterior, menu.

Agora, se alterarmos o produto como celular, forno, forno micro-ondas, lavadoras de louça, computadores, videogame. Percebemos que todos caem cabem nesse exemplo.

Desta forma, a forma que se usa um determinado produto deve ser adequado para que se torne intuitivo. Assim, quando o produto não o faz, o problema do projeto é do *design*, da usabilidade.



Vejamos um caso de sucesso: a Apple se esforça para que seus produtos se tornem completamente intuitivo e que todo o “ritual” de utilizar um produto da Apple torne-se único e de uma facilidade sem precedentes.

---

Botões e informações nos devidos locais com sua hierarquia de importância; cores e diagramação intuitivas e adequadas; comportamento adequado ao público. Isso é usabilidade em ação. Quando um produto é usável, nem se pensa no conceito de usabilidade. O problema está quando o produto não se adequa ao utilizador. A usabilidade é um caminhar progressivo em direção ao utilizador, e não o contrário.



Na internet a usabilidade não basta, mas corresponde a uma boa parte do caminho. Se considerarmos o exemplo de uma loja *online*, veremos que é algo simples: se um usuário não encontra um produto, não o compra. Fazer com que o utilizador encontre o que procura é usabilidade em ação.

---

Um *site* tem, em uma perspectiva otimista, entre 10 e 15 segundos para convencer, por isso todos os elementos da página

precisam ser avaliados, pesados, medidos em termos de impacto. Um dos fatores mais apontados para páginas Web é a velocidade de carregamento. Um usuário que espera dez segundos para uma página carregar, tende a desistir da visita. Todos os elementos gráficos necessitam ser altamente otimizados (tamanho, qualidade, interesse). Sempre é de boa prática, verificar se todas as imagens são necessárias. Questionar se todos os elementos gráficos são fundamentais. Tratando de aplicações corporativas, esse questionamento deve ser realizado com muito mais seriedade. Aplicações corporativas não necessitam vender um produto, mas maximizar a funcionalidade.

Um *layout* deve ser consistente e implementado com o foco da aplicação. O *layout* deve mostrar a filosofia, a ideia central em que a aplicação desempenha dentro da empresa. Cada empresa tem uma linguagem, uma imagem que o *layout* do site necessita sublinhar.



O layout deverá ser concebido a partir do ponto onde se pretende que o usuário inicie a movimentação visual. A distribuição dos elementos e as cores escolhidas são importantíssimas, normalmente os olhos movimentam-se da esquerda para a direita e de cima para baixo.

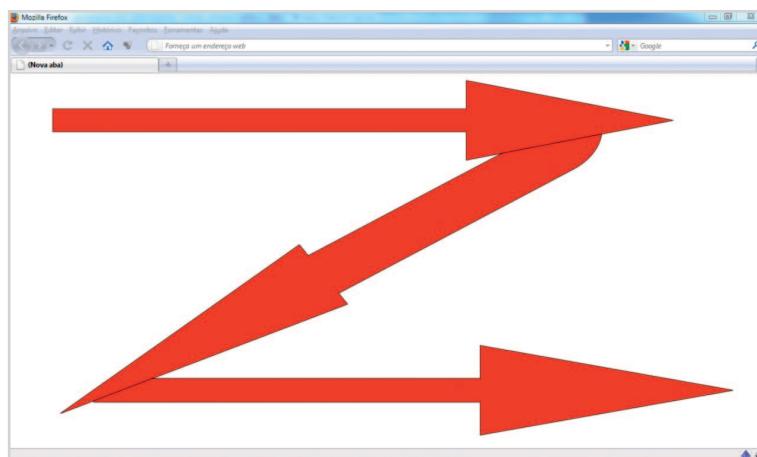


Figura 1.11 - Simulação do movimento dos olhos ao ler uma página Web.  
Fonte: Elaborada pelo autor (2010).

Esse formato indica que o *layout* ideal é aquele que leva o usuário sempre a observar primeiro os elementos superiores

esquerdos, normalmente utilizados em muitos *sites* e *softwares* como uma área reservada para logotipo, após este ponto descendo em diagonal. Os olhos procuram normalmente primeiro os elementos maiores e só depois os menores. A cor influencia, a tendência é primeiramente observamos os elementos mais escuros e só depois os mais claros.

Quando falamos de *layout*, sempre devemos levar em consideração a **coerência** e o **equilíbrio** do visual apresentado. Não devemos imprimir um *layout* onde os elementos gráficos sejam maiores que as informações textuais. No caso de formulários, devemos evitar cadastros longos. Normalmente os formulários devem estar totalmente visíveis na janela do *browser*.

Existe uma técnica de avaliar o equilíbrio da tela, devemos traçar uma linha imaginária vertical ou horizontalmente em metade da tela. As medidas devem conter aproximadamente a mesma quantidade de texto.

## Seção 4 – HTML

**HTML** é a linguagem padrão de todos os *browsers*. Portanto, é fundamental que você conheça e entenda muito bem esta linguagem.

Os comandos HTML são representados no texto por meio de *tags*. O *browser* interpreta como *tags* todos os elementos HTML válidos que estejam entre os sinais “menor que” (<) e “maior que” (>). Sua sintaxe básica é:

```
<tag>Descrição</tag>
```

Algumas *tags* podem ter um ou mais atributos, que definem alguma característica especial. As *tags* permitem que você crie atributos novos. O *browser* irá ignorar os atributos conhecidos. Esse artifício permite que você adicione características únicas

para cada elemento do HTML. As *tags* com atributos são formatadas da seguinte forma:

```
<p align="center">teste</p>
<table border="1">
<tr>
<td>Célula</td></tr>
</table>
```

Veja que, no exemplo, o parágrafo está sendo centralizado. Já a tabela possui uma borda aparente.

Todo o documento HTML possui uma estrutura de hierarquia. Essa estrutura inicia com a identificação do documento por meio da *tag* <HTML>. Dentro do documento, a estrutura separa em cabeçalho e corpo.

```
<html>
<head>
<title>Teste</title>
</head>
<body>
<p align="center">teste</p>
<table border="1">
<tr>
<td>Célula</td></tr>
</table>
</body>
</html>
```

A *tag* <head> delimita a parte do documento sendo o cabeçalho. O cabeçalho é onde se encontram as informações do documento. Nenhuma informação contida dentro desse bloco deve conter informações a serem escritas no *browser*. Além de informações de configuração da página, a *tag* <title> informa o título do documento, que será apresentado na barra superior do *browser*. Essa *tag* deve sempre estar dentro do cabeçalho.

A *tag* <body> contém o corpo do documento. Essa *tag* contém em seu interior toda a informação que será mostrada em sua página. Os textos, as imagens, os formulários, tudo que será

apresentado pelo *browser*. Os atributos dessa *tag* definem cores para os textos, *links*, imagem de fundo, entre outras características.

O HTML ainda oferece uma série de *tag* que são utilizadas para formatar as informações nele contidas. Podemos formatar textos de títulos utilizando as *tags* `<h1>` até `<h6>`. A numeração representa os níveis dos títulos.

```
<html>
  <head>
    <title>Teste de títulos</title>
  </head>
  <body>
    <h1>Título 1</h1>
    <h2>Título 2</h2>
    <h3>Título 3</h3>
    <h4>Título 4</h4>
    <h5>Título 5</h5>
    <h6>Título 6</h6>
  </body>
</html>
```

O *browser* interpretará da seguinte forma:

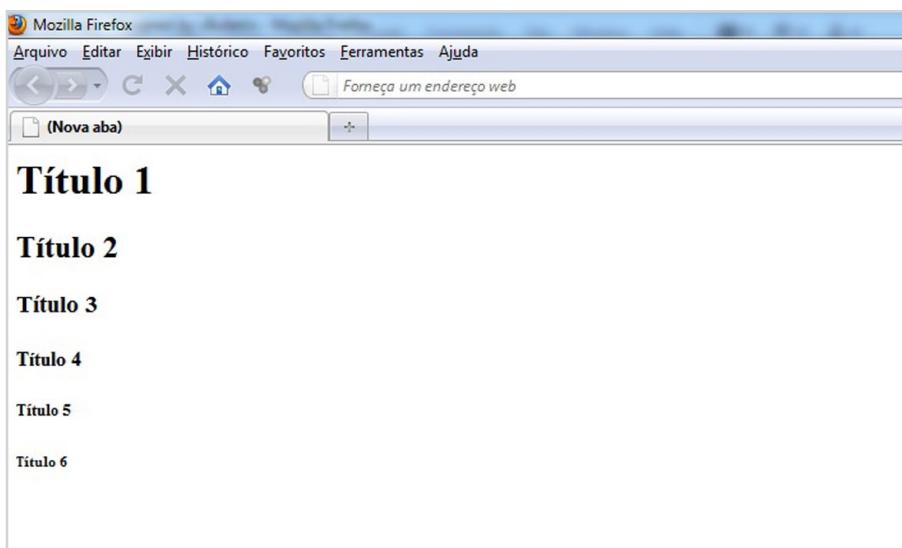


Figura 1.12 - *Tags* de formatação.

Fonte: Elaborada pelo autor (2010).

Além das *tags* de títulos, as quebras de linhas e parágrafos são muito utilizadas em textos e formatação de formulários. O HTML usa comandos especiais para definir a quebra de um parágrafo ou de uma linha.

A tag `<p>` indica a quebra de parágrafos, ela insere automaticamente uma linha em branco. O uso da tag de parágrafo pode ser feito em uma única linha ou em bloco, esse uso é opcional. O comando `<p>` permite alinhar o parágrafo dentro do documento.

Como fica no *browser*:

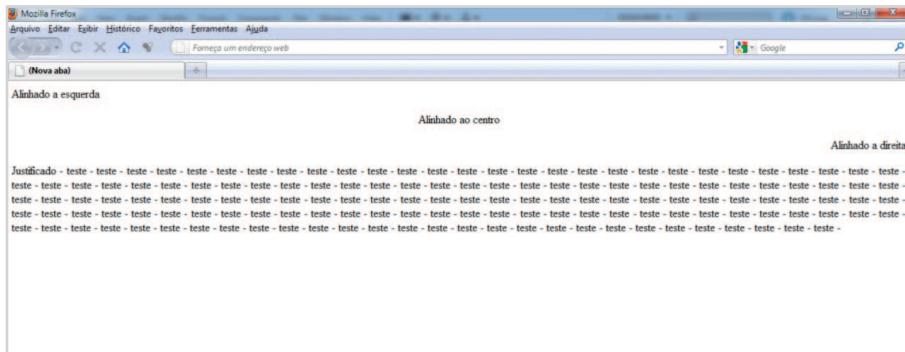


Figura 1.13 - Tags de parágrafo e alinhamento de texto.

Fonte: Elaborada pelo autor (2010).

Em vez de inserirmos uma quebra de parágrafo, pode ser necessária uma quebra de linha somente. Neste caso, devemos utilizar a tag `<br>`, pois ela finaliza a linha atual e passa para a próxima. Esse elemento não é utilizado em bloco. Portanto, o browser quando a encontra entende que o texto posterior a ela é colocado em uma nova linha.

```
<html>
<head>
<title>Teste de quebra de linha</title>
</head>
<body>
Quebra de linha<br>Nova linha
</body>
</html>
```

O browser interpreta da seguinte forma:

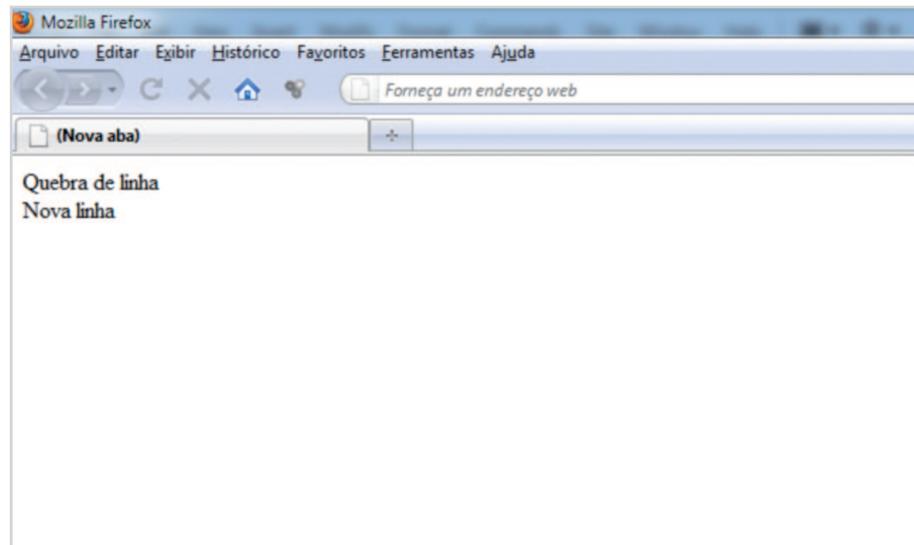


Figura 1.14 - Tag de nova linha.

Fonte: Elaborada pelo autor (2010).

A **tag <PRE>** informa ao *browser* que deve interpretar o texto conforme foi digitado.

```
<html>
<head>
<title>Teste de pré-formatação</title>
</head>
<body>
<pre>Texto pré-formatado.
O browser interpretará exatamente desta forma que está sendo digitado.
Inclusive com espaços      em branco</pre>
</body>
</html>
```

O *browser* interpreta desta forma:

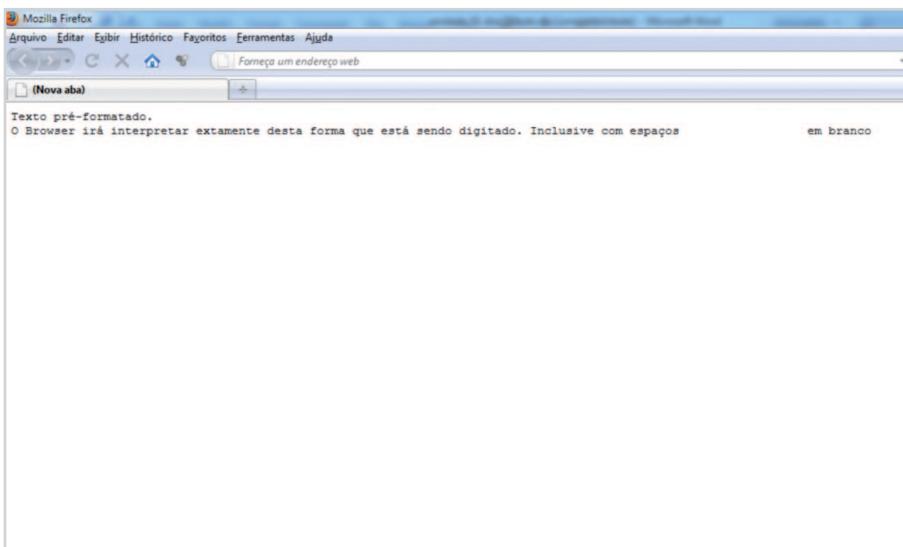


Figura 1.15 - Tag de pré-formatação.  
Fonte: Elaborada pelo autor (2010).

Por ser uma linguagem de formatação, o HTML também permite alterar o estilo do texto. Como aplicar negrito, itálico, sublinhado, subscrito, entre outros.

```
<html>
<head>
<title>Teste de formatação de texto</title>
</head>
<body>
<b>Texto em negrito</b>
<br>
<i>Texto em itálico</i>
<br>
<u>Texto sublinhado</u>
<br>
<sup>Texto sobreescrito</sup>
<br>
<sub>Texto subscrito</sub>
<br>
<big>Almenta a fonte</big>
<br>
<small>Reduz a fonte</small>
</body>
</html>
```

No *browser* fica assim:

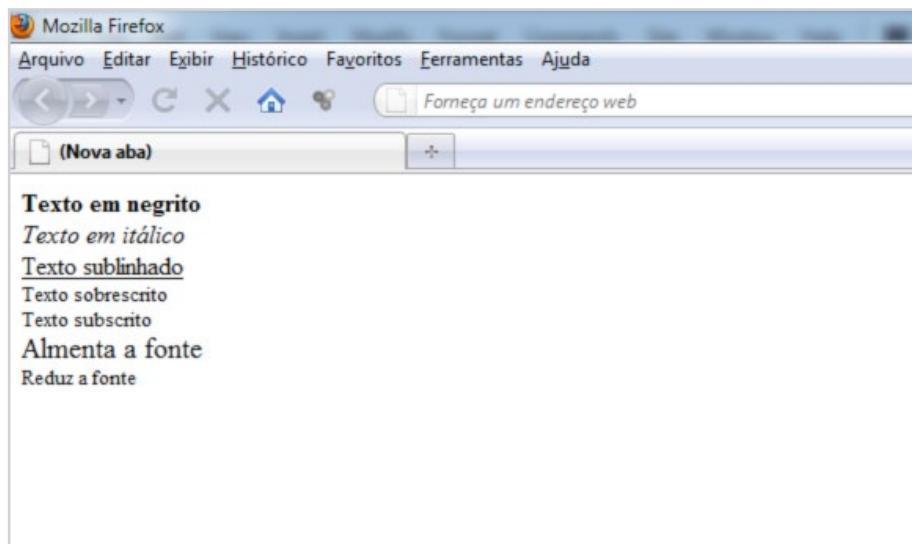


Figura 1.16 - *Tags* de alteração de estilo.

Fonte: Elaborada pelo autor (2010).

O HTML ainda possibilita a configuração da fonte de texto. O elemento utilizado é a *tag* <font>. Além do elemento, ainda é possível manipular algumas informações por meio dos atributos, como tamanho e cor da fonte.

```
<html>
<head>
<title>Teste de fonte de texto</title>
</head>
<body>
<font face="Verdana, Geneva, sans-serif"> Teste fonte Verdana</
font><br />
<font face="Arial, Helvetica, sans-serif" size="+1" color="#000066">Teste
fonte arial</font><br />
<font face="Times New Roman, Times, serif" size="+2"
color="#006633">Teste fonte Times New Roman</font><br />
<font face="Tahoma, Geneva, sans-serif" size="+3"
color="#009900">Teste fonte Tahoma</font><br />
<font face="Lucida Sans Unicode, Lucida Grande, sans-serif" size="+2"
color="#0000FF">Teste fonte Lucida</font><br />
<font face="Courier New, Courier, monospace" size="+4"
color="#FF0000">Teste fonte Courier</font><br />
</body>
</html>
```

O *browser* interpreta da seguinte forma:

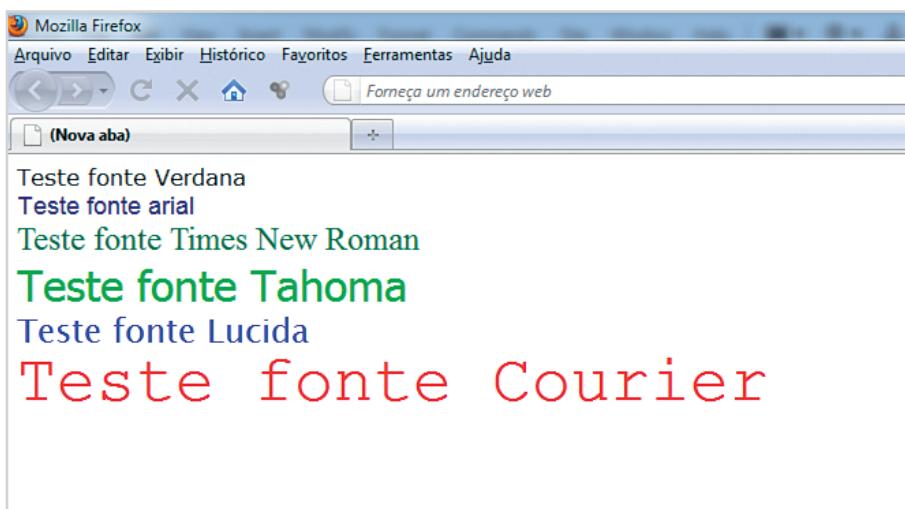


Figura 1.17 - Tags manipuladores de fontes.

Fonte: Elaborada pelo autor (2010).

Uma das *tags* extremamente utilizadas em uma tentativa de tornar os *layouts* mais amigáveis e atrativos para os usuários. Esta *tag* é a `<div>`, ela é tratada também como camada visual em DHTML. Já no HTML básico é utilizado como bloco de alinhamento horizontal.

```

<html>
<head>
<title>Teste de Div</title>
</head>
<body>
<div align="center">Usando a TAG DIV:</div>
<BR>
<div align="right">
<P>Usando a TAG DIV você pode alinhar seu texto sem problemas.
<P>Pode utilizar o alinhamento à direita, que alinha seu texto na
margem
direita da tela. Como um texto criado em editor de textos para ser
impresso em uma página.
<P>Com a TAG DIV você pode aproveitar o mesmo alinhamento para
vários
parágrafos.
</div>
</body>
</html>
  
```

O *browser* interpreta da seguinte forma:

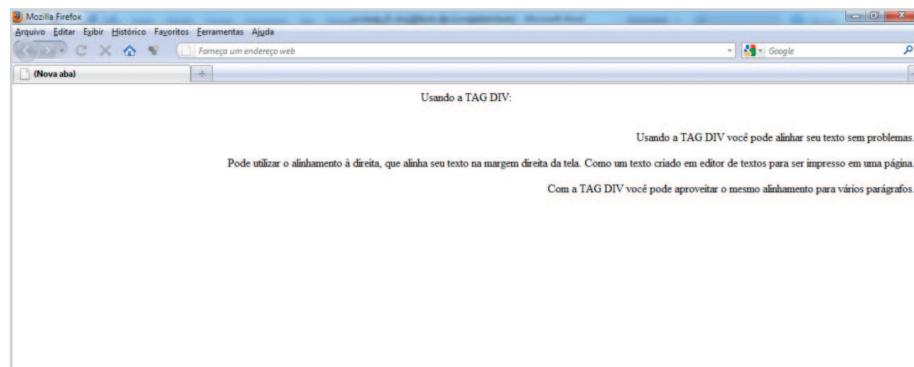


Figura 1.18 - Tag de camadas.

Fonte: Elaborada pelo autor (2010).

Uma forma de organizar as informações dispostas no *browser* é a utilização de listas. As listas podem servir como resumos ou índices de todo o *site*, contendo *links* para outras páginas. Podemos dividir as listas em três tipos:

- listas não ordenadas;
- listas ordenadas ou numeradas;
- listas descritivas.

As listas não ordenadas representam a formatação simples de informações por meio de indicadores. A *tag* do comando é a `<ul>` e seus tópicos são a `<li>`. Podemos ainda alterar o tipo do marcador pelo atributo *type* da *tag* `<ul>`.

```
<html>
<head>
<title>Teste de Lista não Ordenada</title>
</head>
<body>
<ul type="circle">
<li>Tópico 2</li>
<li>Tópico 3</li>
<li>Tópico 4</li>
</ul>
<ul type="disc">
<li>Tópico 2</li>
<li>Tópico 3</li>
<li>Tópico 4</li>
</ul>
<ul type="square">
<li>Tópico 2</li>
<li>Tópico 3</li>
<li>Tópico 4</li>
</ul>
</body>
</html>
```

O *browser* interpretará da seguinte forma:

```
<html>
<head>
<title>Teste de Lista Ordenada</title>
</head>
<body>
<ol type="1">
<li>Tópico 2</li>
<li>Tópico 3</li>
<li>Tópico 4</li>
</ol>
<ol type="a" start="4">
<li>Tópico 2</li>
<li>Tópico 3</li>
<li>Tópico 4</li>
</ol>
```

```
<ol type="A">
<li>Tópico 2</li><li>Tópico 3</li>
<li>Tópico 4</li>
</ol>
<ol type="i" start="2">
<li>Tópico 2</li>
<li>Tópico 3</li>
<li>Tópico 4</li>
</ol>
<ol type="l">
<li>Tópico 2</li>
<li>Tópico 3</li>
<li>Tópico 4</li>
</ol>
</body>
</html>
```

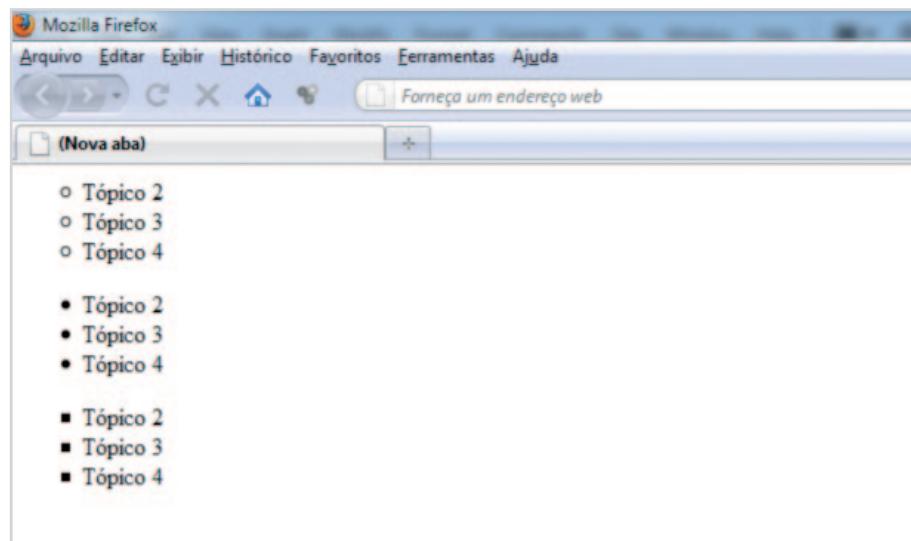


Figura 1.19 - Tags de listas não ordenadas.

Fonte: Elaborada pelo autor (2010).

As listas ordenadas não possibilitam enumerar os tópicos por meio de números ou em ordem alfabética. A tag utilizada é a `<ol>`, além do atributo *type* que define o tipo de numerador, a tag possui um segundo atributo, o *start*. Esse atributo informa ao *browser* em qual número a lista deve iniciar.

A apresentação no *browser* será:

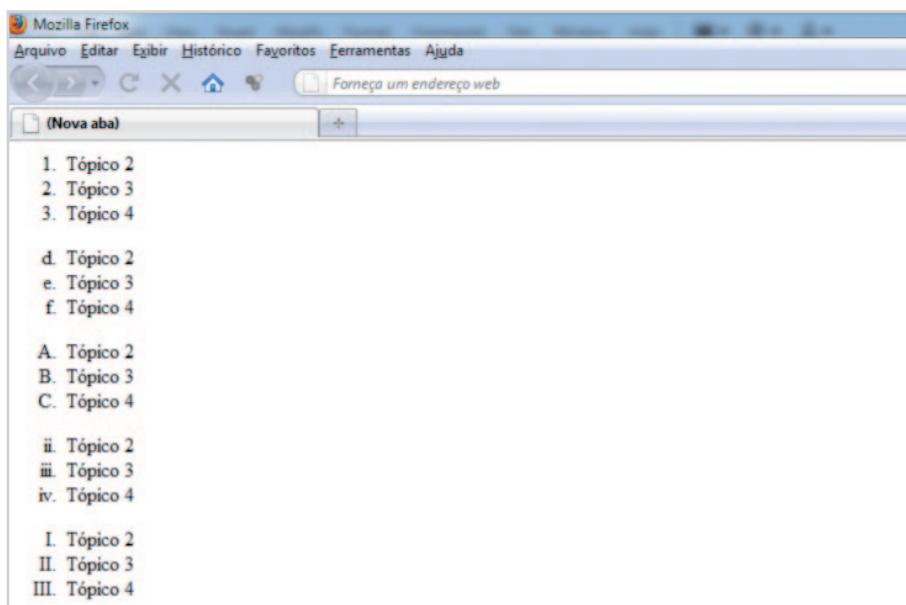


Figura 1.20 - *Tags* lista ordenada.

Fonte: Elaborada pelo autor (2010).

As listas descritivas são diferentes das outras listas. Essa lista inicia pela *tag* <dl>. Cada item de uma lista descritiva possui dois componentes: um tópico <dt> e uma descrição do tópico <dd>.

```
<html>
<head>
<title>Teste de Lista Descritivas</title>
</head>
<body>
<dl>
<dt>Tópico 1</dt>
<dd>Descrição do tópico 1</dd>
<dt>Tópico 2</dt>
<dd>Descrição do tópico 2</dd>
<dt>Tópico 3</dt>
<dd>Descrição do tópico 3</dd>
</dl>
</body>
</html>
```

O *browser* deve apresentar da seguinte forma:

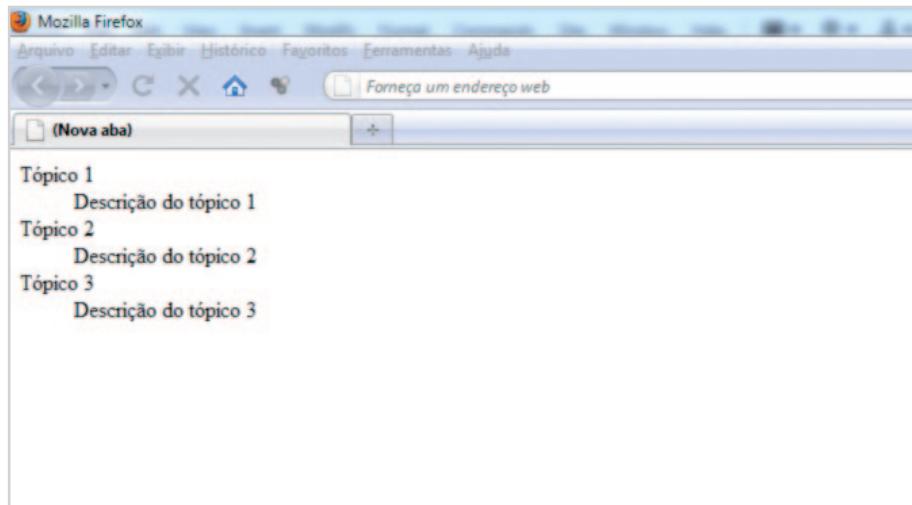


Figura 1.21 - Tags de lista descritivas.

Fonte: Elaborada pelo autor (2010).

Outra funcionalidade que é amplamente usada em praticamente todas as páginas da internet é o *link* ou hipertexto. Essa funcionalidade representa a ligação primitiva entre as páginas formando a teia da internet.

Os *links* proporcionam a ligação entre páginas e textos da própria página. Essa ligação interna é utilizada principalmente em textos longos com tópicos específicos.

Os *links* para outras páginas podem ser classificados como locais e externos. Os *links* locais representam uma ligação com uma página que está salva no mesmo local da página principal. Isso significa que a página que está sendo acessada fará uma referência direta a outro arquivo, sem a necessidade de utilizar o protocolo HTTP. O código para esse tipo de *link* se torna extremamente simples:

```
<a href="segundaPagina.htm">Acessar a segunda página</a>
```

Já quando tratamos de *links* externos, o *link* necessita realizar a chamada pelo protocolo HTTP.

```
<a href="http://www.google.com.br">Acessar o Google</a>
<a href="../outros/paginatres.html">Acessar página três em outros
diretório</a>
```

Porém, quando o acesso se tornar de âncora para uma parte do texto da própria página, a sintaxe do HTML se altera um pouco:

```
<html>
<head>
<title>Link de Texto (Ancora)</title>
</head>

<body>
<a href="#primeiraparte">Tabela</a>
<br />
<a href="#Atalho">Atalhos</a>
<pre>
<a name="primeiraparte" />
Tabelas

Recurso
Criando tabelas com um teclado
Vantagem
Você pode capturar informações tabulares de forma rápida e fácil
Situações
Reuniões, apresentações, debates pessoais

Tente agora

1. Coloque o cursor ao final da palavra “Produto” abaixo:

    Produto

2. Pressione TAB e digite “Quantidade”.
3. Pressione TAB novamente e digite “preço”.
4. Pressione ENTER.
5. Pressione TAB para passar de uma célula a outra e digite alguma
informação nas células.
6. Pressione ENTER ao final da linha.
7. Tente pressionar TAB abaixo, após “João” e após “Maria”:

    Pessoa  Dinheiro que lhes devo
    João
    Maria
<a name="Atalho" />
```

Atalhos úteis:

- ALT + ENTER — insere outra linha dentro de uma célula (quando ENTER não funciona na última célula de uma tabela).
- CTRL + ENTER — insere uma linha abaixo.
- CTRL + ALT + E, R — insere uma coluna à esquerda ou à direita.

Além disso, tente fazer experiências com DEL, BACKSPACE, HOME, END e ENTER em vários pontos da tabela. Verifique os Atalhos do Teclado no Office Online para descobrir outras funções de tabela que podem ser executadas em um teclado.

```
</pre>
</body>
</html>
```

Além dos hipertextos, as tabelas são elementos amplamente utilizados na internet. Ela não faz nenhuma ligação com outras páginas, mas organiza de uma forma matricial qualquer dado da página, tanto textos como outros elementos. No caso das aplicações, as tabelas são utilizadas para organizar o formulário na página.

Quando você domina o uso das tabelas, acaba tendo uma enorme facilidade em criar *layouts* organizados. As informações são dispostas de forma mais clara para o usuário. Quando usada com as folhas de estilos CSS, tornam-se ainda mais amigáveis. Porém, o próprio CSS é responsável pelo declínio no uso das tabelas para melhoramento dos *layouts* de páginas.

Uma tabela pode ser criada de forma simples, apenas para organizar algum elemento ou mesmo o texto que você queira utilizar. O padrão das tabelas é não conter bordas. As tabelas são dispostas por linhas e células, apesar de no *browser* dar a falsa impressão que possui colunas.

```
<html>
<head>
<title>Exemplo de Tabela</title>
</head>
<table>
<tr>
<td>Linha 1 C&acute;lula 1</td>
<td>Linha 1 C&acute;lula 2</td>
<td>Linha 1 C&acute;lula 3</td>
<td>Linha 1 C&acute;lula 4</td>
</tr>
<tr>
<td>Linha 2 C&acute;lula 1</td>
<td>Linha 2 C&acute;lula 2</td>
<td>Linha 2 C&acute;lula 3</td>
<td>Linha 2 C&acute;lula 4</td>
</tr>
<tr>
<td>Linha 3 C&acute;lula 1</td>
<td>Linha 3 C&acute;lula 2</td>
<td>Linha 3 C&acute;lula 3</td>
<td>Linha 3 C&acute;lula 4</td>
</tr>
</table>
<body>
</body>
</html>
```

O *browser* interpreta desta forma:

The screenshot shows a Microsoft Word window titled "Exemplo de Tabela". Inside the table, there are three rows of text. The first row contains four cells with the text "Linha 1 Célula 1 Linha 1 Célula 2 Linha 1 Célula 3 Linha 1 Célula 4". The second row contains four cells with the text "Linha 2 Célula 1 Linha 2 Célula 2 Linha 2 Célula 3 Linha 2 Célula 4". The third row contains four cells with the text "Linha 3 Célula 1 Linha 3 Célula 2 Linha 3 Célula 3 Linha 3 Célula 4". The table has a light blue header row.

Figura 1.22 - Tag de tabelas.

Fonte: Elaborada pelo autor (2010).

Os atributos das tabelas podem alterar a aparência, como a disposição do conteúdo. As bordas, o alinhamento do conteúdo, o espaçamento entre as células, os espaços entre os dados e a próxima célula, entre outros, podem ser alterados.

Alguns dos mais usados são:

- *width* - atributo que manipula a largura da tabela em relação ao *browser*, pode ser tratado com *pixels* ou percentual relativo.
- *height* - atributo responsável pela altura da tabela em relação ao *browser*, pode ser tratado com *pixels* ou percentual relativo.
- *border* - atributo que trata da espessura da borda em *pixels*.
- *cellspacing* - atributo responsável pelo espaço entre uma célula e outra tratado em *pixel*.
- *cellpadding* - atributo responsável pelo espaço entre os dados e a borda da tabela tratado em *pixels*.



Procure pesquisar sobre os atributos das tabelas.  
Eles são de grande auxílio no desenvolvimento e na apresentação.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Atributos de células</title>
</head>

<body>
<table width="400" border="1">
<tr>
<td>Linha 1 C&eacute;lula 1</td>
<td valign="bottom">Linha 1 C&eacute;lula 2</td>
<td bgcolor="#FF0000">Linha 1 C&eacute;lula 3</td>
<td>Linha 1 C&eacute;lula 4</td>
</tr>
<tr>
<td valign="bottom">Linha 2 C&eacute;lula 1</td>
<td valign="middle" nowrap>Linha 2 C&eacute;lula 2</td>
<td>Linha 2 C&eacute;lula 3</td>
<td align="center">Linha 2 C&eacute;lula 4</td>
</tr>
<tr>
<td width="20" height="20">Linha 3 C&eacute;lula 1</td>
<td valign="top">Linha 3 C&eacute;lula 2</td>
<td align="right">Linha 3 C&eacute;lula 3</td>
<td>Linha 3 C&eacute;lula 4</td>
</tr>
</table>
</body>
</html>
```

```
<html>
<head>
<title>Tabela com atributos</title>
</head>
<body>
<table border="1" cellpadding="2" cellspacing="1" width="70%" height="50%">
    <tr>
        <td>Linha 1 C&acute;lula 1</td>
        <td>Linha 1 C&acute;lula 2</td>
        <td>Linha 1 C&acute;lula 3</td>
        <td>Linha 1 C&acute;lula 4</td>
    </tr>
    <tr>
        <td>Linha 2 C&acute;lula 1</td>
        <td>Linha 2 C&acute;lula 2</td>
        <td>Linha 2 C&acute;lula 3</td>
        <td>Linha 2 C&acute;lula 4</td>
    </tr>
    <tr>
        <td>Linha 3 C&acute;lula 1</td>
        <td>Linha 3 C&acute;lula 2</td>
        <td>Linha 3 C&acute;lula 3</td>
        <td>Linha 3 C&acute;lula 4</td>
    </tr>
</table>
</body>
</html>
```

Já no *browser* fica:

Linha 1 Célula 1	Linha 1 Célula 2	Linha 1 Célula 3	Linha 1 Célula 4
Linha 2 Célula 1	Linha 2 Célula 2	Linha 2 Célula 3	Linha 2 Célula 4
Linha 3 Célula 1	Linha 3 Célula 2	Linha 3 Célula 3	Linha 3 Célula 4

Figura 1.23 - Propriedade de tabela.

Fonte: Elaborada pelo autor (2010).

As células, por sua vez, possuem atributos que são igualmente fundamentais para uma boa utilização das tabelas. Você pode manipular o alinhamento vertical e horizontal, a cor da célula, se o texto poderá ou não ter quebra automática de linha, entre outras.

O *browser* interpreta da seguinte forma:

Linha 1 Célula 1	Linha 1 Célula 2	Linha 1 Célula 3	Linha 1 Célula 4
Linha 2 Célula 1	Linha 2 Célula 2	Linha 2 Célula 3	Linha 2 Célula 4
Linha 3 Célula 1	Linha 3 Célula 2	Linha 3 Célula 3	Linha 3 Célula 4

Figura 1.24 - Propriedades de células.

Fonte: Elaborada pelo autor (2010).

Como você pode notar, a estrutura da tabela sempre tenderá a um ajuste automático. Ou seja, todas as células da mesma ordem são forçadas a assumir a característica que define o tamanho de uma célula.

As linhas não fogem à regra, você pode utilizar alguns atributos como cor, alinhamento, entre outros. Lembre-se sempre que os atributos da linha modificarão todas as células contidas nela.



Os atributos das células e das linhas podem auxiliá-lo de forma surpreendente, procure pesquisar mais sobre quais são e como utilizá-los.

As células e as linhas ainda podem ser mescladas. Esse artifício é muito utilizado para facilitar a organização das informações. Para mesclar as células de uma mesma linha, utilize o atributo *colspan* e o número de células que você está mesclando. Já para mesclar células de linhas diferentes, utilize o atributo *rowspan* e o número de célula que você está mesclando.

```
<html>
<head>
<title>Tabela Mesclando Células</title>
</head>
<body>
<table border="1">
<tr>
<td>Linha 1 C&acute;lula 1</td>
<td colspan="3">Linha 1 C&acute;lula Mesclada 2 - 3 - 4</td>
</tr>
<tr>
<td rowspan="2">C&acute;lula 1- Linhas 2 e 3 mescladas</td>
<td>Linha 2 C&acute;lula 2</td>
<td>Linha 2 C&acute;lula 3</td>
<td>Linha 2 C&acute;lula 4</td>
</tr>
<tr>
<td>Linha 3 C&acute;lula 2</td>
<td>Linha 3 C&acute;lula 3</td>
<td>Linha 3 C&acute;lula 4</td>
</tr>
</table>
</body>
</html>
```

A apresentação no *browser* fica:

Linha 1 Célula 1	Linha 1 Célula 2	Linha 1 Célula 3	Linha 1 Célula 4
Linha 2 Célula 1	Linha 2 Célula 2	Linha 2 Célula 3	Linha 2 Célula 4
Linha 3 Célula 1	Linha 3 Célula 2	Linha 3 Célula 3	Linha 3 Célula 4

Figura 1.25 – Mesclando células.

Fonte: Elaborada pelo autor (2010).

Os formulários são utilizados em todas as aplicações Web com base em HTML. O formulário possibilita ao usuário entrar com informações destinadas ao servidor. Estes, por sua vez, processam e devolvem uma resposta. Os formulários não processam as informações. Essa responsabilidade é efetuada por um interpretador no servidor, como o caso do ASP ou PHP. O interpretador ASP lê os dados imputados do formulário e realizar as instruções contidas no programa. Após o término, remete para o usuário o carregamento completo.

Uma linguagem *Server-side*, como o ASP, podem ser utilizadas de diversas formas. Podemos relacionar algumas:

- processar as requisições e os dados enviados pelo usuário por meio de formulários;
- tornar-se uma interface entre HTML e banco de dados SQL, fazendo a conversão da transação HTML para SQL e vice-versa;
- converter dados do sistema em HTML gerando respostas para o cliente.

Os *scripts* ou programas são escritos em linguagens compatíveis com a plataforma sob a qual o servidor está rodando e devem produzir arquivos executáveis.

Os elementos de um formulário são definidos separadamente. Na definição do formulário, você informa o local do programa (CGI, ASP, PHP) que controlará o formulário e a forma ou o método como os dados serão enviados para o servidor.

A *tag* utilizada para o formulário é o *form*. Essa *tag* possui atributos que definem o destino do formulário, o método de envio, o nome, entre outros. Sobre esses atributos, podemos relacionar:

- *method*: define o método utilizado pelo servidor para receber os dados do formulário. Este atributo pode receber dois valores:
  - *post*: método que transmite toda a informação do formulário incorporada no documento do HTML;
  - *get*: anexa o conteúdo do formulário ao endereço da URL, possui limitação de tamanho das informações;
- *action*: especifica o programa ou a página do servidor que processará os dados do formulário;
- *name*: define o nome do formulário.

Os elementos de formulário são campos destinados a receber dados do usuário. Os mais utilizados são: *input*, *select*, *textarea*.

#### a) Elemento *input*

O elemento *input* é de longe o mais utilizado. Sua configuração gráfica varia conforme os valores do atributo *type*. Visualmente pode variar de uma caixa de texto para um botão.

Os principais atributos deste elemento são:

- *type* - define o tipo de entrada que o elemento será:
  - *text* - campo simples de entrada de texto, utiliza como principais atributos associados: *type*, *name*, *value*, *size*, *maxlength*.
  - *password* - texto protegido, não é possível visualizar os caracteres digitados, utiliza como principais atributos associados: *type*, *name*, *value*, *size*, *maxlength*.
  - *hidden* - utiliza como principais atributos associados: *type*, *name*, *value*.
  - *radio* - elemento com um botão de seleção única, utiliza como principais atributos associados: *type*, *name*, *value*, *checked*.
  - *checkbox* - caixa de seleção onde pode ser checado diversos elementos, utiliza como principais atributos associados: *type*, *name*, *value*, *checked*.
  - *reset* - botão responsável em limpar os dados do formulário, utiliza como principais atributos associados: *type*, *name*, *value*.
  - *submit* - botão responsável em enviar o formulário para o servidor, utiliza como principais atributos associados: *type*, *name*, *value*.
  - *image* - mesma função no atributo *submit*, porém o botão é associado a uma imagem, utiliza como principais atributos associados: *type*, *name*, *value*, *src*.
  - *button* - cria um botão sem ação predefinida, utiliza como principais atributos associados: *type*, *name*, *value*.
  - *file* - abre uma janela para selecionar arquivo, utiliza como principais atributos associados: *type*, *name*, *value*.
- *name* - nome do campo ou da variável.
- *src* - local no servidor onde a imagem devia ser carregada.
- *value* - valor *default* do campo.

- *checked* - indica que o campo deve estar marcado ou não como padrão. Quando não possuir valor atribuído assume *true*.
- *size* - define o tamanho do campo, ou seja, a quantidade de caracteres que será exibida em campos do tipo *text* e *password*.
- *maxlength* - define a quantidade de caracteres que o campo pode receber em campos do tipo *text* e *password*.

```
<html>
<head>
<title>Teste formul&aacute;rio</title>
</head>

<body>
<form action="processa.asp" method="POST">
<table>
<tr>
<th align="left" Valign="top">Caixa de texto comum:</th>
<td><input type="text" name="txtTexto" size="20"
maxlength="25"></td>
</tr>
<tr>
<th>Texto protegido por senha:</th>
<td><input type="password" name="pwsSenha" size="6"
maxlength="6"></td>
</tr>
<tr>
<th>Bot&otilde;es de Radio:</th>
<td> Qual o seu time?
<input type="radio" name="radOpcao" value="F"
checked="true">Flamengo
<input type="radio" name="radOpcao" value="V">Vasco</td>
</tr>
<tr>
<th>Checkboxes:</th>
<td> Que softwares voc&ecirc; cohece?
<input type="checkbox" name="word" value="Sim"> Word
<input type="checkbox" name="excel" value="Sim"> Excel
```

```

<input type="checkbox" name="firefox" value="Sim">Firefox</td>
</tr>
<tr>
<th>Botão de Envio:</th>
<td><input type="submit" name="cmdEnvio" value="Enviar"></td>
</tr>
<tr>
<th>Botão de Limpeza:</th>
<td><input type="reset" name="cmdLimpar" value="Limpar"></td>
</tr>
<tr>
<th>Tipo File:</th>
<td><input type="file" name="filArquivo"></td>
</tr>
</table>
</form>
</body>
</html>

```

O *browser* fica:

Figura 1.26 - Formulários, entrada de dados dos usuários.

Fonte: Elaborada pelo autor (2010).

### b) Elemento select

O elemento *select* trata-se de uma lista de itens que podem ser selecionados pelo usuário. Esta seleção pode se dar de forma única ou múltipla, dependendo dos atributos que o elemento possa ter.

- *name* - atributo nome do elemento.
- *size* - atributo representa a quantidade de linhas que a lista mostrará.
- *multiple* - define o tipo de seleção da lista. Caso este atributo esteja presente, a lista terá seleção múltipla.
- *option* - esta *tag* representa a lista propriamente dita. Define cada opção da lista, as opções devem ser definidas entre as *tags* <option></option>.
- *selected* - indica que a opção estará selecionada como *default*. Este atributo é usado na *tag option*.

```
<html>
<head>
<title>Teste formulário select</title>
</head>
<body>
<form method="get" name="formulario" action="processa.asp">
<table>
<tr>
<th>Quais as linguagens você já viu?</th>
<td><select name="linguagens" size="5" multiple>
<option>CSS</option>
<option selected>HTML</option>
<option>XHTML</option>
<option>Javascript</option>
<option>Flash – Action Script</option>
<option>Microsoft Silverlight</option>
<option>PHP</option>
<option>ASP</option>
<option>.NET</option>
<option>CGI, Perl</option>
<option>Java, J2EE, WebObjects</option>
<option>Python, Django</option>
<option>Ruby, Ruby on Rails</option>
<option>Smalltalk Seaside</option>
<option>ColdFusion</option>
```

```
</select>
</tr>
<tr>
    <th>Qual seu estado civil</th>
    <td><select name="estadoCivil">
        <option>Casado</option>
        <option selected>Solteiro</option>
        <option>Divorciado</option>
        <option>Viúvo</option>
    </select></td>
</tr>
</table>
</form>
</body>
</html>
```

O *browser* aparecerá da seguinte forma:

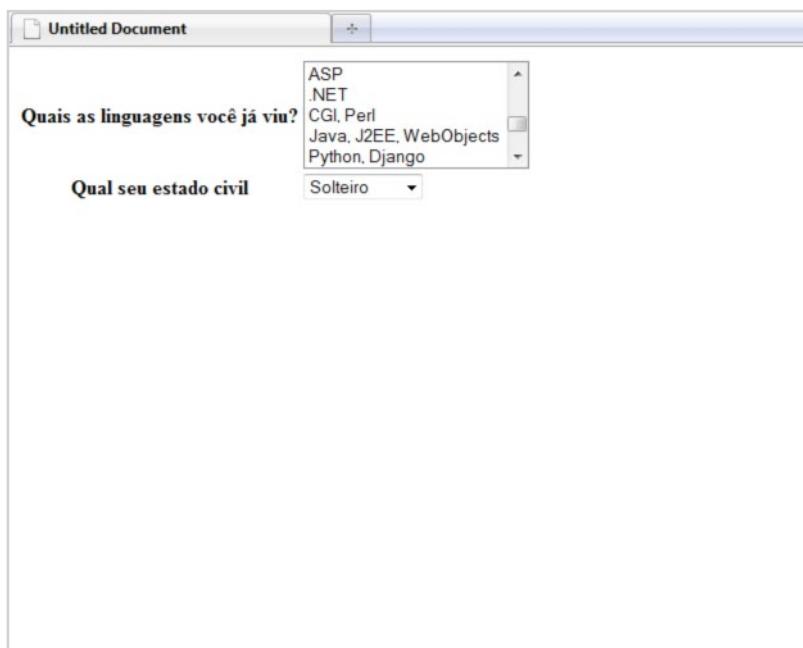


Figura 1.27 – Elemento *select* (*combobox*).

Fonte: Elaborada pelo autor (2010).

### a) Elemento ***textarea***

Trata-se de uma caixa de texto para serem digitadas várias linhas de dados. Este tipo de elemento é utilizado quando o usuário necessita digitar um grande volume de informação. Normalmente com várias linhas de texto. Os principais atributos são:

- *name* - atributo nome do elemento.
- *rows* - número de linhas da caixa de texto.
- *cols* - número de colunas da caixa de texto.
- *texto* - define o texto que aparecerá como *default*.

```
<html>
<head>
<title>Formul&aacute;rio TextArea</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
</head>
<body>
<table>
<tr>
<th>Digite seu coment&aacute;rio</th>
<td><textarea name="comentario" rows="10" cols="30">Pode digitar
aqui!</textarea></td>
</tr>
<tr>
<td><input type="submit" name="enviar" value="Enviar"></td>
<td><input type="reset" name="limpar" value="Limpar"></td>
</tr>
</table>
</body>
</html>
```

O *browser* interpretará da seguinte forma:

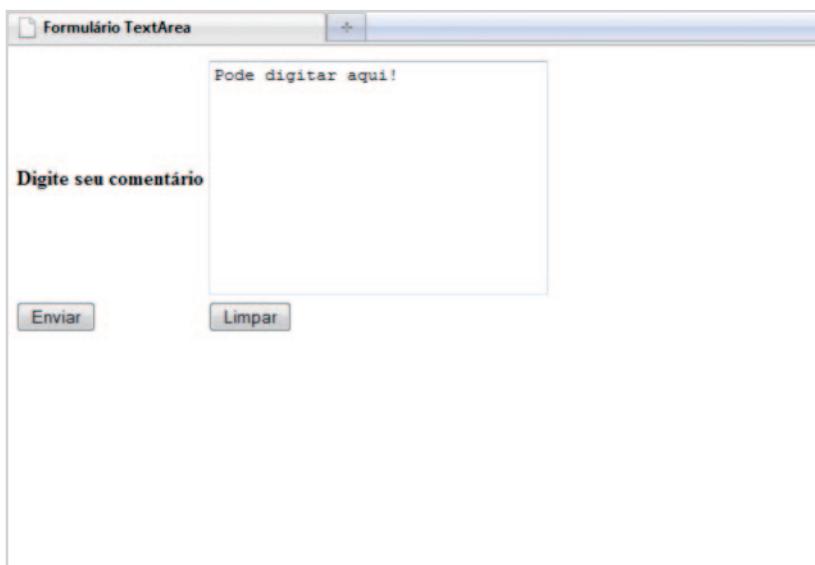


Figura 1.28 - Elemento de área de texto.

Fonte: Elaborada pelo autor (2010).

Você pode relembrar alguns dos elementos do HTML. Estes elementos são fundamentais para um bom entendimento de como o desenvolvimento da aplicação é realizada e como pode ser controlada.



O HTML exposto nesta seção descreveu diversos elementos nos quais você utilizará durante o curso. Pesquise e aprofunde seus conhecimentos buscando mais elementos e atributos que possam auxiliá-lo no desenvolvimento.

## Seção 4 – Integração de linguagens

As aplicações Web possuem um alto nível de integração entre as linguagens, uma vez que a plataforma em que é desenvolvida não proporciona todas as facilidades e ferramentas para que possamos construir uma solução amigável.



Quando falamos de integração, entenda que nos referimos à troca de informações entre uma linguagem e outra por meio de tecnologias disponíveis.

Vamos tentar entender desta forma. Via de regra, linguagens não conversam entre si. Para superar esse obstáculo, é possível utilizar alguns artifícios para que possamos alcançar os objetivos.

Como já vimos, o HTML é a linguagem utilizada para que o *browser* interprete os dados de uma forma padronizada e formatada. Vimos ainda que as linguagens *server-side* são responsáveis por processar as informações provenientes do HTML. Então, como podemos resgatar as informações de um servidor e escrever de forma dinâmica?

Basicamente existem duas formas de enviar informações para o servidor via HTML. São os métodos **GET** e **POST**.

### a) Método GET

O **método GET** é usado quando, no endereço da página Web, informações são encapsuladas para serem lidas pelo servidor. Este encapsulamento pode ser feito de duas formas: **escrita da url** e **por formulários**.

Quando você utiliza o método GET no formulário HTML, toda a informação contida no bloco *form* é enviada pela url. Para utilizar, basta colocar na tag *form* o atributo *method="GET"* e automaticamente será processado desta forma.

Usando de uma forma prática, teríamos o seguinte formulário HTML:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<title>Método Get</title>
</head>
<body>
<form action="Dadosget.asp" method="get" name="form1">
<table>
<tr>
<th>Marca :</th>
<td><input name="marca" type="text" id="marca" value="" /></td>
</tr>
<tr>
<th> Modelo:</th>
<td><input name="modelo" type="text" id="modelo" value="" /></td>
</tr>
<tr>
<th> Combustível :</th>
<td><input name="combustivel" type="text" id="combustivel" value="" /></td>
</tr>
<tr>
<th> Cor :</th>
<td><input name="cor" type="text" id="cor" value="" /></td>
</tr>
<tr>
<th> Ano : </th>
<td><input name="ano" type="text" id="ano" value="" size="4" /></td>
</tr>
<tr>
<td><input name="enviar" type="submit" id="enviar" value="enviar" /></td>
<td><input name="limpar" type="reset" id="limpar" value="limpar" /></td>
</tr>
</table>
</form>
</body>
</html>
```

Repare que está sendo enviado para um arquivo de processamento ASP chamado *Dadosget.asp*. Neste arquivo é onde se trabalha a diferença de requisição do formulário. Portanto, teríamos o seguinte código:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<title>Processamento dados GET</title>
</head>
<body>
<h2> Dados do Automóvel: </h2>
<% Response.Write "Marca : " &Request.querystring("marca") %><br>
<% Response.Write "Modelo : " &Request.querystring("modelo")
%><br>
<% Response.Write "Combustível: " &Request.querystring("combustivel")
%><br>
<% Response.Write "Cor : " &Request.querystring("cor") %><br>
<% Response.Write "Ano : " &Request.querystring("ano") %><br>
</body>
</html>
```

O comando *Response.Write* apenas escreve na página. Já o comando *Request.QueryString* recupera a informação contida nos campos do formulário. Desta forma, temos um *Request.QueryString* para cada campo do formulário que será recuperado.

Ainda para ilustrar, você pode perceber que a url é montada com os valores dos campos contidos no formulário.

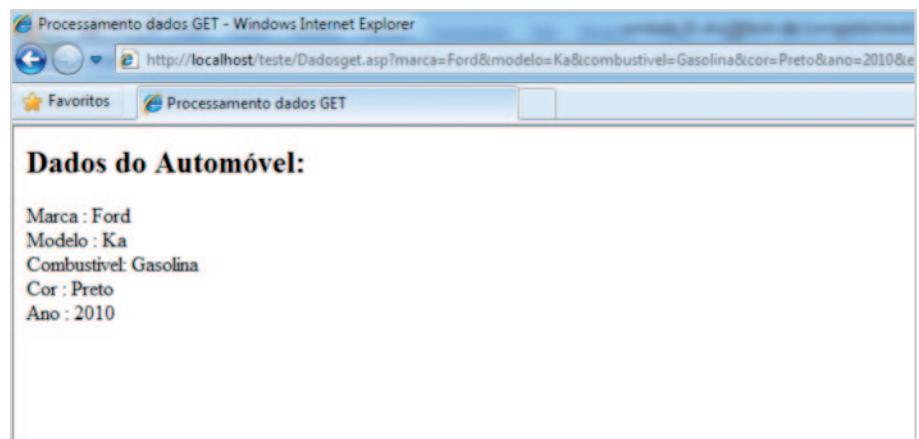


Figura 1.29 - Recuperação de dados por meio do método GET.

Fonte: Elaborada pelo autor (2010).

Outra forma de utilizar este método é escrever manualmente na url quais são os parâmetros que deseja. Este método normalmente é utilizado nos *links* das páginas.



Quando você digita o endereço <<http://www.universia.com.br/ead>>, entrará no *site* da EaD da Universia. Porém, se você quer acessar uma notícia específica, deve acessar: <<http://www.universia.com.br/ead/materia.jsp?id=19065>>. Onde “materia.jsp” é a página onde possui o designer e a parte “id=19065” está transmitindo para o sistema que tem de recuperar a notícia 19065, por exemplo.



#### **Atenção!**

Por usar a url como meio de comunicação, existe um limite de caracteres a serem enviados. A url aceita 255 caracteres no total. Quando você for usar o método GET, você tem de levar em consideração o endereço da página e as variáveis que você está transmitindo.

### b) Método POST

O segundo método de transmissão de informações para o servidor do HTML é o **POST**. Este método realiza o encapsulamento dos elementos do formulário diretamente no corpo da página. Isso significa que toda a informação contida no formulário é enviada como se fizesse parte do HTML estático da página.

---

**Atenção:** este método não pode ser escrito via url. A forma de se acessar ele é utilizando o elemento form com o atributo method="POST".

---

Utilizando o mesmo formulário HTML do exemplo anterior, é possível demonstrar verificar que não se deve utilizar a mesma forma para ler um formulário submetido via método *post*.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<title>Processamento dados Post</title>
</head>
<body>
<h2> Dados do Automóvel: </h2>
<% Response.Write "Marca : " &Request.Form("marca") %><br>
<% Response.Write "Modelo : " &Request.Form("modelo") %><br>
<% Response.Write "Combustível: " &Request.Form("combustivel") %><br>
<% Response.Write "Cor : " &Request.Form("cor") %><br>
<% Response.Write "Ano : " &Request.Form("ano") %><br>
</body>
</html>
```

Veja que agora se altera o comando para recuperação da informação, mas a estrutura é a mesma. Perceba que agora a url não possui parâmetros algum. Toda a informação está contida no corpo do HTML.

Conter dois comandos distintos para recuperar as informações do formulário é um padrão entre as linguagens.

Agora que enviamos os dados para o servidor podemos processar o retorno da informação. Normalmente esse retorno está associado a uma busca no banco de dados e a aplicação das regras de negócio sobre essa informação. Quando se envia o resultado, podemos utilizar a forma mais básica para interagir as linguagens *Server-side* com as *Client-side*: as linguagens *Server-side* escrevem as linguagens *Client-side*.

Como as linguagens *Server-side* são executadas no servidor, elas são processadas antes que a informação chegue ao *browser*. Assim, têm-se as respostas prontas para serem formatas conforme a necessidade e possibilitando a adequação das linguagens *Client-side* como o HTML e Javascript.

**Atenção!**

Todo o browser possui um tempo de espera da resposta do servidor. Quando um processamento no servidor for muito demorado, o *browser* interpretará que a conexão com o servidor não existe mais e não receberá resposta. Assim sua aplicação irá gerar um erro pelo simples fato de estar com um processamento muito elevado.

---

A tecnologia está sempre em constantes alterações e inovações. Procure pesquisar e manter-se atualizado sobre as novas tendências que estão surgindo.



## Síntese

Durante esta unidade você viu como é possível identificar uma aplicação Web e como ela trabalha conceitualmente. Você estudou sobre a importância do HTML e como utilizar formulários e seus atributos.

Percebeu que, apesar de o HTML ser uma linguagem simples, possui diversas opções que apresentam variações significativas.

Para finalizar, você viu ainda que existem diversas linguagens disponíveis para o seu aprendizado, mas todas compartilham dos mesmos conceitos aqui vistos.



## **Atividades de autoavaliação**

Ao final de cada unidade, você será provocado a realizar atividades de autoavaliação. O gabarito está disponível no final do livro-didático. Mas se esforce para resolver as atividades sem ajuda do gabarito, pois assim você estará estimulando a sua aprendizagem.

- 1) Você viu diversas utilidades para as aplicações Web. Cite e comente duas vantagens e duas desvantagens de uma aplicação Webtop.

---

---

---

---

---

---

---

---

---

---

- 2) As aplicações Web possuem diversas linguagens que podem ser classificadas como Server-side e client-side. Com base nesta afirmação, enumere a segunda coluna de acordo com a primeira.

- |                                  |                           |
|----------------------------------|---------------------------|
| 1 – Linguagem <i>Server-side</i> | ( ) PostgreSQL            |
| 2 – Linguagem <i>Client-side</i> | ( ) HTML                  |
|                                  | ( ) Oracle                |
|                                  | ( ) SQLite                |
|                                  | ( ) Javascript            |
|                                  | ( ) Microsoft Silverlight |
|                                  | ( ) PHP                   |
|                                  | ( ) MySQL                 |
|                                  | ( ) ASP                   |
|                                  | ( ) DB2                   |
|                                  | ( ) .NET                  |
|                                  | ( ) CGI                   |
|                                  | ( ) Perl                  |
|                                  | ( ) Java                  |

- CSS
- Python
- Microsoft SQL Server
- Ruby
- Firebird
- ColdFusion
- Websphere
- Flash

3) Você viu a importância do HTML e como ele é necessário para o desenvolvimento Web. Viu também como existem diversos atributos que manipulam aparência e em alguns casos os dados que existem neles. Qual o limite de atributos que podemos utilizar em cada *tag*?

---

---

---



## Saiba mais

Se você desejar, aprofunde os conteúdos estudados nesta unidade ao consultar as seguintes referências:

W3Schools. **W3Schools Online Web Tutorials**, 2010.  
Disponível em: <[www.w3schools.com/](http://www.w3schools.com/)>. Acesso em: 01 mar. 2010.

W3C. **World Wide Web Consortium**, 2010. Disponível em: <[www.w3.org/](http://www.w3.org/)>. Acesso em: 2 mar. 2010.



## UNIDADE 2

# JavaScript

2



### Objetivos de aprendizagem

Ao final do estudo desta unidade, é importante que você:

- Identifique os elementos do Javascript.
- Compreenda a sintaxe da linguagem.
- Conheça os principais comandos do Javascript.



### Seções de estudo

**Seção 1**   JavaScript aplicado

**Seção 2**   Tipos de dados

**Seção 3**   Elementos de linguagem

**Seção 4**   Scripts JS



## Para início de estudo

A linguagem JavaScript possui a característica de ser *client-side*, ou seja, o *browser* se encarrega de interpretar seu código. Por causa disso, existem ligeiras diferenças nesta interpretação. A linguagem é a mesma para qualquer *browser*, porém as ações dos *browsers* divergem um pouco uma das outras. Essas divergências proporcionam chamadas de eventos distintas, além de interpretações de ações diferenciadas como a captura das ações de teclas.

O JavaScript é uma linguagem simples, porém extremamente poderosa. Foi concebida inicialmente pela Netscape e teve forte colaboração da Sun Microsystems, empresa responsável pela linguagem Java. Existe uma discussão sobre o JavaScript ser ou não orientada a objetos. A prática mostra que ele possui acesso e comportamento de uma linguagem orientada a objetos. Mas esses recursos são limitados e sua programação é feita de forma estruturada acessando, quando necessários, os recursos de orientação a objetos.

Essa linguagem é amplamente conhecida e divulgada como JavaScript, já a versão produzida pela Microsoft foi concebido o nome de JScript. Porém, tratam-se de implementações que sendo fiéis à norma ECMAScript lhe acrescentaram novas funcionalidades úteis, mas respeitando sempre as especificações oficiais.

Vale lembrar que JavaScript não é Java e que a grande semelhança consiste apenas nome.

## Seção 1 – JavaScript aplicado

O JavaScript trabalha com os elementos do HTML de forma a conceituá-los como objetos a serem acessados. Assim todos os elementos existentes em uma página Web são transformados

em objetos pelo JavaScript. Internamente são criados objetos que permitem aos usuários criarem novos objetos dependendo da necessidade. Durante o carregamento da página, o JavaScript cria os objetos em uso com suas respectivas propriedades e valores que são ajustados pelo conteúdo da própria página.

Qualquer objeto JavaScript possui uma hierarquia que deve ser respeitada. Cada propriedade pode ser acessada descrevendo toda a hierarquia que se segue.

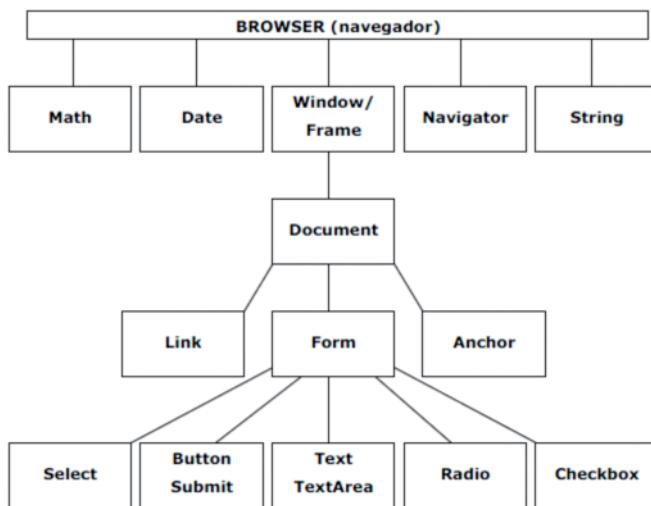


Figura 2.1 - Hierarquia dos objetos do JavaScript.

Fonte: JavaScript – aplicações interativas para a Web (2006).

A figura 1 representa a **hierarquia de objetos do JavaScript**. Veja que vários objetos são filhos de outros, e para acessar o filho temos de escrever todo o caminho a ser percorrido até ele. Assim, se desejamos acessar a propriedade *form*, devemos primeiramente acessar o objeto *document* separando-os por ponto:

```
document.form
```

Esta forma hierárquica de chamadas dos objetos HTML são sempre utilizadas no JavaScript. Assim como as chamadas dos métodos de cada objeto.



O JavaScript permite omitir o objeto *window*, uma vez que considera todos os objetos da página estão abaixo desse objeto. Assim, o exemplo anterior pode ser escrito como: *document.form*.

Todo o objeto do JavaScript possui características que podem ser manipuladas. A essas características damos o nome de **propriedades**. Podemos dizer então que o objeto *form* é uma propriedade do objeto *document*. Assim como esse mesmo objeto *document* possui outras propriedades que não são objetos como a propriedade *title*.

Além das propriedades, existe mais um conjunto de atribuições que são denominados de **métodos**. Cada método está associado a um objeto existente no documento fazendo parte do seu escopo de atuação.



**Métodos** são funções predefinidas pela linguagem para executar uma operação específica. Caso necessitamos que algo seja escrito em um documento, podemos utilizar um método para exibir um texto qualquer.



**Atenção!**  
Nunca utilize métodos em objetos que não o contenham, caso contrário causará erro no *script*.

Os métodos são usados comumente para alterar o valor de uma propriedade ou executar tarefas distintas. A sintaxe é representada da seguinte forma:

```
nomeObjeto.metodo(argumento)
```

Vale ressaltar que o **argumento** pode ser opcional ou obrigatório, dependendo do método associado ao objeto.

Com essas informações, você sabe como é formada a linguagem, como o JavaScript foi construído. Assim, podemos sempre ter em mente que qualquer algoritmo desenvolvido sobre essa linguagem deverá sempre respeitar esses padrões.

O JavaScript pode ser escrito de duas formas: **embutida no código HTML** ou em um **arquivo externo com a extensão ".js"**.

Ambas as formas não alteram a sintaxe da linguagem. Porém, a forma que é feita a referência necessita de cuidado especial. Quando está embutida no código, devemos utilizar as *tags script* da seguinte forma:

```
<html>
<head>
<title>Teste</title>
<script language="javascript" type="text/javascript">
(...)
</script>
</head>
<body>
(...
</body>
</html>
```

Os atributos *language* e *type* não necessitam estar discriminados na *tag script*, porém a especificação técnica que foi construída pela **ECMAScript** recomenda o seu uso.

Quando queremos utilizar um arquivo externo, a *tag script* recebe um novo atributo. Esse atributo é onde indicamos o local e o nome do arquivo JS. Quanto ao arquivo, basta escrever o JavaScript normalmente. Veja a seguir:

Este padrão é o oficial reconhecido pelas indústrias.

```
<html>
<head>
<title>Teste</title>
<script language="javascript" type="text/javascript" src="tetsse.js">
</script>
</head>
<body>
...
</body>
</html>
```

Perceba que, quando chamamos um arquivo externo, a *tag* não pode conter um *script*. Caso seja necessário, basta abrir uma segunda *tag script* e escrever o código. A *tag script* pode ser escrita em qualquer parte do código, porém é aconselhável

utilizar antes do corpo da página, já que existem formas de realizar as chamadas JavaScript antes do carregamento total da página. Desta forma ocorrerá um erro caso a chamada venha antes da escrita da função.

Uma prática que os desenvolvedores utilizam para auxiliar na documentação do código em qualquer *software* é o uso dos comentários de código. Esse tipo de declaração é normalmente feita por meio de comandos especiais em que a linguagem permite que o desenvolvedor adicione qualquer tipo de informação indicado ao interpretador do programa que deve ignorar qualquer tipo de instrução contida. Assim, o JavaScript, como as demais linguagens, permite que seja comentado todo o código que produzimos, tornando-o mais legível e mais fácil de mantê-lo. Um comentário adequado do código que você produz torna mais fácil realizar alterações e melhorias mais tarde.

O compartilhamento de códigos (muito comum no desenvolvimento Web) com outras pessoas torna os comentários ainda mais importantes para que os outros entendam aquilo que você escreveu.

Em JavaScript podemos usar **comentários de linha** e **comentários de bloco**. Os comentários de linha começam com os caracteres `//`. Isso dá ao interpretador de JavaScript a indicação de que o resto da linha é um comentário. Deste modo, este ignora o resto da linha, continuando a interpretar o código na linha seguinte.

Um comentário de bloco inicia com a abertura do bloco por meio do comando `/*` e continua até ser encontrada a sequência de caracteres `*/`, que marcam o fim do comentário. Ao encontrar a sequência `/*`, o interpretador de JavaScript procura imediatamente a sequência de finalização `*/`, continuando aí a interpretação do código e ignorando o que está no meio.

A seguir estão alguns exemplos de comentários em JavaScript.

```
// Comentário de linha
/* Comentário de bloco, mas apenas utilizando uma linha */
/*
Comentário de bloco utilizando várias linhas. Qualquer comando escrito
dentro do bloco é ignorado pelo interpretador. Inclusive o comando de
abertura de bloco /* ou ainda o comentário de linha //
Todas as informações são ignoradas.
*/
```

Os **blocos** não são exclusividade dos comentários. Eles são encontrados em diversos comandos do JavaScript. Os blocos são encontrados nos laços de decisão e repetição. Ainda são encontrados nas funções para delimitar seu conteúdo. Isso se consegue colocando-as entre chaves ({}).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de bloco</title>
</head>
<script language="javascript" type="text/javascript">
function fatorial(valor) {
    if (valor<2) return valor;
    return valor*fatorial(valor-1);
}
function controlador(vl_valor) {
    return document.form.txt_fatorial.value=fatorial(vl_valor);
}
</script>
<body>
<form name="form" id="form" action="" method="get">
<table cellpadding="0" cellspacing="0" border="0">
<tr>
    <td> Digite um valor inteiro e positivo</td>
    <td><input type="text" name="txt_valor" id="txt_valor" value="">
/></td>
</tr>
<tr>
```

```
<td colspan="2" align="center"><input type="button"
name="btn_calcular" id="btn_calcular" value="Calcular"
onclick="controlador(document.form.txt_valor.value)" /></td>
</tr>
<tr>
<td> Fatorial</td>
<td><input type="text" name="txt_fatorial" id="txt_fatorial" value=""
readonly="readonly"/>
<td>
</tr>
</table>
</form>
</body>
</html>
```

Veja que nesse exemplo as funções estão delimitadas por blocos e seu conteúdo interno fica restrito ao bloco.

## Seção 2 – Tipos de dados

Assim como outras linguagens, o JavaScript possui uma **padronização** quanto aos tipos de objetos. Podemos relacionar os tipos: numérico, lógico, caractere.

O tipo caractere pode ser chamado também como *string*. Representa uma sequência de caracteres, podendo esta ser qualquer caractere (pontuação, símbolo, números, letras etc.). O seu limite de conteúdo dependerá do navegador que está interpretando. Os seus valores são delimitados por aspas simples ou duplas. Exemplo: ‘texto’ ou “texto”.

Toda a atribuição realizada a uma variável contendo uma *String* contém internamente na linguagem a herança do objeto *String*. Isso significa que essa variável tem um comportamento da uma instância do objeto *String*, estão aptas a utilizar as propriedades e os seus métodos.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de String</title>
</head>
<script language="javascript" type="text/javascript">
var str = "Teste de string!";
document.write("Escreve o texto da variável: ");
document.write(str);
document.write("<br>");
document.write("Escreve uma parte do conteúdo da variável: ");
document.write(str.substr(9,6));
</script>
<body>
</body>
</html>

```

Veja que no exemplo anterior está sendo utilizada uma função chamada *write*. Essa função escreve no HTML alguma informação. A variável **str** está sendo atribuída com um valor de caractere. Com isso, ela herda os métodos da classe *String* e podemos utilizar o método **substr**, que recupera um pedaço de uma *string*. O resultado no *browser* é:

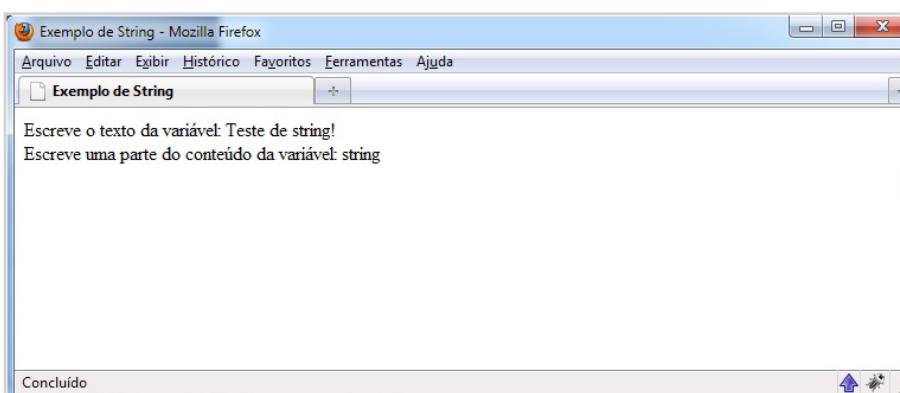


Figura 2.2 - Exemplo de *string*.

Fonte: Elaborada pelo autor (2010).

Essa característica se assemelha e muito com o método de orientação a objetos. Porém, não se pode considerar uma

linguagem orientada a objetos, já que não é obrigatória sua construção nessa metodologia.

Outro tipo de dados de suma importância são os numéricos. Esse tipo de dado armazena valores, tanto inteiros como ponto flutuante (decimal).

- 23
- 1
- 2e10
- 3.98733
- 8990.09889
- 0.0999898

Os valores numéricos são utilizados normalmente em operações aritméticas (adição, subtração, multiplicação e divisão). Podendo estarem sendo utilizados em laços de decisão e repetição.

O JavaScript possui uma boa quantidade de funções para manipulação de valores numéricos, como funções trigonométricas, funções de arredondamento e exponencial, funções de transformação de tipos, entre outras.

Existem alguns **valores numéricos especiais**, são eles:

- **NaN** – o valor NaN, nos tipos de dados numéricos, é a abreviação de *Not a Number* (Não é um Número). O JavaScript apura esse valor normalmente quando ocorre alguma operação inválida com o tipo de dados numéricos. Um exemplo clássico é o resultado da operação (0/0), resultará no NaN.
- **Infinity**: representa um valor infinito, podendo ser tanto positivo quanto negativo. Todas as operações com valores infinitos resultarão em um valor infinito, exceto divisão e subtração que resultará no NaN.

O JavaScript ainda possui outros tipos de dados, como: Lógicos, Null e Undefined.

- **Lógicos:** os valores lógicos podem assumir dois valores, *true* (verdadeiro) e *false* (falso). São utilizados normalmente como controle de decisão.
- **Null:** este tipo de dado é um valor especial, representa um objeto nulo, não deve ser confundido com uma variável não inicializada, pois o valor *null* existe. Portanto, uma variável com o conteúdo *null* existe em memória, referenciando este objeto especial.
- **Undefined:** o valor *undefined* significa que a variável não foi instanciada, ou seja, não foi atribuída um valor. Inicialmente todas as variáveis se encontram neste estado.

## Seção 3 – Elementos de linguagem

Embora JavaScript tenha algumas semelhanças com outras linguagens, ele é uma linguagem diferenciada em alguns aspectos. O JS possui variáveis, operadores lógicos, operadores aritméticos, laços de repetição, laços de decisão, entre outros. Nesta seção, abordaremos com mais detalhes alguns desses aspectos.

### a) Variáveis

As variáveis no JavaSscript podem ser de dois tipos: **variáveis globais** e **variáveis locais**. Ambas podem suportar quaisquer tipos de valores. Porém, o que diferencia é o escopo e a forma de declaração da variável.

A **variável local** para ser declarada deve ser antecedida pela palavra reservada ‘var’. Essa declaração pode ser feita a qualquer

momento dentro de uma função e poderá ser acessada somente dentro dela:

```
var nome;  
var ano=2009;
```

Ao tentar acessar uma variável local fora da função em que ela foi declarada, será gerado um erro porque a variável só existe no universo da função que foi declarada. Essa variável não faz parte do mundo exterior a essa função e como tal não pode ser utilizada.

A **variável global** deve ser antecedida pela palavra reservada ‘var’ ou receber uma atribuição de valor. O seu escopo é acessível em toda a página, sendo que não é transmitida de uma página para outra.

As variáveis podem ser declaradas da seguinte forma:

```
valor = 19;  
var numero = 3;  
fruta = "Morango";  
var fruta2 = "Banana";  
pi = 3.14159;
```

Nesses exemplos podemos considerar que todas as variáveis declaradas serão variáveis globais. Como já vimos, para declarar variáveis cuja existência se limita a uma pequena seção do código (variável local) teremos de usar a declaração **var**, assim: var numero = 3.

Caso usemos essa declaração fora de qualquer função, isto é, a variável é declarada na base da estrutura de código, ela será global. Temos assim que a declaração *var* serve para limitar o contexto em que a variável existe e que:

- as variáveis declaradas sem a declaração *var* são variáveis globais;
- as variáveis declaradas usando a declaração *var* existem apenas no contexto em que foram definidas.

Antes de começar a escrever código em JavaScript, é importante planejar o modo como este será organizado. Deve-se começar a identificar os dados que serão utilizados. A seguir escolhem-se os nomes das variáveis que guardarão esses dados e só depois é que se começa a escrever o código propriamente dito.

Além do seu escopo, algumas regras quanto ao nome devem ser respeitadas para que o interpretador possa trabalhar livremente:

- todos os nomes têm de começar com uma letra ou com o caractere;
- os restantes caracteres que compõem o nome podem igualmente conter números. Nunca se esqueça que para o JavaScript letra maiúscula e letra minúscula são coisas diferentes e que, por exemplo, as variáveis variavel1, Variavel1 e vaRiavel1 são três objetos distintos.

Muitas vezes nos deparamos com a necessidade de escrever algum caractere dentro de uma *String*, mas seu uso é reservado como o caso das aspas. Para isso, utiliza-se a contrabarra dentro da *string*. Isso indica para o *browser* que a sua utilização faz parte da *string*, e não um símbolo de comando.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de String</title>
</head>
<script language="javascript" type="text/javascript">
var str = "O texto contém \"aspas duplas\" e 'aspas simples'";
document.write("Verificando o caso das \"aspas\"<br>");
document.write(str);
</script>
<body>
</body>
</html>
```

Você pode notar que somente é obrigatório o uso da contrabarra quando o interpretador pode confundir o término de uma *string*.

Veja que a *string* está delimitada por aspas duplas e quando é utilizada aspas simples dentro da *string* o uso da contrabarra é opcional. Desta forma, o *browser* interpretará o comando.

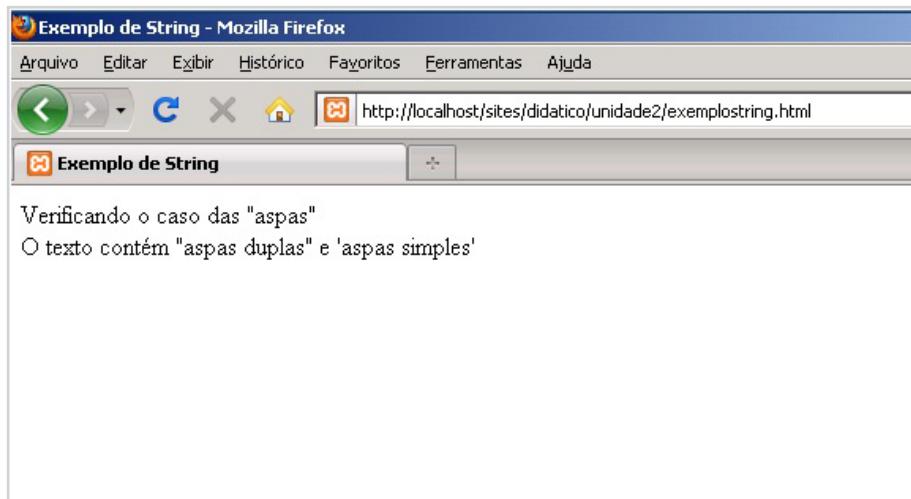


Figura 2.3 - Exemplo de contrabarra.

Fonte: Elaborada pelo autor (2010).

Assim como outras linguagens, o Javascript necessita de alguns **caracteres especiais** para poder armazenar ou imprimir alguma informação que tenha conflito com sua linguagem. Veja o quadro a seguir:

Tabela 2 – Referente a caracteres especiais.

Caractere	Descrição
\n	Insere uma quebra de linha.
\t	Insere uma tabulação.
\r	Insere um retorno.
\f	Insere um caractere de barra.
\t	Tabulação.
\'	Apóstrofo.
\"	Aspas.
\	Barra invertida.
\XXX	Caractere representado pela codificação Latin-1. Exemplo \251 representa o caractere de copyright ©.

Fonte: Javascript – aplicações interativas para a Web (2006).

Representando os códigos dentro de um *script* teríamos:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de contra-barra</title>
</head>

<body>
<script language="javascript" type="text/javascript">
var str="";
str+="Teste de \b backspace\n";// - Representa o backspace (caracter ascii 8)
str+="Tabulação entre\ttextos\n";// - Tabulação (caracter ascii 9)
str+="quebra de \r parágrafo\n";// – Quebra de parágrafo (caracter ascii 13)
str+="quebra de \n linha\n";// – Quebra de linha (caracter ascii 10)
str+="tabulação\vvertical\n                                "// - Tabulação
vertical (caracter ascii 11)
str+="Caracter unicode \u0198\n";// - Caracter unicode (NNNN representa
um valor hexadecimal de 0000 a FFFF)
str+="Caracter Asc \x28\n";// - Caracter ascii (NN representa um valor
hexadecimal de 00 a FF)
str+="Teste de \'apóstrofo\'\n";// - Apóstrofo
str+="Teste de \"aspas\"\n";// - Aspas
str+="Teste de \\barra inversa\n";// - Barra inversa
alert(str);
</script>
</body>
</html>
```

O *browser* interpretará da seguinte forma:

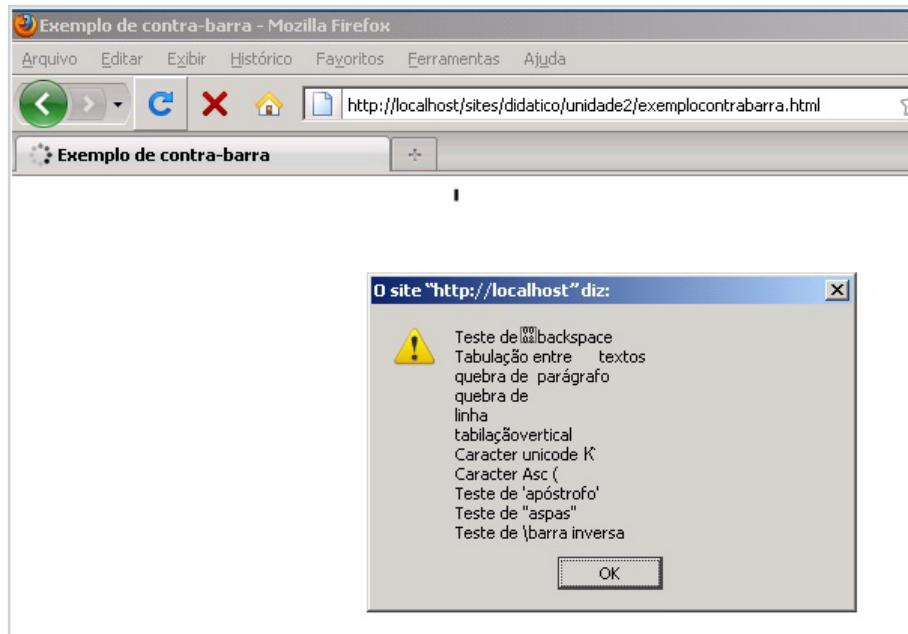


Figura 2.4 – Exemplo do uso de caracteres especiais.

Fonte: Elaborada pelo autor (2010)

Perceba que a interpretação dos comandos depende não somente do JavaScript mas do meio que irá ler. Um exemplo seria a quebra de linha: para o JavaScript, a quebra de linha é o caractere \n, porém para o *browser* isso só pode ocorrer por meio do HTML por meio da tag <BR>. Assim como o comando \r, o JavaScript não possui quebra de parágrafo, e não faz sua interpretação. Porém, é um comando padrão para diversos editores de texto conseguirem identificar essa quebra.

Ao se trabalhar com variáveis, acaba sendo inevitável demandar uma atenção especial à conversão de valores para que determinado valor tenha um comportamento mais adequado ao seu código, apesar de o JavaScript exigir pouco trabalho ao desenvolvedor para definir o tipo de dados que uma variável deve guardar. Acaba sendo o próprio interpretador de JavaScript que decide o tipo de dados que a variável será. Assim, se escrever:

```
var nota = 9;
```

O interpretador decidirá guardar internamente a variável ‘nota’ como um número inteiro, mas se escrevermos:

```
var nota = 9;
```

```
nota = "O JavaScript necessita de atenção.";
```

Quando o interpretador lê a segunda linha de código mudará o tipo de dado da variável e a variável ‘nota’ deixará de ser armazenada como um número inteiro. Ela passa a ser armazenada como uma *String*.

*Essa conversão no tipo da variável acontece de forma automática e o desenvolvedor não precisa fazer nada para que ela aconteça.*

Essa liberdade serve apenas para simplificar a escrita do código. Quando é mal utilizada, ela pode dar origem a código difícil de ler e a erros. As regras de boa programação dizem que ao definir uma variável o desenvolvedor deve decidir qual o tipo de dados que esta conterá e não deverá escrever código que provoque uma conversão no tipo de dados que a variável guarda. Sempre que uma conversão for necessária deverá ser definida uma nova variável para guardar o resultado da conversão, mantendo inalterados os tipos das variáveis antigas.

As variáveis podem receber diversos tipos de dados. Na prática esses dados são convertidos em objetos pelo interpretador do JavaScript. Portanto, quando você define uma variável do tipo *string*, o interpretador considerará que essa variável contém um objeto *string*. Essa conversão proporciona o acesso a diversos métodos específicos para cada tipo de dados a ser atribuído.

Um objeto muito comum na orientação a objetos é o *this*. O objeto *this* pode ser considerado como coringa. Ele representa o objeto atual de trabalho. Normalmente é utilizado no JavaScript para referenciar os argumentos dentro das funções. Ele representa o próprio objeto em que é usado. Esse objeto pode ser uma função, uma ligação de hipertexto, uma imagem etc.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo Objeto This</title>
</head>
<script type="text/javascript" language="javascript">
function testedeobjetos(objeto) {
    alert("Atributo Type: "+objeto.type+"\nAtributo Name: "+objeto.name+"\nAtributo Value: "+objeto.value);
}
</script>
<body>
<h1>Ao sair do campo irá emitir um alerta informando alguns atributos de cada campo</h1>
<form name="form" method="get">
Código:&nbsp;<input name="txt_codigo" type="text" value="" onblur="testedeobjetos(this)" /><br />
Nome:&nbsp;<input name="txt_nome" type="text" value="" onblur="testedeobjetos(this)" /><br />
Endereço:&nbsp;<input name="txt_endereco" type="text" value="" onblur="testedeobjetos(this)" /><br />
Complemento:&nbsp;<input name="txt_complemento" type="text" value="" value="" onblur="testedeobjetos(this)" /><br />
</form>
</body>
</html>
```

Observe que o código de chamada da função não se altera. O objeto *this* assume por referência o objeto padrão de chamada. No caso do exemplo, a cada chamada ele assume a tag *input* como sendo o elemento base ao qual está se trabalhando. Quando se altera o objeto, altera o valor representativo do *this*.

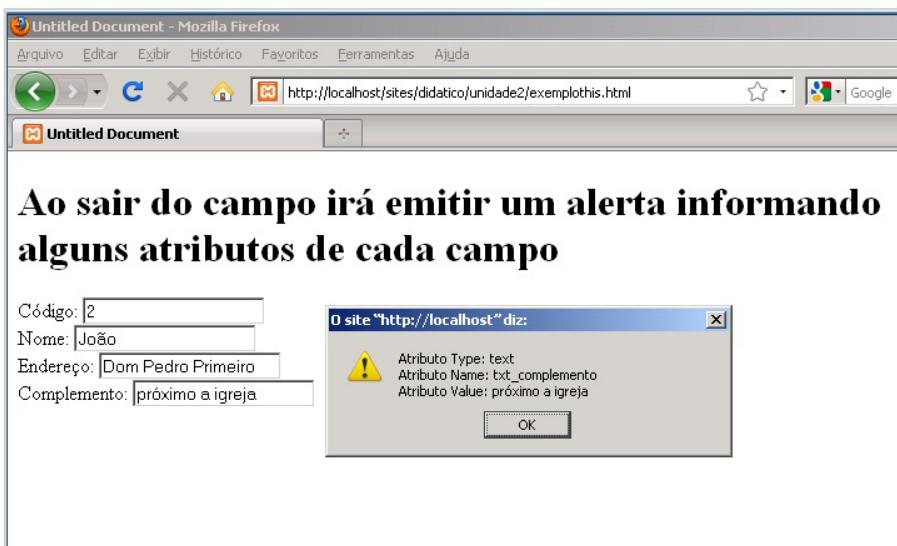


Figura 2.5 - Exemplo do objeto this.

Fonte: Elaborada pelo autor (2010)

O objeto *string* possui métodos que podem ser utilizados em diversas situações e auxiliam o desenvolvimento das aplicações.

Método	Descrição
charAt(índice)	Devolve o caractere que ocupa a posição índice na <i>string</i> .
charCodeAt(índice)	Devolve o código (conjunto Unicode) do caractere que ocupa a posição índice na <i>string</i> .
indexOf(string_busca, índice_opcional)	Devolve a posição em que se encontra a primeira ocorrência de <i>string_busca</i> ou -1 se essa ocorrência não existir. Se não fornecermos um <i>índice_opcional</i> , a busca inicia-se na posição zero, mas se o fornecermos é nesse ponto que se inicia a busca.
lastIndexOf(string_busca, índice_opcional)	Devolve a posição em que se encontra a última ocorrência de <i>string_busca</i> ou -1 se essa ocorrência não existir. A busca faz-se a partir do fim e caminha progressivamente para o início. Se não fornecermos um <i>índice_opcional</i> , a busca inicia-se no fim, mas se o fornecermos é nesse ponto que se inicia a busca.

Método	Descrição
split(string_separador, limite_opcional)	Divide uma <i>string</i> em partes usando as ocorrências de <i>string_separador</i> como pontos de divisão. Devolve um <i>array</i> com todas as divisões ( <i>substrings</i> ) encontradas. Cada elemento do <i>array</i> é uma <i>substring</i> da <i>string</i> original. O <i>limite_opcional</i> indica o número máximo de partes a incluir no <i>array</i> que é devolvido. A <i>string_separador</i> é excluída das divisões e o objeto <i>String</i> sobre o qual foi invocado este método não sofre alterações.
substring(início, fim)	Devolve uma sessão da <i>string</i> composta pelos caracteres que ocupam as posições com índices entre início (incluída) e fim (excluída).
toLowerCase()	Devolve uma versão da <i>string</i> com todos os caracteres convertidos para letra pequena.
toUpperCase()	Devolve uma versão da <i>string</i> com todos os caracteres convertidos para letra grande.

Fonte: JavaScript e HTML Dinâmico (2007).

Toda a variável do tipo *string* possui acesso aos métodos descritos e a alguns outros.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo Métodos do objeto String</title>
</head>
<body>
<h3>Propriedade length</h3>
<script type="text/javascript">
    var str="Não há nada melhor que uma cervejinha gelada na sexta à noite.";
    document.write('A String <b> '+ str + "</b> possui ");

```

```
document.write(str.length+" caracteres");

</script>

<h3>Método IndexOf</h3>

<h6>Este exemplo verifica se uma string (variável de texto)
contém uma determinada palavra. Se a palavra for
encontrada será indicada a posição em que foi localizado o seu
primeiro caractere.<br />

<b>Nota:</b> A posição do primeiro caractere da string é 0, não é 1. </
h6>

<script type="text/javascript">

    var str="Não há nada melhor que uma cervejinha gelada na sexta à
noite./";

    var pos=str.indexOf("cervejinha ");

    if (pos>=0) {

        document.write('a palavra "cervejinha" foi encontrada na
posição: ');

        document.write(pos + "<br>");

    } else{

        document.write('a palavra "cervejinha" não foi encontrada!');

    }

</script>

<h3>Método Match</h3>

<h6>Este exemplo verifica se uma palavra (que neste
caso é "gelada") está contida na string. Se estiver
ela será devolvida.</h6>

<script type="text/javascript">

var str="Não há nada melhor que uma cervejinha gelada na sexta à
noite./";

document.write(str.match("gelada"));

</script>

<h3> Método substring</h3>
```

```
<script type="text/javascript">

    var str="Não há nada melhor que uma cervejinha gelada na sexta à
    noite./";

    document.write("str.substring(27,31) = "+str.substring(27,31));
    document.write("<br>");
    document.write("str.substring(27,36)="+str.substring(27,36));

</script>

<h3>Converte para Maiuscula e Minúscula</h3>

<script type="text/javascript">

    var str="Olá Pessoal do JavaScript!";
    document.write(str.toLowerCase());
    document.write("<br>");
    document.write(str.toUpperCase());

</script>

</body>
</html>
```

Fonte: JavaScript e HTML Dinâmico (2007).



Figura 2.6 - Propriedades e métodos de strings.

Fonte: Elaborada pelo autor (2010).

Além do objeto *string*, o JavaScript oferece o objeto *date*. Esse objeto possui a característica de manipular datas e horas. A definição de valores é possível utilizando valores numéricos e datas por extenso. Os métodos atrelados a esse objeto são:

Método	Descrição
getDate()	Devolve o dia do mês (de 1 a 31) que está guardado no objeto Date.
getDay()	Devolve o dia da semana (de 0 = domingo até 6 = sábado) guardado no objeto Date.
getMonth()	Devolve o mês (de 0 = janeiro até 11 = dezembro) guardado no objeto Date.
getFullYear()	Devolve o valor completo do ano (quatro dígitos) guardado no objeto Date.
getYear()	Devolve o valor incompleto do ano (apenas dois dígitos) guardado no objeto Date. Não use este método, use getFullYear em seu lugar.
getHours()	Devolve o valor da hora (de 0 a 23) guardada no objeto Date.
getMinutes()	Devolve o valor dos minutos (de 0 a 59) guardados no objeto Date.
getSeconds()	Devolve o valor dos segundos (de 0 a 59) guardados no objeto Date.
getMilliseconds()	Devolve o valor dos milissegundos (de 0 a 999) guardados no objeto Date.
getTime()	Devolve a quantidade de milissegundos decorridos desde zero hora do dia 1 de janeiro de 1970 até a data que está guardada no objeto Date.
getTimezoneOffset()	Devolve a diferença de tempo entre o fuso horário do computador que está a ser usado e o Tempo Médio de Greenwich ( <i>Greenwich Mean Time</i> , ou <i>GMT</i> ).
getUTCDate()	Devolve o dia do mês (de 1 a 31) guardado no objeto Date medido no padrão de tempo universal (UTC).
getUTCDay()	Devolve o dia da semana (de 0 a 6) guardado no objeto Date medido no padrão de tempo universal (UTC).
getUTCMonth()	Devolve o valor do mês (de 0 a 11) guardado no objeto Date medido no padrão de tempo universal (UTC).
getUTCFullYear()	Devolve o valor completo do ano (com quatro dígitos) guardado no objeto Date medido no padrão de tempo universal (UTC).

Método	Descrição
getUTCHours()	Devolve a hora (de 0 a 23) guardada no objeto Date medida no padrão de tempo universal (UTC).
getUTCMinutes()	Devolve o valor dos minutos (de 0 a 59) guardados no objeto Date medidos no padrão de tempo universal (UTC).
getUTCSeconds()	Devolve o valor dos segundos (de 0 a 59) guardados no objeto Date medidos no padrão de tempo universal (UTC).
getUTCMilliseconds()	Devolve o valor dos milissegundos (de 0 a 999) guardados no objeto Date medidos no padrão de tempo universal (UTC).
setDate(dia_mês)	Acerta o valor do dia (de 1 a 31) do mês guardado no objeto Date.
setFullYear(ano)	Acerta o valor do ano (com quatro dígitos) guardado no objeto Date.
setHours(horas)	Acerta o valor da hora (de 0 a 23) guardada no objeto Date.
setMilliseconds(milisegundos)	Acerta o valor dos milissegundos (de 0 a 999) guardados no objeto Date.
setMinutes(minutos)	Acerta o valor dos minutos (de 0 a 59) guardados no objeto Date.
setMonth(mês)	Acerta o valor do mês (de 0 = janeiro a 11 = dezembro) guardado no objeto Date.
setSeconds(segundos)	Acerta o valor dos segundos (de 0 a 59) guardados no objeto Date.
setTime(data_numérica)	Acerta a data e a hora guardadas no objeto Date para o valor correspondente a zero hora do dia 1 de janeiro de 1970 mais o número de milissegundos que é fornecido como argumento do método.
setYear(ano)	Acerta o valor do ano guardado no objeto Date. Não use este método, use antes o método setFullYear.
setUTCDate(dia_mês)	Acerta o valor do dia (de 1 a 31) do mês guardado no objeto Date usando o padrão de tempo universal (UTC).
setUTCDay(dia_semana)	Acerta o valor do dia da semana (de 0 = domingo a 6 = sábado) guardado no objeto Date usando o padrão de tempo universal (UTC).
setUTCMonth(mês)	Acerta o valor do mês (de 0 = janeiro a 11 = dezembro) guardado no objeto Date usando o padrão de tempo universal (UTC).

Método	Descrição
setUTCFullYear(ano)	Acerta o valor do ano (com quatro dígitos) guardado no objeto Date.
setUTCHour(horas)	Acerta o valor da hora (de 0 a 23) guardada no objeto Date usando o padrão de tempo universal (UTC).
setUTCMinutes(minutos)	Acerta o valor dos minutos (de 0 a 59) guardados no objeto Date usando o padrão de tempo universal (UTC).
setUTCSeconds(segundos)	Acerta o valor dos segundos (de 0 a 59) guardados no objeto Date usando o padrão de tempo universal (UTC).
setUTCMilliseconds(milisegundos)	Acerta o valor dos milissegundos (de 0 a 999) guardados no objeto Date usando o padrão de tempo universal (UTC).
toGMTString()	Devolve uma representação textual (a data e a hora escritas por extenso) usando como referência o fuso horário do Tempo Médio de Greenwich (GMT).
toLocaleString()	Devolve uma representação textual (a data e a hora escritas por extenso) no fuso horário do computador local.
toUTCString()	Fornece uma representação textual (a data e a hora escritas por extenso) da data contida no objeto Date medida no padrão UTC.

Fonte: JavaScript e HTML Dinâmico (2007).

A utilização do objeto *date*, assim como seus métodos, é feita de forma simplificada.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplos objeto Date</title>
</head>
<body>
<h3>Mostrar o dia da semana</h3>
<script type="text/javascript">
    var d = new Date();
    var dia = d.getDay();
    document.write("O número do dia da semana de hoje é "+dia);
    // Para obtermos o nome do dia, criamos um Array em que Domingo
    // ocupa a
    // posição 0, segunda ocupa a posição 1, ...
    var nomesDias=new Array("Domingo","Segunda-feira","Terça-
    feira","Quarta-feira","Quinta-feira","Sexta-feira","Sábado");
    document.write("<br><br>O nome do dia de hoje é "+nomesDias[dia]);
</script>
<h3>Obter a hora marcada pelo relógio do seu computador</h3>
<script type="text/javascript">
    var d = new Date();
    document.write(d.getHours()+" horas ");
    document.write(d.getMinutes()+" minutos ");
    document.write(d.getSeconds()+" segundos");
</script>
<h3>Criar uma data com um valor predefinido</h3>
<script type="text/javascript">
    var data1 = new Date("Fri, 21 Nov 2003 10:43:34 UTC");
    document.write("A data1 é: "+data1.toLocaleString());
</script>
```

```
document.write("<br><br>");

var data2 = new Date(1069411529550);

document.write("A data2 é: "+data2.toLocaleString());

</script>

<h3>Estabelecer o ano para uma data</h3>

<script type="text/javascript">

    var d = new Date();

    d.setFullYear("1998");

    document.write(d.toLocaleString());

</script>

<h3>Escrever a data completa com os nomes do dia e do mês</h3>

<script type="text/javascript">

    var d = new Date();

    var dia = d.getDay();

    var mes = d.getMonth();

    var ano = d.getFullYear();

    // Para obtermos o nome do dia criamos um Array em que domingo
    // ocupa a

    // posição 0, segunda ocupa a posição 1, ...

    var nomesDias=new Array("Domingo","Segunda-feira","Terça-
    feira","Quarta-feira","Quinta-feira","Sexta-feira","Sábado");

    var nomesMeses=new Array("Janeiro","Fevereiro","Março","Abril","Maio","J
    unho","Julho","Agosto","Setembro","Outubro","Novembro","Dezembro");

    var s="Hoje é "+nomesDias[dia]+", "+d.getDate();

    s+=" de "+nomesMeses[mes]+" de "+ano;

    document.write(s);

</script>

</body>

</html>
```

Fonte: JavaScript e HTML Dinâmico (2007).

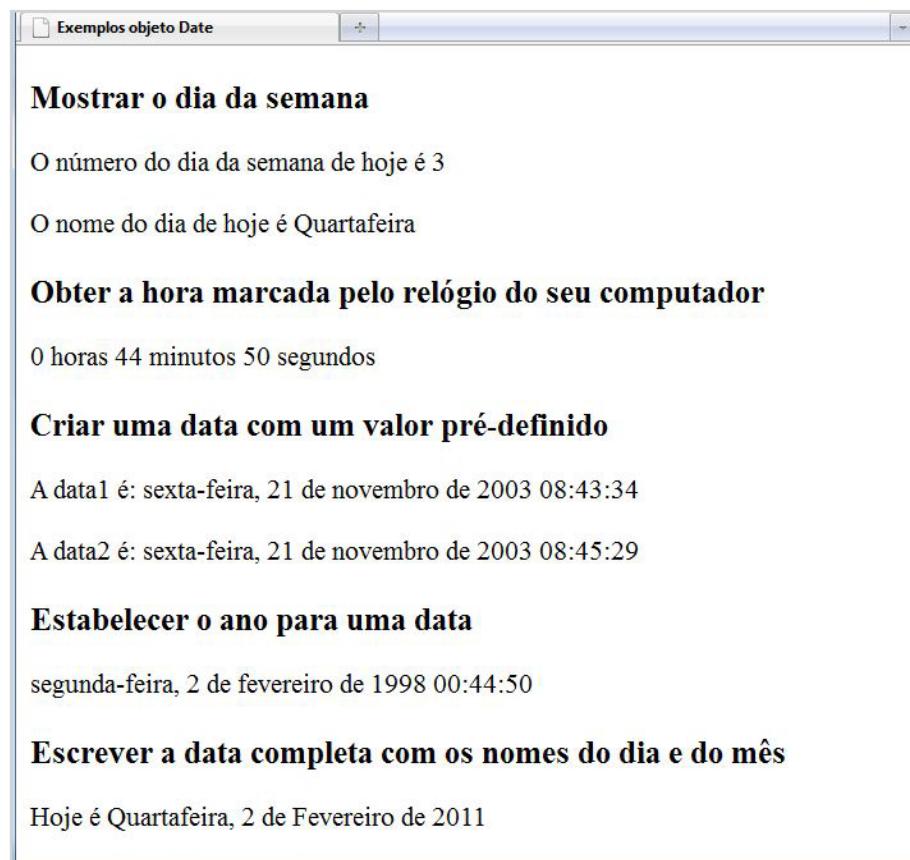


Figura 2.7 - Métodos e atributos de data.

Fonte: Elaborada pelo autor (2010).

Um tipo especial de variável é o tipo de **vetor** ou **cadeia de variáveis**. Um vetor é um objeto capaz de guardar muitos valores, tantos quanto a memória disponível na máquina permitir. Um vetor é caracterizado como lacunas indexadas na memória que armazenam um conjunto finito de valores. Cada uma dessas lacunas que compõem o vetor possui um índice.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de Array</title>
</head>
<body>
<script type="text/javascript" language="javascript">
var numeros_primos = new Array( 2, 3, 5,7);
var nomes = new Array(3);
nomes[0] = "João";
nomes[1] = "Maria";
nomes[2] = "Joaquim";
for (var i=0; i<numeros_primos.length; i++)
    document.write("Posição "+(i+1)+" :" +numeros_primos[i]+ "<br>");
</script>
<br />
<script type="text/javascript" language="javascript">
for (var i=0; i<nomes.length; i++)
    document.write("Posição "+(i+1)+" :" +nomes[i]+ "<br>");
</script>
</body>
</html>
```

Ao ser declarada a variável ‘numeros\_primos’, declaramos o *Array* (vetor) e atribuímos os valores em uma única operação. Já no segundo caso, declaramos primeiro o *Array* e somente depois definimos os valores que ele deve conter. O *browser* interpretará da seguinte forma:

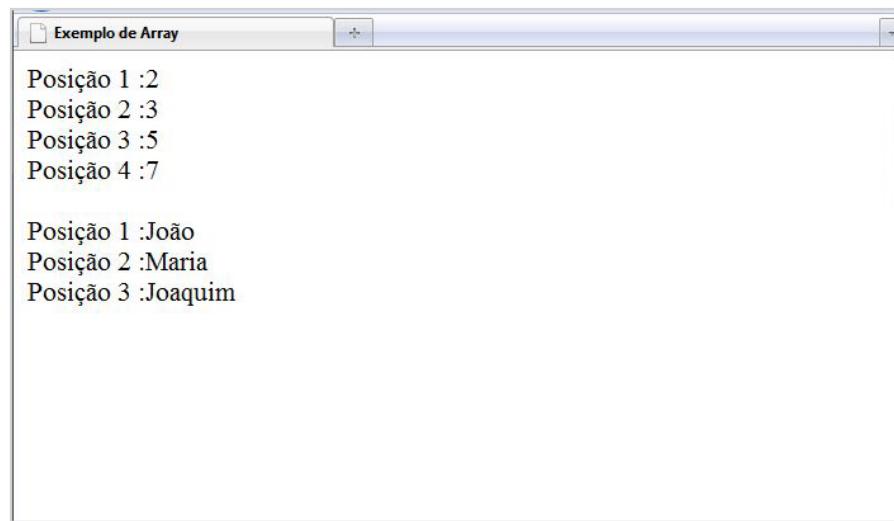


Figura 2.8 - Exemplo de array

Fonte: Elaborada pelo autor (2010).

Já vimos que o JavaScript converte o tipo de dados conforme a atribuição que realizamos, por isso um *array* pode conter valores de tipos diferentes em cada lacuna. Podemos alterar sempre que necessário, como mostra a seguir:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo conversão de dados no Array</title>
</head>
<body>
<script type="text/javascript">
var produto = new Array(45, "Morango", false, 1.25, "Biscoito", "Wafer");
document.write("Produto = " + produto);
produto[0] = 2;
produto[2] = true;
produto[6] = "Bolacha";
document.write("<br>");
document.write("Produto = " + produto);
</script>
</body>
</html>
```

Quando atribuirmos um valor a um elemento do vetor com um índice mais alto do que o seu comprimento, o interpretador do JavaScript aumenta o tamanho do vetor até chegar ao índice pretendido. É isso que acontece no exemplo anterior quando se chega à linha que tem:

```
produto[6] = "Bolacha";
```

O *browser* interpretará da seguinte forma:

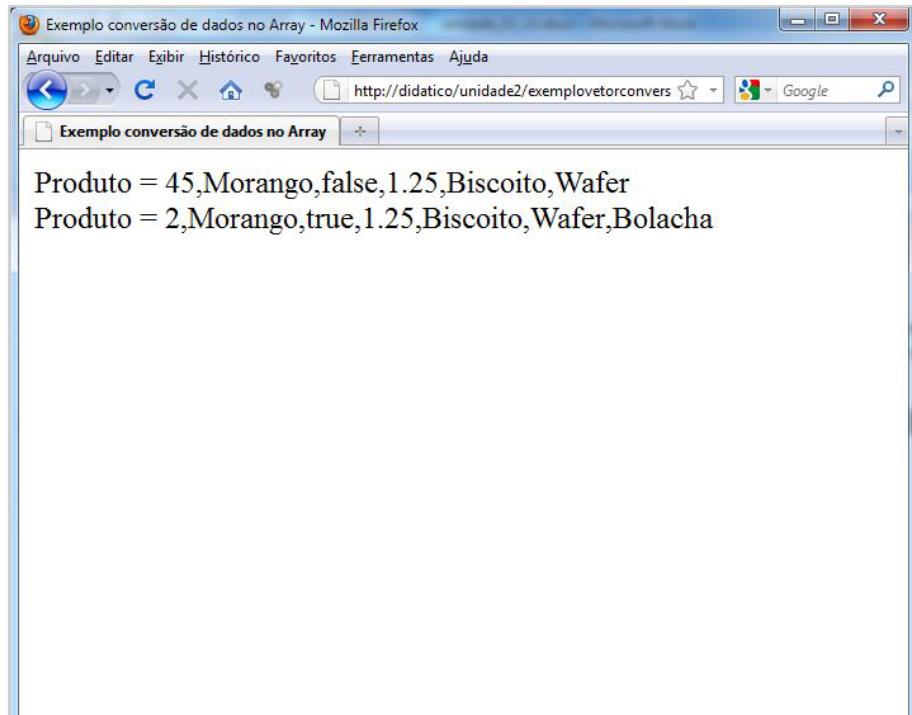


Figura2.9 -Manipulação de arrays.

Fonte: Elaborada pelo autor (2010).

Os vetores são objetos e algumas propriedades entre as quais a *length* é utilizada para contar o número de elementos dentro de um vetor. Assim, se ao exemplo anterior juntarmos uma linha com o seguinte código:

```
var numeroDeElementos = produto.length;
```

A variável ‘numeroDeElementos’ ficará com o valor 7. Entenda que a propriedade *length* não lê os índices, mas conta o número de elementos no vetor. Lembre-se sempre que índice e tamanho são diferentes.

O *array* é um objeto e como tal possui propriedades que auxiliam no desenvolvimento de aplicações.

Propriedade	Descrição
length	Fornece o número de elementos que estão no <i>Array</i> . O valor desta propriedade é estabelecido quando o <i>Array</i> é criado, mas pode crescer se formos juntando mais elementos ao <i>Array</i> .
Método	Descrição
join()	Devolve uma <i>String</i> (variável de texto) que representa o conteúdo do <i>Array</i> .
reverse()	Devolve um novo <i>Array</i> em que a ordem dos elementos está invertida (em particular temos que o elemento que tinha o índice zero passa a ter o índice mais alto e vice-versa).
sort()	Devolve um novo <i>Array</i> em que os elementos estão ordenados (geralmente por ordem crescente).

Fonte: JavaScript e HTML Dinâmico (2007).

Utilizando o método de ordenação sobre um objeto *array* teríamos:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>Exemplo ordenação array</title>

</head>

<body>

<script type="text/javascript">

var fruta = new Array("Laranja","Banana","Morango","Abacaxi","Maçã","Kiwi");

var numeros = new Array(27, 12, 15, 6, 2, 6);

</script>

<table border="1">
```

```
<tr>
<th colspan="2">Ordenar Strings</th>
<th colspan="2">Ordenar Número</th>
<tr>
<td width="140"><b>Ordem normal</b></td>
<td><b>Ordem alfabética</b></td>
<td width="140"><b>Ordem normal</b></td>
<td><b>Ordem alfabética</b></td>
</tr>
<td>
<script type="text/javascript">
for (var i=0; i<6; i++)
document.write(fruta[i] + "<br>")
</script>
</td>
<td>
<script type="text/javascript">
// Agora ordenamos o Array por ordem alfabética invocando o
// método sort()
fruta.sort()
for (var i=0; i<6; i++)
document.write(fruta[i] + "<br>")
</script>
</td>
<td>
<script type="text/javascript">
<!--
for (var i=0; i<6; i++)
document.write(numeros[i] + "<br>")
// -->
```

```

</script>
</td>
<td>
<script type="text/javascript">
<!--

// Agora ordenamos o Array por ordem alfabética invocando o
//método sort()
numeros.sort()

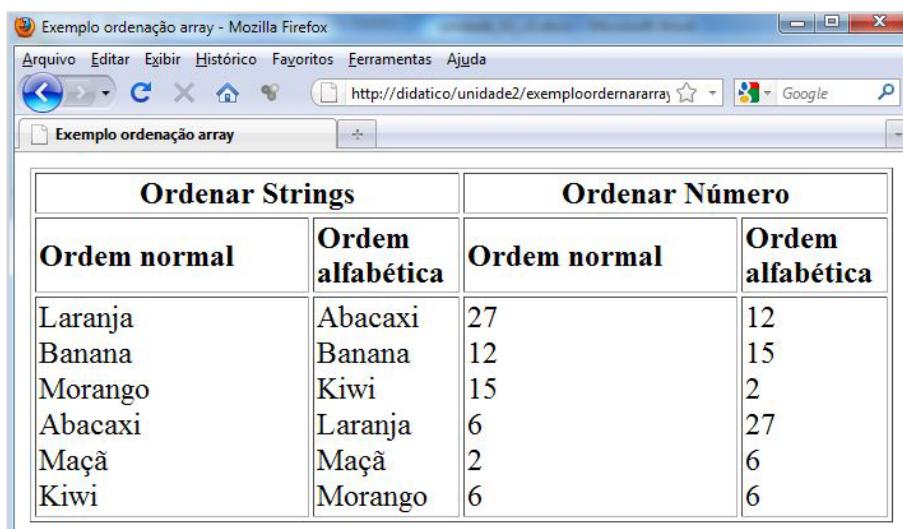
for (var i=0; i&lt;6; i++)
document.write(numeros[i] + "&lt;br&gt;")

// --&gt;

&lt;/script&gt;
&lt;/td&gt;
&lt;/tr&gt;
&lt;/table&gt;
&lt;/body&gt;
&lt;/html&gt;
</pre>

```

Fonte: JavaScript e HTML Dinâmico (2007).



The screenshot shows a Mozilla Firefox window with the title "Exemplo ordenação array - Mozilla Firefox". The address bar displays the URL "http://didatico/unidade2/exemploordernararray". The main content area contains a table with two sections: "Ordenar Strings" and "Ordenar Número".

**Ordenar Strings**

Ordem normal	Ordem alfabetica
Laranja	Abacaxi
Banana	Banana
Morango	Kiwi
Abacaxi	Laranja
Maçã	Maçã
Kiwi	Morango

**Ordenar Número**

Ordem normal	Ordem alfabetica
27	12
12	15
15	2
6	27
2	6
6	6

Figura 2.10 - Propriedade e métodos do array.

Fonte: Elaborada pelo autor (2010).

## b. Operadores

Os operadores são instruções utilizadas em expressões para validá-las ou calculá-las. Há dois tipos de operadores: **lógicos** e **aritméticos**. Os operadores lógicos são aqueles destinados a realizar comparações entre expressões. Normalmente essas comparações estão associadas aos laços de condicionais e os de repetição. Os operadores mais usados são:

Tabela 3 – Operadores lógicos.

Operador	Descrição
&& ou AND	E
ou OR	OU
! ou NOT	NÃO
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual
=	Atribuição
==	Igualdade
====	Idêntico (valor e tipo)
!=	Diferente

Como exemplo, temos:

```
<html>
<head>
<title>Teste</title>
<script language="javascript" type="text/javascript">
var exemplo1= "A";
if (exemplo1=="A") {
    alert("A condição é verdadeira");
} else {
    alert("A condição é falsa");
}
</script>
</head>
<body>
...
</body>
</html>
```

Os operadores aritméticos, como o próprio nome já diz, são usados em operações matemáticas. Veja a relação dos principais operadores:

Tabela 4 – Operadores aritméticos

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão



O operador ‘+’ é usado como concatenador de expressões. Portanto, se você usar esse operador em uma expressão como esta: “1”+1 o resultado será “11”.

O Javascript suporta ainda os operadores de **incremento** que são um contração ou atalho para uma operação. Ou seja, quando queremos realizar um contador aem vez de somar um a um a variável, podemos usar esse operador. A sintaxe é:

```
variável++  
variável--
```

### c. Laço de decisão

O laço de decisão no JavaScript é representado pela instrução *if*. Essa instrução permite executar um bloco de código apenas se for verdadeira uma determinada condição. Caso a condição não seja verdadeira, esse bloco de código não será executado, podendo ser ou não executado outro código alternativo, que será especificado por meio da instrução *else*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml">  
  
<head>  
  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"  
/>  
  
<title>Exemplo de laço de decisão</title>  
  
</head>  
  
<body>  
  
<script type="text/javascript" language="javascript">  
  
function verifica(campo) {  
  
    if(campo[0].checked) alert("Levar um gurada-chuva");  
  
    else alert("Deve estar quente!");  
  
}
```

```
</script>

<form name="form" id="form">

Está chovendo?&nbsp;

<p>

<label>

<input type="radio" name="chuva" value="s" id="chuva_0"
onchange="verifica(document.form.chuva)"/>

Sim</label>

<br />

<label>

<input type="radio" name="chuva" value="n" id="chuva_1"
onchange="verifica(document.form.chuva)"/>

<br />

</p>

<input type="button" name="consultar" value="Consultar"
onclick="verifica(document.form.chuva)"/>

</form>

</body>

</html>
```

O exemplo trata de uma interação com o usuário sobre o tempo. A resposta do usuário resultará em um teste lógico que mostrará uma mensagem de retorno. O *browser* interpreta da seguinte forma:

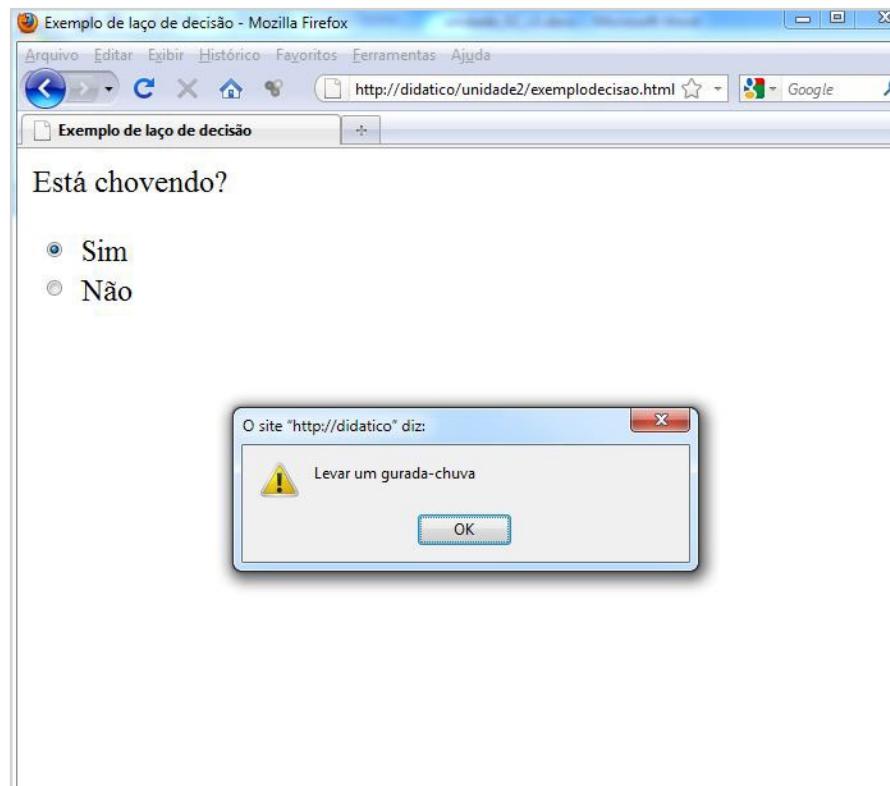


Figura 2.11 - Exemplo do controlador IF.

Fonte: Elaborada pelo autor (2010).

Uma instrução *if* não precisa ser associada a uma instrução *else*. Quando isso acontece, caso a condição não seja verdade, não será executado qualquer código alternativo. O interpretador apontará imediatamente a instrução após o término do comando *if*.

Outro laço de repetição suportado pelo JavaScript é o *switch*. Esse laço não faz parte do padrão ECMAScript, porém é suportado pela maioria dos *browsers*.

O *switch* possui a característica de comparar uma variável com diversos valores, executando blocos independentes para cada valor verdadeiro. Em casos como esse, o *switch* torna-se mais eficiente que a instrução *if* por possuir uma construção mais simplificada e organizada.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de Switch</title>
</head>
<script language="javascript" type="text/javascript">
function testacor(campocor) {
    switch (campocor) {
        case "azul":
            alert("O Azul pode ser usado para escrever um documento");
            break;
        case "vermelho":
            alert("O Vermelho é utilizado em correção");
            break;
        case "verde":
            alert("O Verde é utilizado para marcação");
            break;
        case "amarelo":
            alert("O Amarelo é utilizado para destacar texto");
            break;
        default :
            alert("O Preto é o padrão");
            break;
    }
}
</script>
<body>
<form name="form" action="" method="get">
```

```
Escolha uma cor: &nbsp;

<select name="cor" onchange="testacor(this.value)">

    <optgroup style="color:#000">
        <option value="" selected="selected">Preto</option>
    </optgroup>

    <optgroup style="color:#009">
        <option value="azul">Azul</option>
    </optgroup>

    <optgroup style="color:#F00">
        <option value="vermelho">Vermelho</option>
    </optgroup>

    <optgroup style="color:#0F0">
        <option value="verde">Verde</option>
    </optgroup>

    <optgroup style="color:#FF0">
        <option value="amarelo">Amarelo</option>
    </optgroup>

</select>

</form>

</body>

</html>
```

Perceba que o *switch* abre um bloco e dentro dele são inseridas as constantes a serem testadas. Cada constante é precedida da instrução *case* e deve abrir um novo bloco de comando interno. Neste caso, usa-se o caractere dois pontos (:). O bloco de instrução de cada *case* deve ser encerrado com a instrução *break*, que indica uma interrupção da execução de blocos. O *browser* interpreta da seguinte forma:

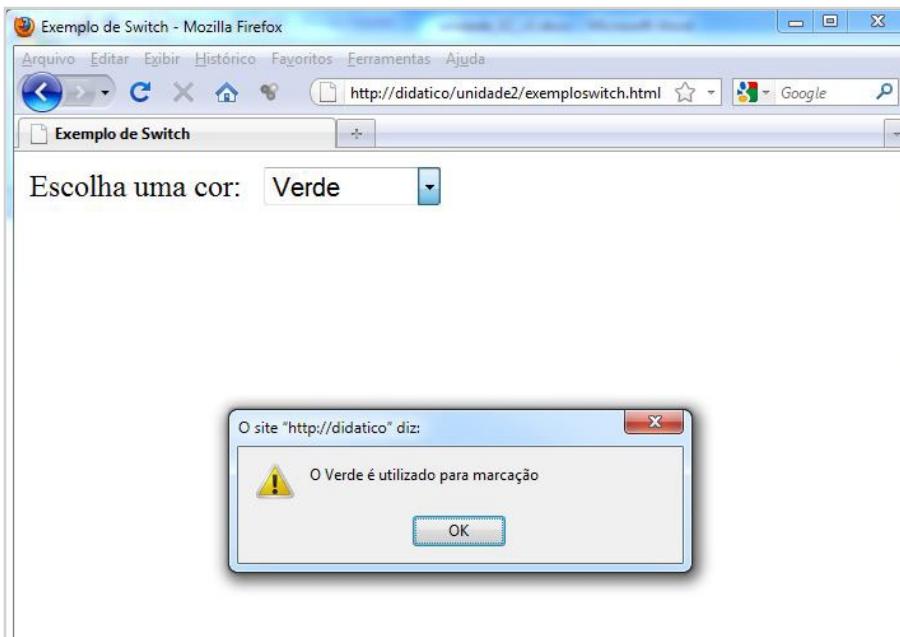


Figura 2.12 – Exemplo do controlador *switch*.

Fonte: Elaborada pelo autor ( 2010).

#### d. Laços de repetição

Os laços de repetição caracterizam-se por repetir um bloco de instrução um número finito de vezes. O bloco utiliza o delimitador de função “{}” para definir o escopo do trabalho. Caso este seja omitido, a primeira linha seguinte à instrução será a única dentro do laço.

Um dos laços mais utilizados é o laço *for*. Esse laço caracteriza-se por utilizar um contador numérico para limitar o número de iterações. Ainda possui um conjunto de três expressões contidas entre parêntesis, separadas pelo caractere ponto e vírgula e pelo código a executar em cada um dos laços.

A primeira das expressões do laço *for* declara a variável para utilizar como índice inicializando-a. A segunda expressão declara uma condição que deve ser testada sempre que se inicia uma nova iteração. Se essa condição resultar em *false*, o laço irá parar e o código definido abaixo não voltará a ser executado. A terceira expressão serve para atualizar o valor do índice no final de cada ciclo.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo laço for</title>
</head>
<body>
<script type="text/javascript" language="javascript">
for (var i=0; i<20; i++) {
    document.write("O do índice é: "+i+"<br>");
}
</script>
</body>
</html>
```

O resultado no *browser* é:

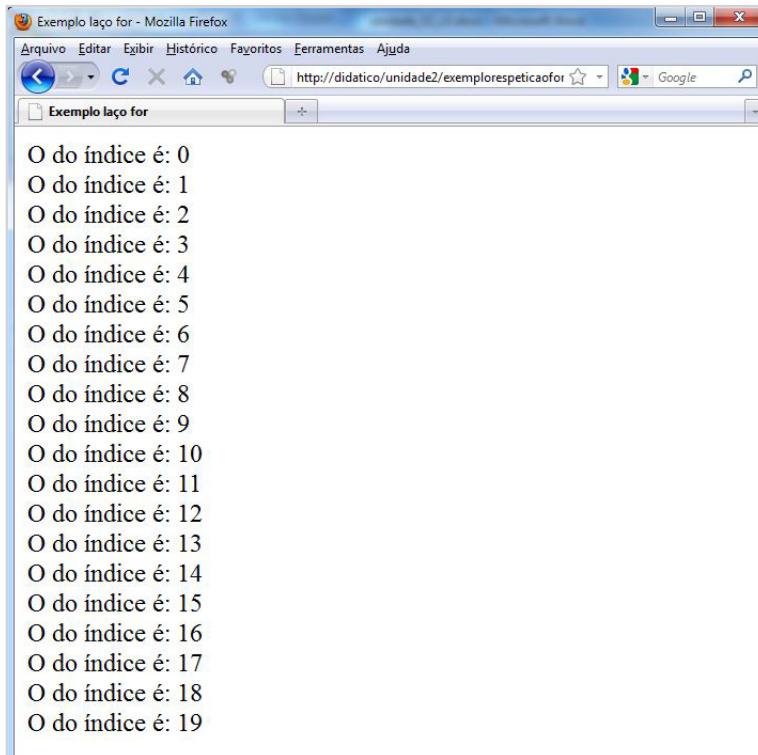


Figura 2.13 - Exemplo do laço FOR.

Fonte: Elaborada pelo autor (2010).

Esse exemplo mostra o laço de repetição *for*, como ele percorre o contador e a condição nele atribuída. Um outro exemplo seria:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>Exemplo de laço de repetição</title>
</head>
<script language="javascript" type="text/javascript">
function validarEntero(valor){

    valor = parseInt(valor);

    if (isNaN(valor)) { return ""; } else { return valor; }

}

```

```
function validarFormulario(campo){  
    textoCampo = campo.value;  
    textoCampo = validarEnterro(textoCampo);  
    campo.value = textoCampo;  
}  
  
function repete(rept, numero, result) {  
  
    var repeticoes= rept.value;  
    var inteiro = parseInt(numero.value);  
    var resultado=0;  
  
    for (var i=1;i<=repeticoes;i++)  
        resultado=resultado+(inteiro*repeticoes);  
  
    result.value=resultado;  
}  
</script>  
<body>  
<form method="post" name="form1">  
    Digite um número inteiro  
    <input type="text" name="inteiro" value="" onkeyup="validarFormular  
io(this)" style="text-align:right;"/>  
    <br />  
    Escolha o número de repetições você deseja  
    <select name="repeticoes" onchange="repete(this,document.form1.  
inteiro,document.form1.resultado)">  
        <option value="" selected>&ampnbsp</option>  
        <option value="1">UM</option>  
        <option value="2">DOIS</option>
```

```

<option value="3">TRÊS</option>
</select>
<br />
Resultado da operação acumulada de (Repetição * Inteiro)<br>
<input type="text" name="resultado" value="" style="text-align:right;">
readonly="true" />
</form>
</body>
</html>

```

No *browser* fica desta forma:

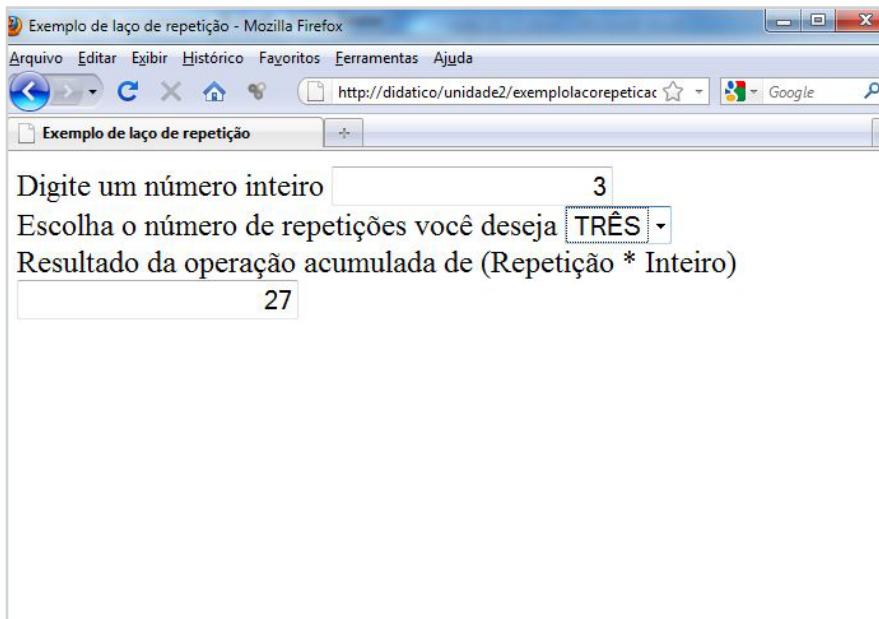


Figura 2.14 - Laço FOR sem bloco.

Fonte: Elaborada pelo autor ( 2010).

Nesse exemplo simples, é possível verificar todos os laços e algumas funções. A função **isNaN** testa um valor a fim de verificar se ele é um número. A função **parseInt** transforma uma expressão em um número possível. Por fim, a função **this** é uma palavra reservada para referenciar o objeto corrente, neste caso, o campo do formulário que está chamando a função.



Outras funções, como o `parseInt`, estão disponíveis para o Javascript. Ainda possui a função `indexOf` sendo muito útil em condições. Pesquise sobre seu funcionamento e teste-a.

Outro laço de repetição importante é o `while`. Esse laço é muito parecido com o laço `for`. Na realidade, tudo o que um faz pode ser feito com o outro, embora por questões de legibilidade cada um deles possa ter áreas de aplicação que para as quais é mais indicado do que o outro.

O laço `while` avalia uma condição e, se ela resultar em `true`, executa o bloco de código que vem imediatamente a seguir. Caso resulte em `false`, o interpretador posiciona na instrução imediatamente após o laço e continua a execução.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo laço while</title>
</head>
<body>
<script type="text/javascript" language="javascript">
var i=0;
while (i<20) {
    document.write("O do índice é: "+i+"<br>");
    i++;
}
</script>
</body>
</html>
```

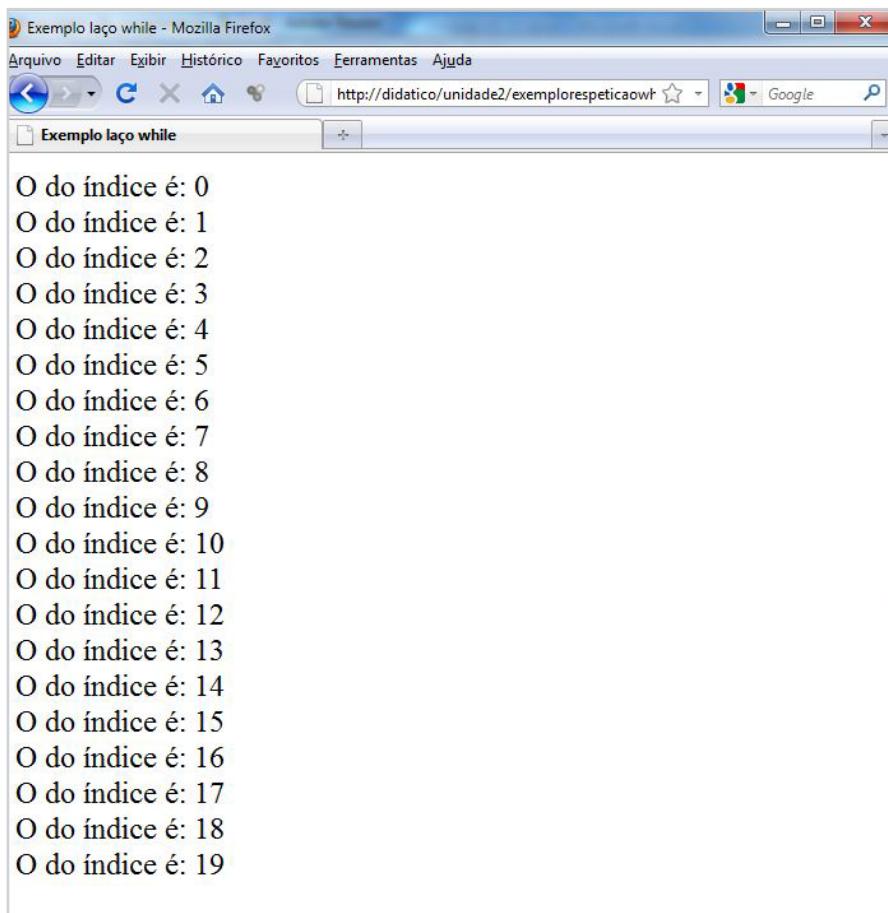


Figura 2.15 – Laço WHILE.

Fonte: Elaborada pelo autor (2010).

O resultado nesse caso é idêntico ao laço *for*. Porém, é nítido que a complexidade envolvida nesse código é superior ao do laço *for*. Mesmo assim, quando você tem a necessidade de utilizar um laço de decisão em que não seja necessário um contador, o laço *while* é a melhor solução.

Um terceiro laço que o JavaScript comporta é o *do..while*. Esse laço é “irmão” do laço *while*, possui uma sutil, mas importante, diferença. Para o laço *while* ser executado uma primeira vez, é necessário que a condição seja atendida ao menos uma vez.

O laço *do..while* executa **sempre** uma vez antes de testar a condição. Isso é possível porque a condição é verificada após o bloco de comando. Diferente de seu “irmão” *while*, que é testado antes do bloco iniciar.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo do..while</title>
</head>
<body>
<script language="javascript" type="text/javascript">
var i = 0;
while (i<8) {
    document.write("Instrução while executada vale: "+i+","
condição atendida<br>");

    i++;
}
i = 0;
do {
    document.write("Instrução do..while executada contador vale: "+i+","
condição atendida<br>");

    i++
} while (i<8)

document.write("Quando a condição não é atendida<br>Os laços são
executados novamente neste exemplo.<br>");

document.write("Porém a variável não é zerada, vejamos como se
comporta<br><br>");

// Condição NÃO é atendida
while (i<8) {
    document.write("Instrução while executada "+i+"ª vez, condição NÃO
atendida<br>");
```

```

    i++;
}

do {
    document.write("Instrução do..while executada vale: "+i+", condição
NÃO atendida<br>");

    i++
} while (i<8)

</script>

</body>

</html>

```

Nesse exemplo, a situação foi forçada para você perceber a diferença. Via de regra, você pode adotar que o *while* deve ser usado sempre que um bloco do laço de repetição necessite de uma condição verdadeira para executar ao menos uma vez.

O *do..while* deve ser construído quando um bloco do laço necessite ser executado ao menos uma vez mesmo que uma condição não seja atendida. No *browser*, o resultado do exemplo fica:

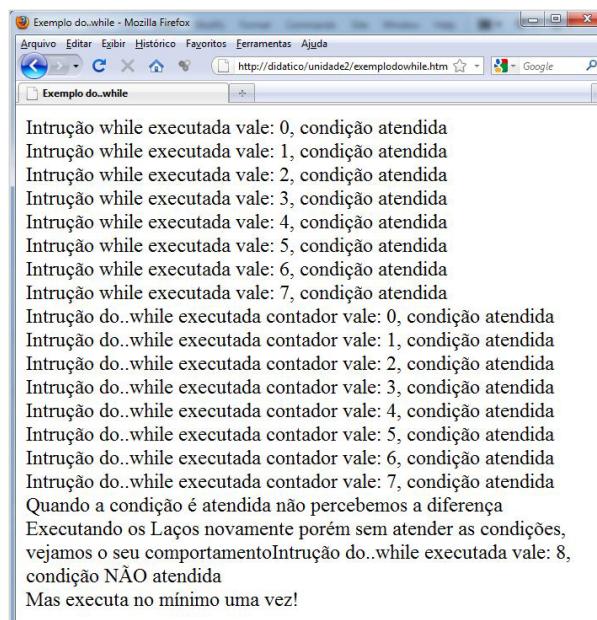


Figura 2.16 - Comparação de estruturas *while* e do *while*.

Fonte: Elaborada pelo autor (2010).

## e. Funções

Uma função é um bloco de código que necessita ser chamado para executar uma determinada operação. Em JavaScript, esse artifício é amplamente utilizado, já que a maior utilização dessa linguagem é a interação com o usuário.

As funções possuem uma declaração de nome e uma delimitação de bloco por meio do caractere {}. O bloco pode conter uma infinidade de instruções para execução de seu código. O retorno de informações é opcional e deve ser feito pela instrução *return*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>Exemplo de Função sem argumento</title>

</head>

<script type="text/javascript" language="javascript">

function testefuncao() {

    return "A função foi chamada";

}

</script>

<body>

<h1>Vejamos um teste de função sem argumento!</h1>

<script type="text/javascript" language="javascript">

document.write(testefuncao());

</script>

</body>

</html>
```

Veja que, além do nome da função, é necessário utilizar a entrada de parâmetros que está entre parênteses. Desta forma, a função é chamada e seu retorno pode ser atribuído a uma variável. O

*browser* interpreta como se a instrução estivesse sendo escrito diretamente na função *write*.

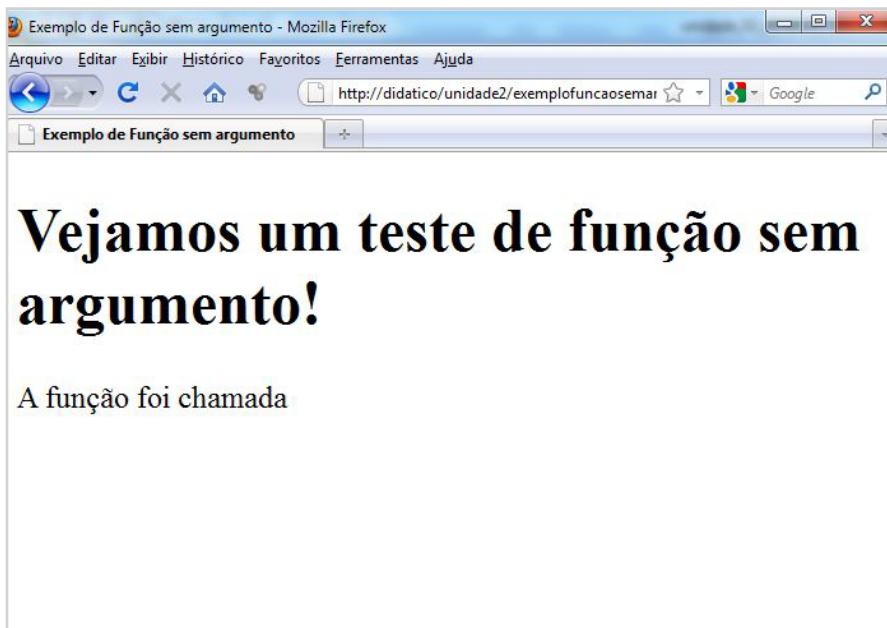


Figura 2.17 - Chamada de função.

Fonte: Elaborada pelo autor (2010).

As funções ainda podem receber argumentos ou parâmetros. Esses argumentos são variáveis que recebem valores passados durante a chamada da função. Isso possibilita o processamento do código utilizando variáveis locais com valores globais. O seu código permanecerá mais limpo e fácil de interpretar, sem questionar a diminuição da incidência de erro ao utilizar variáveis globais dentro das funções.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>Exemplo Função com argumentos fixo</title>

</head>

<script type="text/javascript" language="javascript">
```

```
function autosoma(valor) {  
    var soma=0;  
    do {  
        soma += valor;  
        valor--;  
    } while (valor>0)  
    return soma;  
}  
  
function atribuircampo(campovalor, camporetorno) {  
    camporetorno.value = autosoma(parseInt(campovalor.value));  
}  
</script>  
  
<body>  
  
<h1>Função com argumento fixo</h1>  
  
<form name="form" method="get">  
  
    Digite um número: &ampnbsp  
  
    <input type="text" name="txt_valor" id="txt_valor" value=""  
           align="right" />  
  
    <input type="button" name="btn_calcular" value="Calcular"  
           onclick="atribuircampo(document.form.txt_valor , document.form.  
           txt_resultado)"/><br /><br />  
  
    Resultado:&ampnbsp<input type="text" name="txt_resultado" id="txt_  
           resultado" value="" readonly="readonly" />  
  
</form>  
  
</body>  
</html>
```

Nesse exemplo, está sendo utilizada uma função sem retorno (atribuicampo). Essa função está recebendo dois argumentos, a diferença é que esses argumentos não são valores, mas objetos do formulário. O JavaScript realiza uma referência com o objeto, ou seja, todos os atributos e métodos associados a esse objeto são

propagados para o argumento, assim como qualquer alteração de valores ou estados é refletida diretamente no objeto HTML.

A segunda função é feita uma chamada com argumento. Mas este não é passado um objeto, mas sim um valor. Assim o JavaScript interpreta que o valor foi passado por cópia e qualquer alteração nesse valor não influencia no ser progenitor, nesse caso, o campo “txt\_valor”.

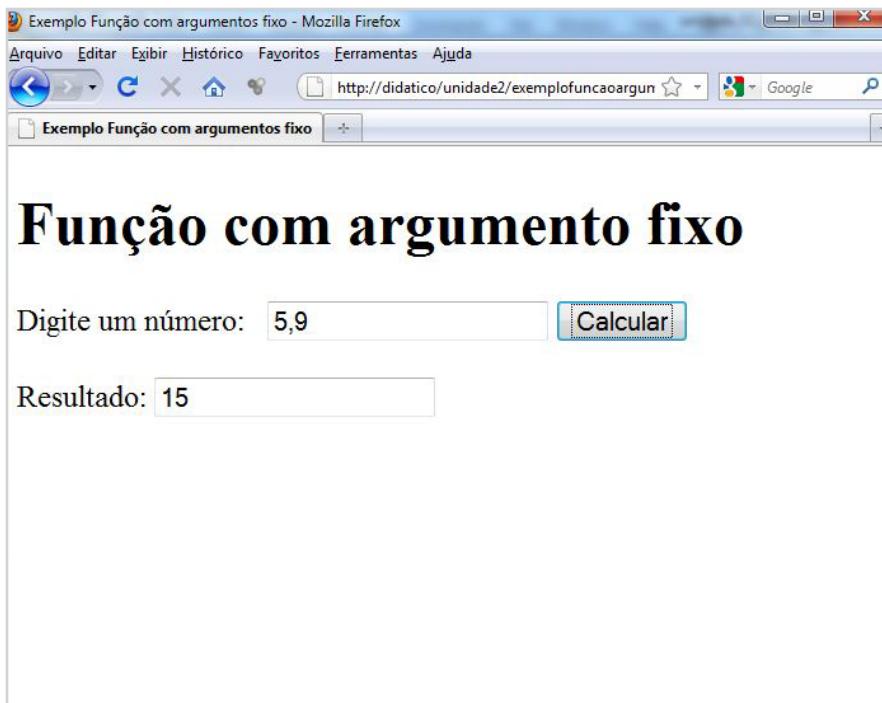


Figura 2.18 - Função de retorno.

Fonte: Elaborada pelo autor (2010).

Os argumentos são extremamente úteis e muitas vezes necessários. Mas imagine que seja necessário que se faça uma função que some qualquer cadeia de números. Ou seja, uma função que aceite receber qualquer número de argumentos e trabalhe com todos de forma dinâmica. O JavaScript resolve esse problema com uma propriedade de todas as funções: o *arguments*. Essa propriedade atribui um *array* de argumentos que estão disponíveis em dentro da função.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo Função argumento variável</title>
</head>

<script type="text/javascript" language="javascript">

function autosoma() {

    var soma=0;

    for (var i=0; i<autosoma.arguments.length; i++) {
        soma += parseInt(autosoma.arguments[i]);
    }

    return soma;
}

</script>

<body>

<script type="text/javascript" language="javascript">

document.write("Chamada da função, somar 1,5,6,7,8 =
"+autosoma(1,5,6,7,8));

document.write("<br>");

document.write("Chamada da função, somar 67, 8, 9 ,7 , 5, 7 ,9 =
"+autosoma(67, 8, 9 ,7 , 5, 7 ,9));

document.write("<br>");

document.write("Chamada da função, somar 7, 9, 5 = "+autosoma(7, 9, 5));

</script>

</body>

</html>
```

Veja que nesse exemplo são passados diversos argumentos diferentes, quanto a valores e a número de argumentos. A

declaração da função não está esperando nenhum argumento definido. Porém, por meio do atributo *arguments*, podemos acessar todos os argumentos transmitidos.

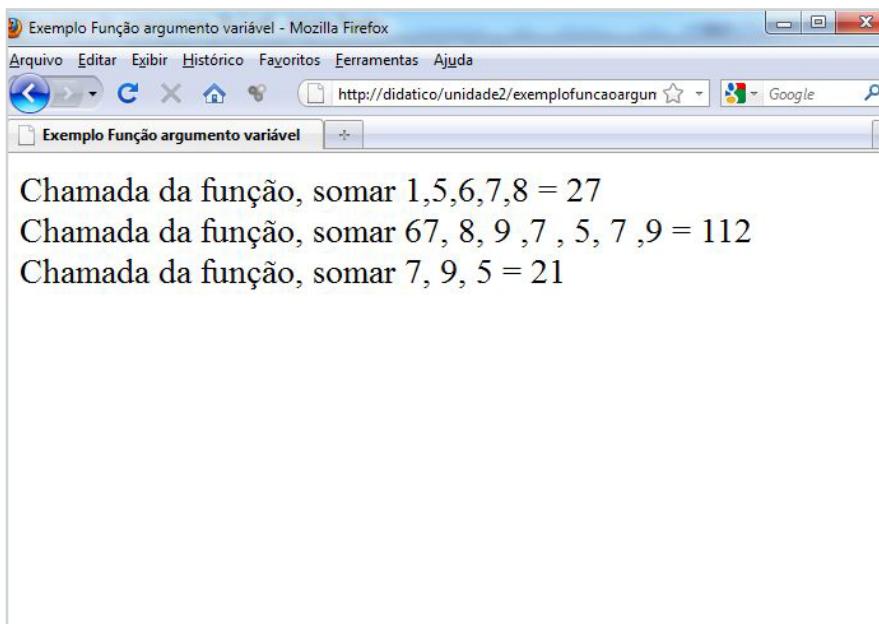


Figura 2.19 - Função com argumentos genéricos.

Fonte: Elaborada pelo autor (2010)

## e. Eventos

Como em outras linguagens, o Javascript possui manipulação de eventos. Esses eventos são as ações e as reações que executam um determinado procedimento. Os eventos são quaisquer ações iniciadas pelo usuário (clicar em determinada área da página, por exemplo) e normalmente são utilizados para realizar as chamadas das **funções Javascript**.

A utilização do evento é feita como um atributo de uma *tag* HTML. A sintaxe é feita da seguinte forma:

```
<tag Evento="chamada javascript">
```

A tabela a seguir representa os eventos mais comumente utilizados:

Tabela 1 – Eventos e manipuladores.

EVENTO	MANIPULADOR	DESCRIÇÃO
Blur	onBlur	Ocorre quando o usuário retira o foco de um objeto de formulário.
Change	onChange	Ocorre quando o usuário muda o valor de um objeto de formulário.
Click	onClick	Ocorre quando o usuário clica sobre o objeto.
Focus	onFocus	Ocorre quando o usuário focaliza o objeto.
Load	onLoad	Ocorre quando o usuário carrega a página.
Unload	onUnload	Ocorre quando o usuário abandona a página.
MouseOver	onMouseOver	Ocorre quando o ponteiro do <i>mouse</i> passa sobre um <i>link</i> ou uma âncora. Válidos apenas para <i>hiperlinks</i> .
Select	onSelect	Ocorre quando o usuário seleciona um elemento de um formulário.
Submit	onSubmit	Ocorre quando o usuário envia um formulário.
MouseDown	onMouseDown	Ocorre quando o botão do <i>mouse</i> é pressionado.
MouseMove	onMouseMove	Ocorre quando o ponteiro do <i>mouse</i> se movimenta sobre o objeto.
MouseOut	onMouseOut	Ocorre quando o ponteiro do <i>mouse</i> afasta de um objeto. Válidos apenas para <i>hiperlinks</i> .
MouseUp	onMouseUp	Ocorre quando o botão do <i>mouse</i> é solto.
KeyDown	onKeyDown	Ocorre quando uma tecla é segurada.
KeyPress	onKeyPress	Ocorre quando uma tecla é pressionada.
KeyUp	onKeyUp	Ocorre quando uma tecla é solta.

Fonte: Javascript – aplicações interativas para a Web (2006).

Existem eventos que são específicos para cada *tag*. A utilização de um evento em uma *tag* que não o suporta acarretará em não

ativação do *script*. Veja então a utilização de um evento dentro do código HTML:

```
<html>
<head>
<title>Teste</title>
</head>
<body onload="defaultStatus='Seja bem vindo'" onunload="alert('Até
mais')">
...
</body>
</html>
```



O browser Firefox traz por *default* o bloqueio para alteração na barra de *status*. Para visualizar o exemplo, é necessário habilitá-lo nas opções do *browser*.

Esses dois eventos são característicos da **tag body**. Eles são ativados quando a página é carregada e quando é descarregada do *browser*. Perceba que diretamente não houve uma ação direta do usuário, porém, considera-se uma ação do usuário o acesso e a saída da página.



Existem muitos outros eventos na linguagem. Alguns somente são portados em algumas versões de *browser* e outros somente em um tipo de *browser*. Procure pesquisar a fim de conhecer esses eventos e quais se adaptam melhor às suas necessidades.

## Seção 4 – Scripts JS

Nesta seção, vamos utilizar o JS na prática, realizando algumas validações de campos. Assim, poderemos ver como utilizar o JS na sua principal função.

Você já viu um exemplo para validar um campo somente numérico. Podemos começar então acrescentando uma máscara para um campo de valores, como um campo de fone:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Mascara de fone</title>
<script language="javascript" type="text/javascript">
function mascarafone(objeto) {
    var aux;
    aux = '(' + objeto.value.substr(0,2) + ")";
    aux += objeto.value.substr(2,4)+"-"+objeto.value.substr(6,4);
    objeto.value = aux;
}
</script>
</head>
<body>
<form name="form1">
<input type="text" onblur="mascarafone(this);">
</form>
</body>
</html>
```

Nesse exemplo, podemos ver que o evento associado à chamada da função é o **Blur**. Portanto, a máscara somente será executada quando o campo perder o foco, ou seja, quando o cursor sair do campo será chamada a função.

Outra função usada foi a ***substr***. Essa função pega um pedaço da literal que inicia na posição do primeiro argumento e o segundo argumento indica o número de caractere será capturado. Vale ainda comentar que essa máscara visa colocar caracteres no formato dos fones.

Vamos agora criar um *script* simples de validação de campos.

Neste exemplo, não vamos nos preocupar com o tipo de dados apenas verificar se os itens obrigatórios estão preenchidos. Antes de iniciar, vale lembrar que os atributos das *tags* HTML não possuem validação, ou seja, posso criar qualquer atributo que eu venha a desejar. Então, vamos ao *script*:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>Validação dos campos</title>
<script language="javascript" type="text/javascript">
function valida(objeto) {
    //a proriedade elements permite acessar todos os objetos existentes
    //em forma de vetor. Assim, podemos colocar todos os objeto dentro de
    //um laço acessando cada um sem saber qual o nome.

    var elementos = objeto.elements;
    var resp="";
    for (var i=0; i<elementos.length; i++){
        //neste exemplo, estamos validando apenas os campos do tipo text e
        //que tem no atributo que foi criado "obrigatorio" igual a S, assim com
        //uma única função podemos acrescentar vários campos sem necessitar
        //alterar a função
        if ((elementos[i].getAttribute("type")=="text")&&
            (elementos[i].getAttribute("obrigatorio")=="S") &&
```

```
(elementos[i].value=="")) {  
    //style é o objeto referente aos estilos CSS, neste caso  
    estamos trocando a cor de fundo  
    elementos[i].style.backgroundColor = "#ffffcc";  
    resp+="O campo "+elementos[i].getAttribute("descr") +  
    " é obrigatorio.\n";  
}  
}  
if (resp!="") {  
    alert(resp);  
    return false;  
}  
return true;  
}  
</script>  
</head>  
<body>  
<!-- usar o return na chamada deste evento significa que se retornar false<br/>o browser entende que houve algum problema e não continuará. Se for  
true, não há problema algum em continuar.-->  
<form id="form1" name="form1" method="post" action="www.google.  
com.br" onsubmit="return valida(this);">  
    <p>Nome:  
        <label>  
            <input type="text" name="nome" id="nome" obrigatorio="S"  
            descr="O nome deve ser preenchido" />  
        </label>  
    </p>  
    <p>Apelido:  
        <label>  
            <input type="text" name="apelido" id="apelido" />  
        </label>
```

```
</p>
<p> Ano de nascimento
    <input name="ano" type="text" id="ano" maxlength="4"
obrigatorio="S" descr="Ano deve ter especificamente 4 caracteres" />
</p>
<p>
    <label>
        <input type="submit" name="button" id="button" value="Enviar" />
    </label>
</p>
</form>
</body>
</html>
```

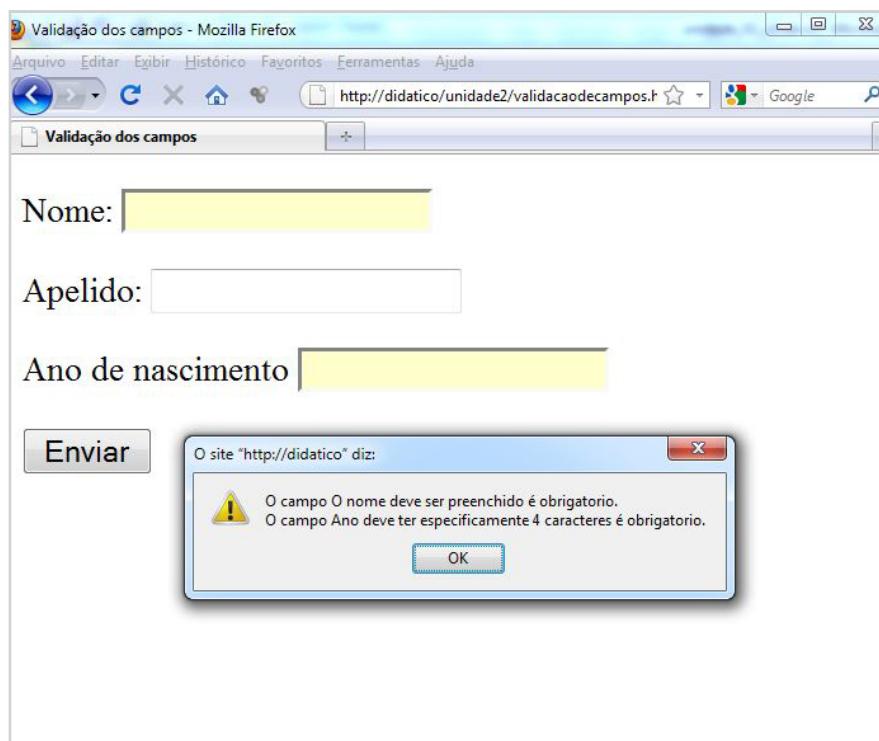


Figura 2.20- Validação de campo.  
Fonte: Elaborada pelo autor (2010).

Vamos para algo mais complexo: vamos verificar um *script* montado por terceiros, já que será o mais comum a se encontrar na internet. Vamos tentar avalizá-lo e entendê-lo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>Mascara de moedas</title>

</head>

<script language="javascript">

//-----
//Funcao: MascaraMoeda
//Sinopse: Mascara de preenchimento de moeda
//Parametro:
// objTextBox : Objeto (TextBox)
// SeparadorMilesimo : Caracter separador de milésimos
// SeparadorDecimal : Caracter separador de decimais
// e : Evento
//Retorno: Booleano
//Autor: Gabriel Fróes - www.codigofonte.com.br
//Comentários: Edson Lessa
//-----

function MascaraMoeda(objTextBox, SeparadorMilesimo,
SeparadorDecimal, e){

    var sep = 0;
    var key = ",";
    var i = j = 0;
    var len = len2 = 0;
    // Caracteres válidos durante a digitação
```

```

var strCheck = '0123456789';

var aux = aux2 = "";

// o comando ? possui a mesma função do if, basicamente pode se
trocar esta linha por

// if ( (window.Event)==true) { var whichCode=e.which; } else { var
whichCode=e.keyCode; }

var whichCode = (window.Event) ? e.which : e.keyCode;

if (whichCode == 13) return true;

// Valor para do número que foi digitado

key = String.fromCharCode(whichCode);

//verifica se o caracter está presente nos caracteres válidos

if (strCheck.indexOf(key) == -1) return false; // Chave inválida

//length retorna o tamanho de caracteres de uma expressão

len = objTextBox.value.length;

for(i = 0; i < len; i++) {

    //charAt é o método que retornar um caractere da posição
    indicada. Lembrando que o número da posição é contado da esquerda
    para direita e inicia com 0.

    //esta parte do código é responsável pela separação do inteiro e
    do decimal

    if ((objTextBox.value.charAt(i) != '0') && (objTextBox.value.charAt(i) !=
SeparadorDecimal)) break;

}

aux = "";

for(; i < len; i++) {

    //armazena na variável aux os números validos já digitados

    if (strCheck.indexOf(objTextBox.value.charAt(i))!=-1) aux +=
objTextBox.value.charAt(i);

}

//adiciona a última tecla digitada

aux += key;

//total de números digitados

```

```
len = aux.length;

    //este bloco adiciona zeros para os primeiros números decimais
    digitados

    if (len == 0) objTextBox.value = "";

    if (len == 1) objTextBox.value = '0'+ SeparadorDecimal + '0' + aux;

    if (len == 2) objTextBox.value = '0'+ SeparadorDecimal + aux;

        //acrescenta do separador de milhar

    if (len > 2) {

        aux2 = "";

        for (j = 0, i = len - 3; i >= 0; i--) {

            if (j == 3) {

                aux2 += SeparadorMilesimo;

                j = 0;

            }

            aux2 += aux.charAt(i);

            j++;

        }

        objTextBox.value = "";

        len2 = aux2.length;

        for (i = len2 - 1; i >= 0; i--)

            objTextBox.value += aux2.charAt(i);

            //acrescenta símbolo do decimal nos dois últimos caracteres

            objTextBox.value += SeparadorDecimal + aux.substr(len - 2, len);

        }

        return false;

    }

</script>

<body>

Exemplo de utilização:

<form>
```

Valor R\$:

```
<input type="text" name="valor" onKeyPress="return(MascaraMoeda(this,'.,.,event))" style="text-align:right;">  
</form>  
</body>  
</html>
```

Perceba que nesse *script* estamos trabalhando diretamente com a captura das informações digitadas pelo usuário por meio do evento *onKeyPress* e da função *event*. Perceba também que estamos quebrando a cadeia de caracteres por meio do método *charAt* para que seja possível trabalhá-la individualmente facilitando o processamento.



Existe mais de uma forma de se fazer uma casa. Procure outras formas de validação de campos e máscaras. Perceba que, por mais diferentes elas podem parecer, em sua essência são todas iguais.

O JavaScript é uma poderosa ferramenta a nosso dispor. Aprimorar e alcançar seus limites depende muito mais da criatividade do desenvolvedor. A internet está repleta de códigos prontos disponibilizados por desenvolvedores para que não seja necessário reinventar a roda. Procure sempre explorar novos horizontes.



## Síntese

O JavaScript é uma linguagem orientada a objetos, porém possui recursos limitados que são predefinidos pela própria linguagem. O JavaScript procura transformar todo o conteúdo HTML em objeto, para assim simplificar a sua manipulação. Toda a estrutura de variáveis segue essa linha de objetos, porém a construção da aplicação com essa linguagem é feita utilizando a forma estruturada. Esse método simplifica a construção e o entendimento. As variáveis podem ser do tipo *string* (caractere), *date* (data), *array* (cadeia de variáveis), *int* (inteiro) ou *boolean* (lógico). Cada qual herda os métodos e a propriedade respectiva de seus objetos. As funções são amplamente utilizadas por causa de a linguagem ser utilizada para interagir com o usuário. Os eventos existentes proporcionam um leque razoável de opções para interagir com as ações dos usuários.



## Atividades de autoavaliação

Ao final de cada unidade, você será desafiado a realizar atividades de autoavaliação. O gabarito está disponível no final do livro-didático. Esforce-se para resolver as atividades sem ajuda do gabarito, pois assim você estará estimulando a sua aprendizagem.

1. Faça um *script* para validar um formulário que contenha nome, endereço, nome da mãe, idade e *e-mail*. Os campos obrigatórios são nome, nome da mãe, idade e *e-mail*. Valide a idade para aceitar somente inteiros positivos e entre 10 e 120 anos. O *e-mail* deve ser validado de tal forma que deve conter o caractere @ e pelo menos um ponto após o @. Publique sua resposta na ferramenta Exposição do EVA.
2. Crie um formulário que contenha um elemento *textarea*. Você deve utilizar esse elemento para inserir informações em uma *tag div* por meio do JavaScript. Publique o *script* na ferramenta exposição do EVA.



## Saiba mais

Se você desejar, aprofunde os conteúdos estudados nesta unidade ao consultar as seguintes referências:

Código Fonte, 2010. Disponível em <<http://codigofonte.uol.com.br/artigos/javascript-dhtml>>. Acesso em: 01 mar. 2010.

Truquese Dicas: Tutoriais de Javascript, 2010. Disponível em <[www.truquesedicas.com/tutoriais/javascript](http://www.truquesedicas.com/tutoriais/javascript)>. Acesso em: 2 mar. 2010.

[jsfromhell.com](http://jsfromhell.com) JavaScript Repository, 2010. Disponível em: <<http://jsfromhell.com/>>. Acesso em: 15 mar. 2010.



# UNIDADE 3

## PHP

# 3



### Objetivos de aprendizagem

Ao final do estudo desta unidade, é importante que você:

- Saiba utilizar conteúdo dinâmico.
- Conheça variáveis, constantes e escopo de ação.
- Compreenda uso de operadores e expressões.
- Identifique e desenvolva os laços de decisão e repetição.
- Desenvolva aplicações Serve-side.



### Seções de estudo

**Seção 1** Visão geral

**Seção 2** Variáveis e constantes

**Seção 3** Operadores

**Seção 4** Estruturas de controle

**Seção 5** Trabalhando com arquivos



## Para início de estudo

Você já observou que o desenvolvimento de uma aplicação para Web possui inúmeras vantagens. Verificou a aplicabilidade dessa plataforma para o desenvolvimento de aplicações de pequeno e grande porte. Estudou sobre o HTML e o JavaScript, que são linguagens *client-side*. Agora entraremos em um universo um pouco diferente, uma linguagem *serve-side*.

Uma linguagem *serve-side* consiste em trabalhar diretamente no servidor de aplicação. Esse tipo de linguagem proporciona recursos de processamentos centralizados. Um maior controle sobre o ambiente o qual está sendo trabalhado. As linguagens *server-side* são indispensáveis para se construir aplicações que necessitam de armazenamento de dados. A linguagem PHP se encaixa perfeitamente a essa necessidade.

O PHP é uma linguagem muito utilizada e está presente em mais 20 milhões de domínios. Essa linguagem é Open Source, ou seja, você pode acessar o código-fonte e alterá-lo.

O PHP foi desenvolvido em 1994 pelo desenvolvedor Rasmus Lerdorf. Mais tarde foi regravado por diversas equipes e alcançou a maturidade existente hoje. PHP teve seu significado original como sendo *Personal Home Page*, mas hoje foi alterado conforme a convenção para atribuição de nomes recursiva GNU, adotou-se o nome de PHP *Hypertext Preprocessor*. Sua Sintaxe lembra um pouco a sintaxe do C e do Perl, e é uma linguagem bem fácil de aprender.

## Seção 1 – Visão geral

O PHP é multiplataforma e pode ser usado na maioria dos sistemas operacionais. Diferente de *scripts* como o JavaScript, ele roda no servidor, que é suportado pela maioria dos servidores Web que existem hoje no mercado, como Apache, IIS etc.

O cliente recebe apenas os resultado dos *scripts*, que são interpretados no servidor, não tendo acesso ao código. Muito simples de se aprender e trabalhar, o PHP atende desde os mais experientes desenvolvedores até os iniciantes na área.



---

Existem diversos tutoriais para instalação do PHP. É aconselhável utilizar uma suíte pronta de servidor como Xampp <<http://www.apachefriends.org/en/xampp.html>>, ApacheTriad <<http://www.apachefriends.org/en/xampp.html>>, PHPTriad <<http://sourceforge.net/projects/phptriad>> ou outra suíte semelhante em que se tenha o servidor Web + PHP + Mysql.

---

Uma das grandes vantagens do PHP é sua **portabilidade**, ou seja, sua capacidade de ser executada em diversos sistemas operacionais. O PHP está presente nos principais sistemas operacionais existentes no mercado. Uma de suas características é ser *server-side*, isso significa que é executada somente no servidor e, por isso, facilita a sua manutenção e segurança em relação à ocultação de linguagem.

Visando melhorar o entendimento, vamos utilizar uma página baseada em um sistema de requisição. Esse sistema é voltado para uma empresa comercial de informática e tem como base requisitar produtos para o estoque situado em outra localidade. O nome da empresa é **InfoComponente**.

Imaginemos então que a página HTML já esteja pronta e não requer nenhum recurso de usabilidade ou mesmo visual. Assim, temos o seguinte resultado:

Item	Quantidade
Mouse	
Teclado	
Fone	

**Enviar** **Limpar**

Figura 3.1 – O formulário de requisição em que se registra apenas a quantidade de produtos solicitados.

Fonte: Elaborada pelo autor (2010).

Perceba que há duas informações de extrema importância no código da requisição, como mostrado a seguir:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Formulário</title>
</head>
<body>
<form id="frmrequisicao" name="frmrequisicao" method="post"
action="processarrequisicao.php">
<table width="30%" border="1">
<tr bgcolor="#999999">
<th width="40%" scope="col">Item</th>
<th width="60%" scope="col">Quantidade</th>
</tr>
<tr>
<td><label for="txtmouse">Mouse</label></td>
```

```
<td>

    <input name="txtmouse" type="text" id="txtmouse" value=""
maxlength="3"/>

</td>
</tr>

<tr>

    <td><label for="txtteclado"> Teclado</label></td>

    <td><input name="txtteclado" type="text" id="txtteclado" value=""
maxlength="3" /></td>

</tr>
<tr>

    <td><label for="txtfone"> Fone </label></td>

    <td><input name="txtfone" type="text" id="txtfone" value=""
maxlength="3" /></td>

</tr>
</table> <p>

    <input type="submit" name="sbtenviar" id="sbtenviar"
value="Enviar" /> &nbsp;

    <input type="reset" name="sbtlimpar" id="sbtlimpar"
value="Limpar" />

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

A primeira *tag* do corpo do HTML é a *form* e contém o atributo *action*. Esse atributo define qual o *script* lerá as informações enviadas pelo formulário. Vale lembrar neste momento que existem diferenças quando o método de envio é GET ou POST, veremos isso mais adiante.

A segunda informação importante são os nomes dos campos do formulário, pois eles serão utilizados para realizar a leitura das informações.

Assim, podemos escrever o programa ou *script* que será o receptor das informações enviadas pelo formulário. Vamos supor que tenhamos colocado as informações no formulário para *mouse*, teclado e fone, sendo 3, 2 e 1, respectivamente, teríamos o seguinte resultado:

### Requisições:

Itens	Quantidade
Mouse	3
Teclado	2
Fone	1

[Voltar](#)

Figura 3.2 – Representando a resposta do envio da requisição.

Fonte: Elaborada pelo autor (2010).

O código que teríamos seria:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Relação Cadastrada</title>
</head>
<body>
<p><h1>Requisições:</h1></p>
<?php
if ((@$_POST["txtmouse"]>0)||(@$_POST["txtteclado"]>0)||(@$_POST["txtfone"]>0)) {
    ?>
    <table width="40%" border="1">
        <tr>
            <th scope="col">Itens</th>
```

```
<th scope="col">Quantidade</th>
</tr>
<tr>
    <th scope="row">Mouse</th>
    <td><?php echo @$_POST["txtmouse"];?></td>
</tr>
<tr>
    <th scope="row">Teclado</th>
    <td><?=@$_POST["txtteclado"]?></td>
</tr>
<tr>
    <th scope="row">Fone</th>
    <td><?=@$_POST["txtfone"]?></td>
</tr>
</table>
<?php
} else {
?>
<h1> ERRO INFORMAÇÕES NÃO VALIDADAS!</h1>
<?php
}
?>
<a href="javascript:window.history.back(-1);">Voltar</a>
</body>
</html>
```

Vamos atentar neste momento para três elementos do PHP que podemos destacar. Primeiro é a variável predefinida `$_POST`. Essa variável é um vetor contendo todos os elementos de um formulário enviado pelo método `post`. Nesse vetor podemos regatar os botões que estão contidos no formulário. Caso tenhamos utilizado o método `get` no formulário, o vetor do PHP também se altera passando para `$_GET`.

O segundo elemento é o `@` antecedendo a variável `$_POST`. O `@` no PHP é na realidade uma função de inibição de erro. Ou seja, qualquer erro com a variável que seja antecedida por `@` terá como valor padrão nulo.



---

Muito cuidado com o uso do `@`, ele pode esconder um erro importante do seu código.

---

O terceiro elemento é o uso das *tags* do PHP. A *tag* PHP dentro do HTML normalmente se apresenta dentro das *tags* `<?php ?>`. Existem ainda as *tags* de atalho como `<?= ?>`, que substituem a função `echo` (escrever).



---

É possível utilizar as mesmas tags do ASP para escrever o PHP. Pesquise qual a configuração deve ser alterada para que isso ocorra.

Normalmente o uso da tag `<?php ?>` juntamente com a palavra `php` é obrigatório, porém é possível alterar esta configuração de tal modo que aceite também sem a palavra `php <? ?>`. Pesquise qual configuração dever ser feita para que isso ocorra.

---

Embora nesse ponto já seja possível iniciar uma programação simples em PHP, veja mais um elemento importante em programação: o **comentário**. O comentário é muito útil como documentação de baixo nível. No PHP ela pode ser de três formas: **bloco**, **linha** e **estilo Shell**.

O comentário em **bloco** é feito utilizando `/*`, e tudo que estiver após esse código estará comentado e devendo ser encerrado com `*/`.

```
/*
Autor: José da Silva
Modificado para melhorar o desempenho em 01 de março
*/
```

A segunda forma de comentário é o comentário de **linha**, no qual deve ser iniciado com // e terminar ao final da linha. Portanto, não é necessário fechar o bloco, basta trocar a linha para terminar o comentário.

```
<td><?php echo @$_POST["txtmouse"];?> // Escreve a quantidade
requisitada se existir
</td>
```

A terceira forma de comentário é o estilo em **Shell**, ou seja, esse padrão é adotado pelos *scripts* Shell e também pelo PHP. Basicamente tem a mesma sintaxe do comentário de linha, porém para abrir o comentário deve-se utilizar #.

```
<td><?php echo @$_POST["txtmouse"];?> #Escreve a quantidade
requisitada se existir
</td>
```

## Seção 2 – Variáveis e constantes

As variáveis e as constantes possuem papel fundamental, pois com elas podemos manipular as informações que são recebidas. As variáveis no PHP são representadas pelo símbolo do cifrão (\$) seguido pelo nome da variável.

```
<?php
$var = 'Bob';
$Var = 'Joe';
```

```
echo "$var, $Var"; // exibe "Bob, Joe"

$4site = 'not yet'; // inválido; começa com um número
$_4site = 'not yet'; // válido; começa com um sublinhado
$stäyte = 'mansikka'; // válido; 'ä' é um caractere ASCII (extended) 228
?>
```



O exemplo anterior pode ser encontrado no manual do PHP <[http://www.php.net/manual/pt\\_BR/language.variables.basics.php](http://www.php.net/manual/pt_BR/language.variables.basics.php)>, assim como outros exemplos. Veja as funções para manipulação das variáveis que consta no manual do PHP: <[http://www.php.net/manual/pt\\_BR/ref.var.php](http://www.php.net/manual/pt_BR/ref.var.php)>.

## a. Tipo de dados

As variáveis são classificadas pelos tipos de dados que elas podem armazenar. Os tipos mais comuns são:

- **Integer** (inteiro): especifica o tipo de armazenamento de números inteiros.
- **Float**: especifica o tipo de armazenamento de ponto flutuante (decimal, notação científica).
- **String**: especifica o tipo de variável sendo caractere.
- **Array**: especifica o tipo de variável sendo um vetor.
- **Object**: especifica o tipo de variável sendo uma instância de classes.

Uma grande vantagem é que você não necessita definir as variáveis pelo tipo no PHP. O interpretador identifica a dado que está sendo atribuída à variável e define o tipo de dado da variável, assim é possível utilizar suas funções predefinidas específicas para aquele tipo de informação.



Lembre-se que o **tipo** é definido pela última atribuição feita à variável. Portanto, se inicialmente você atribuiu o número 4 para a variável \$a e posteriormente atribuiu uma *string* como "quatro", não causará erro por essa atribuição.

Podemos considerar que os *Arrays* no PHP são vetores indexados. Em outras palavras, um valor do tipo *array* é uma forma de arquivamento em memória em que os índices são as chaves de acesso. Lembre-se que os índices podem assumir valores de qualquer tipo e não somente inteiros. Outra característica é que os índices não necessitam obedecer a uma sequência contínua. No caso dos valores inteiros para os índices, pode haver intervalos sem atribuição de valores.

A verificação de tipos em PHP é dinâmica, valores de tipos diferentes podem ser usados como índices de *array*, assim como os valores mapeados podem ser de diversos tipos.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>Exemplo de Arrays</title>

</head>

<body>

<?

$numeros_primos = array(2, 3, 5,7);

$nomes = array(3);

$nomes[0] = "João";

$nomes[1] = "Maria";

$nomes[2] = "Joaquim";

for ($i=0; $i<count($numeros_primos); $i++)
```

```

echo "Números Primos, posição ".($i+1)." :".$numeros_
primos[$i]."<br>";

?>

<br />

<?

for ($i=0; $i<count($nomes); $i++)

print("Nomes, posição ".($i+1)." :".$nomes[$i]."<br>");

$frutas = array("vermelha"=> array(1=>"maçã", 2=>"morango"),
"amarela"=>"banana", "laranga"=>array("laranja", "mamão"));

$i=0;

foreach ($frutas as $cor => $tipos) {

echo "Índice: $cor; Valor: $tipos<br />\n";

echo "As frutas da cor ".$cor." são:<br>";

echo "<ul>";

if (is_array($tipos))

foreach ($tipos as $indice => $valor)

echo "<li>".$valor."</li>\n";

else

echo "<li>".$tipos."</li>\n";

echo "</ul>";

}

$posicao[1]="primiero valor";

$posicao[3]="segundo valor";

foreach ($posicao as $indice => $valor)

echo "Índice: $indice; Valor: $valor<br />\n";

$comida= array(0=>"Feijoada", "dois"=> "Churrasco", 3=>"Arroz",
"quatro"=>"macarrão", 10=>"Ensopado");

while (list($indice, $valor) = each($comida))

```

```

echo "Valor do Índice é: '$índice' e o conteúdo da posição é:  

$valor<br />";  

?>  

</body>  

</html>

```

Veja que no exemplo estão sendo utilizadas várias formas de atribuir um *array* e de como acessar suas informações. O PHP aceita vetor binário, ou seja, vetor dentro de outro vetor. O comando *count* verifica quantas posições existem no vetor.

A instrução *foreach* realiza um laço para percorrer todos os índices e valores do vetor. Essa instrução somente pode ser utilizada com vetores, assim como o comando *each* que divide o vetor em índice e valores unitariamente. Normalmente é utilizada com a função *list*, que cria variáveis como se fossem *array*. Assim, o resultado esperado é:

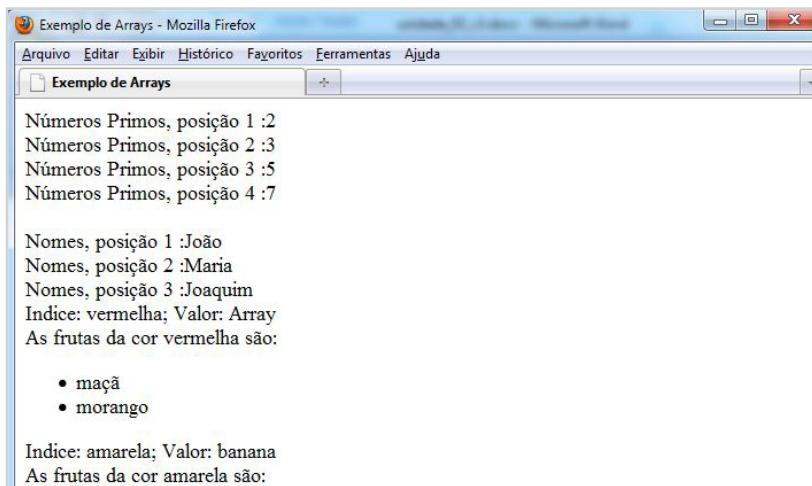


Figura 3.3 - Exemplo de *arrays*.

Fonte: Elaborada pelo autor (2010).



Conheça mais funções do PHP para trabalhar com *arrays* no seu manual: <[http://br3.php.net/manual/pt\\_BR/book.array.php](http://br3.php.net/manual/pt_BR/book.array.php)>

Apesar de o interpretador ser muito eficiente quanto aos tipos de dados, ocorre que algumas vezes é necessário forçar uma conversão dos tipos de dados. Isso pode ocorrer quando são

executadas determinadas operações entre dois valores de tipos diferentes. O interpretador do PHP converte o valor de um deles automaticamente.

Os tipos de dados serão convertidos sempre para aquele que houver menor perda. Isto é, um dos operandos for *float*, o outro será convertido para *float*, senão, se um deles for *integer*, o outro será convertido para *integer*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de tipo de dados</title>
</head>
<body>
<?php
echo $variavel = "1";
echo "<br />";
echo $variavel = $variavel+5;
echo "<br />";
echo $variavel = $variavel+'25';
echo "<br />";
echo $variavel = $variavel+1.999;
?>
</body>
</html>
```

Como podemos notar, o PHP converte *string* para *integer* ou *float* mantendo o valor. Assim, espera-se o resultado:

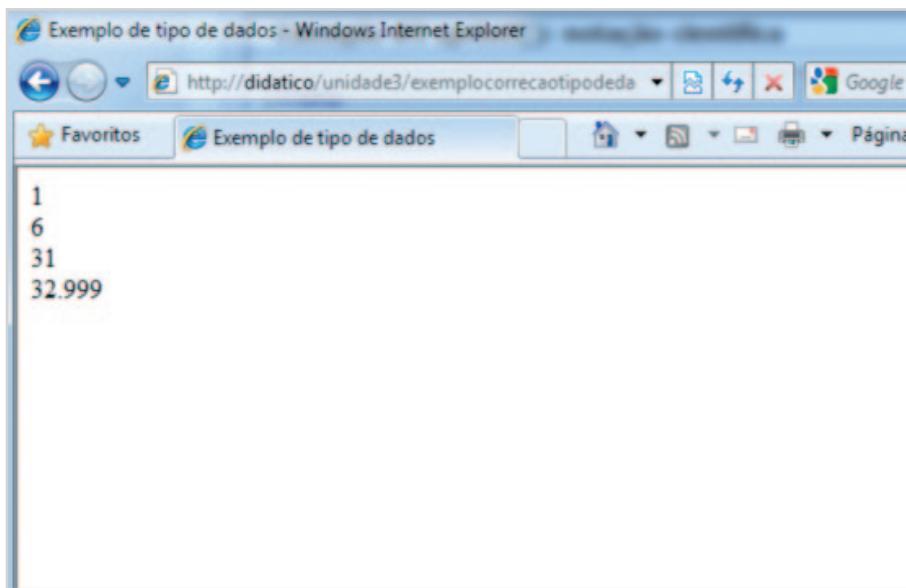


Figura 3.4 - Exemplo de tipo de dados.

Fonte: Elaborada pelo autor (2010).

O sistema utilizado pelo PHP para converter de *strings* para números é o seguinte:

- É analisado o início da *string*. Se contiver um número, ele será avaliado. Senão, o valor será 0 (zero);
- O número pode conter um sinal no início (“+” ou “-”);
- Se a *string* contiver um ponto em sua parte numérica a ser analisada, ele será considerado e o valor obtido será *float*;
- Se a *string* contiver um “e” ou “E” em sua parte numérica a ser analisada, o valor seguinte será considerado como expoente da base 10 e o valor obtido será *float*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de tipo de dados</title>
</head>
<body>
<?php
echo $variavel = "1";
echo "<br />";
echo $variavel = $variavel+5;
echo "<br />";
echo $variavel = $variavel+'25';
echo "<br />";
echo $variavel = $variavel+1.999;
?>
<br />
<br />
<?php
echo $variavel = $variavel+"1.8";
echo "<br />";
echo $variavel = $variavel+"-1.3e3";
echo "<br />";
echo $variavel = $variavel+'variavel3';
echo "<br />";
echo $variavel = $variavel+'222teste dadoas';
echo "<br />";
echo $variavel = $variavel+' 9999x      teste dadoas';
echo "<br />";
```

```
?>
</body>
</html>
```

O *browser* mostrará:

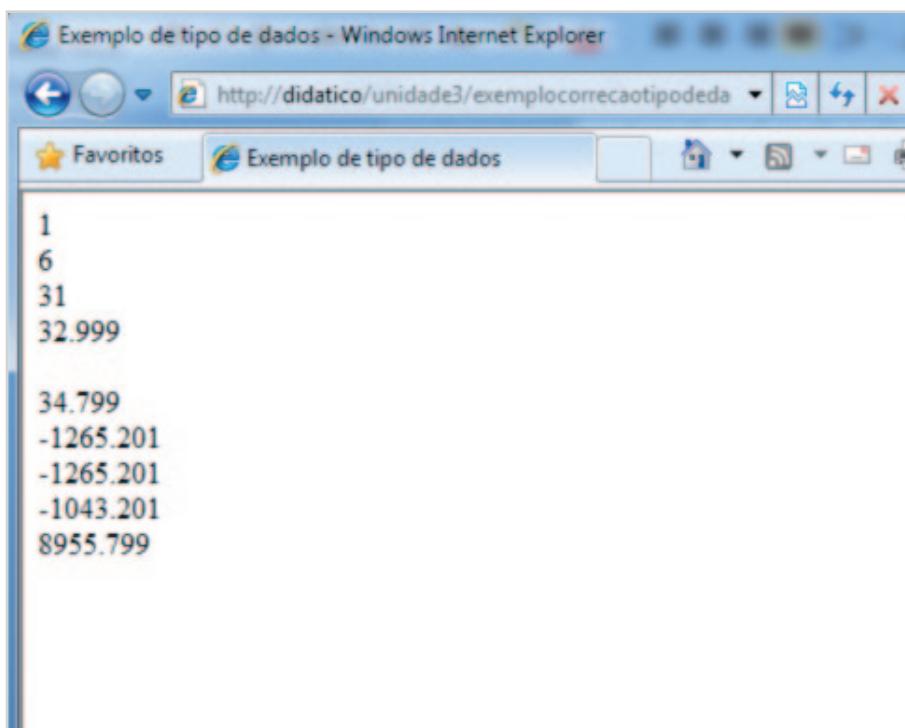


Figura 3.5 - Exemplo de tipo de dados.

Fonte: Elaborada pelo autor (2010).

Outra forma de conversão é você informar para o interpretador qual o tipo de dados está sendo convertido. A sintaxe do *typecast* de PHP: basta escrever o tipo entre parênteses antes do valor.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>Exempl typecast</title>
```

```
</head>
<body>
<?php
$var = 76;
echo $var."<br />";
$var = (float) $var;
echo $var."<br />";
$var = 74.234;
echo $var."<br />";
$var = (int) $var;
echo $var."<br />";
?>
</body>
</html>
```

Os tipos de *cast* permitidos são:

(int), (integer)	é	muda para integer;
(real), (double), (float)	é	muda para float;
(string)	é	muda para string;
(array)	é	muda para array;
(object)	é	muda para objeto.

O resultado do exemplo é:

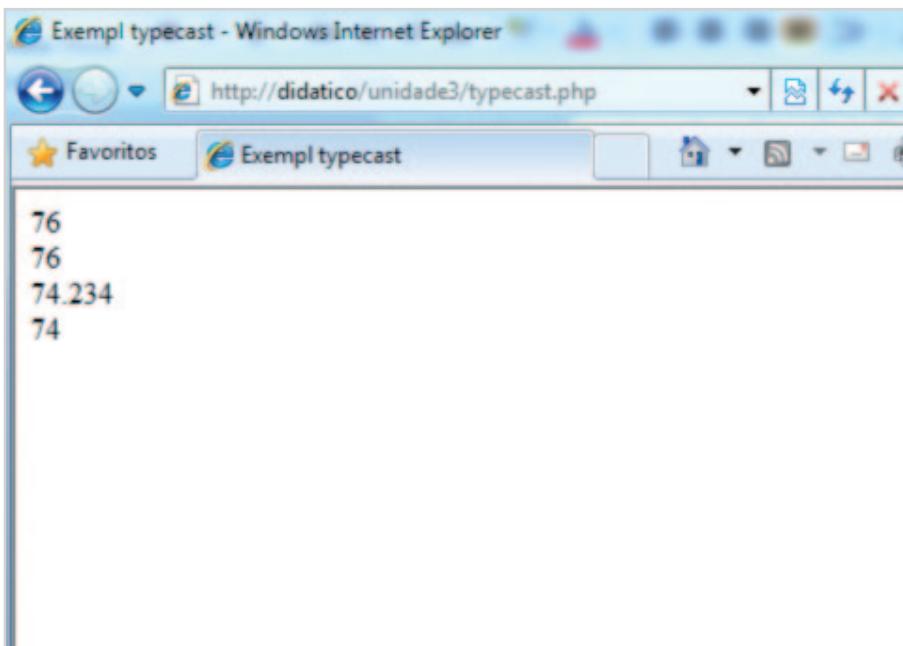


Figura 3.6 - Exemplo de tipo de cast.

Fonte: Elaborada pelo autor (2010).

O PHP ainda permite que sejam criadas variáveis por referência. Isto é, variáveis com nomes diferentes, mas permanecem associadas com seu valor. Quando o conteúdo de uma é alterado, o conteúdo da segunda é alterado automaticamente.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>Variavel por referencia</title>

</head>

<body>

<?php

$nome = "João da Silva";
$identificacao = &$nome;
```

```
echo "Variável nome = $nome<br />";  
echo "Variável identificacao = $identificacao<br /><br />";  
  
$nome = "Maria Firmino";  
  
echo "Variável nome = $nome<br />";  
echo "Variável identificacao = $identificacao<br /><br />";  
  
$identificacao = "José Maria";  
  
echo "Variável nome = $nome<br />";  
echo "Variável identificacao = $identificacao<br />";  
?>  
</body>  
</html>
```

O resultado será:

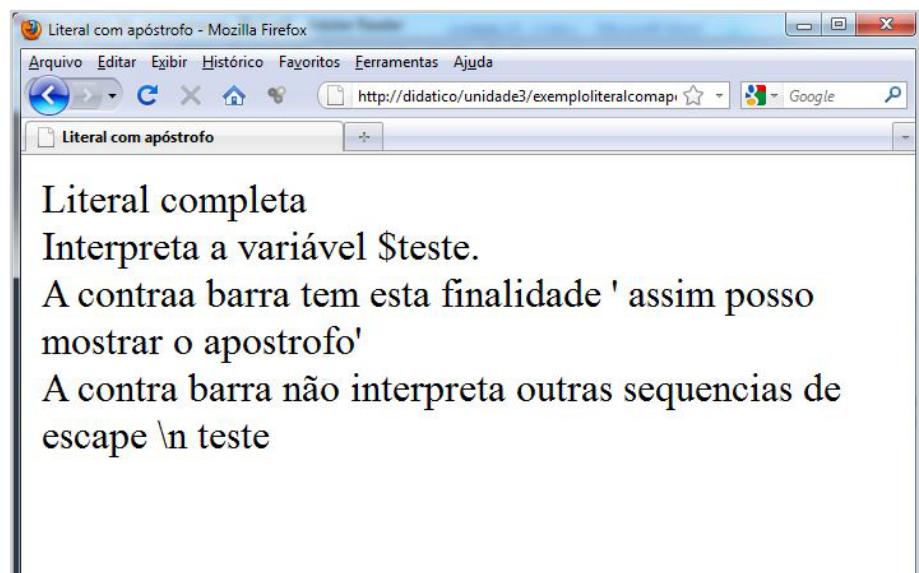


Figura 3.7 - Exemplo de variável por referência.

Fonte: Elaborada pelo autor (2010).

## b. Literais

Podemos abrir um espaço para explicar o uso das literais. As literais são **string brutas**, ou seja, não sofrem alteração. Elas estão contidas basicamente entre aspas ou apóstrofo. Porém, o PHP interpreta um pouco diferente dependendo do tipo de símbolo usado.

Vejamos a literal com apóstrofo: ela força o interpretador do PHP a ler quase todos os caracteres como literal. A exceção é o uso da contra barra antecedendo o apóstrofo, assim ele interpreta o próprio apóstrofo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>Literal com apóstrofo</title>

</head>
<body>
<?php

$teste='banana';
echo 'Literal completa';
echo '<br>';
echo 'Interpreta a variável $teste:';
echo '<br>';

echo 'A contraa barra tem esta finalidade \' assim posso mostrar o
apóstrofo\' ';
echo '<br>';
echo 'A contra barra não interpreta outras sequências de escape \n teste
';
```

```
?>  
</body>  
</html>
```

O *browser* interpreta:

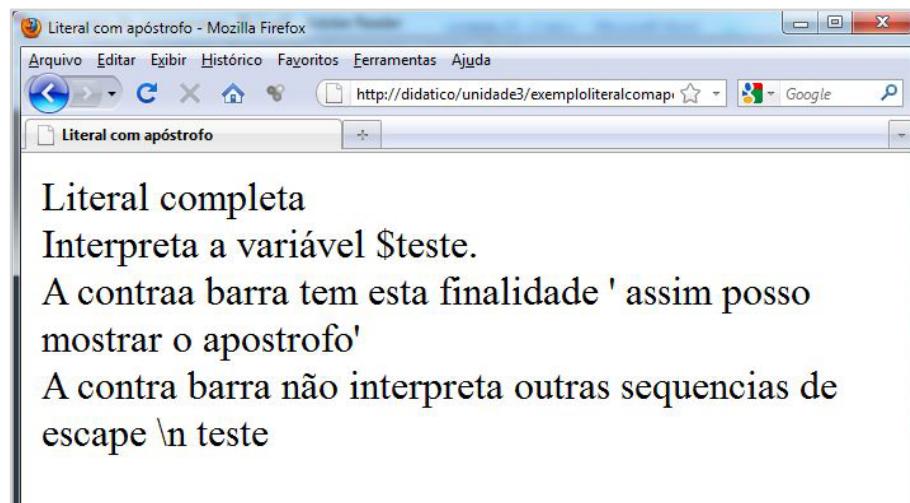


Figura 3.8 - Exemplo de literal com apóstrofo.

Fonte: Elaborada pelo autor (2010).

A literal contida “entre aspas” força o interpretador a identificar sequência de escape, além de interpretar variáveis contidas dentro das próprias literais. Utilizando o exemplo anterior teríamos outras saídas. Observe também que a concatenação entre literais e variáveis por meio do símbolo ponto (.).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml">  
  
<head>  
  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"  
/>  
  
<title>Literal com aspas</title>  
  
</head>
```

```
<body>
<?php
$teste="banana";
echo "Literal completa";
echo "<br>";
echo "Interpreta a variável $teste.";
echo "<br>";

echo "A contrabarra tem esta finalidade \" assim posso mostrar o
apóstrofo\" ";
echo "<br>";
echo "A contra barra não interpreta outras sequências de escape \n teste
"; // neste exemplo deve-se olhar o código fonte resultante... terá uma
quebra de linha no código
?>
</body>
</html>
```

O *browser* mostrará:

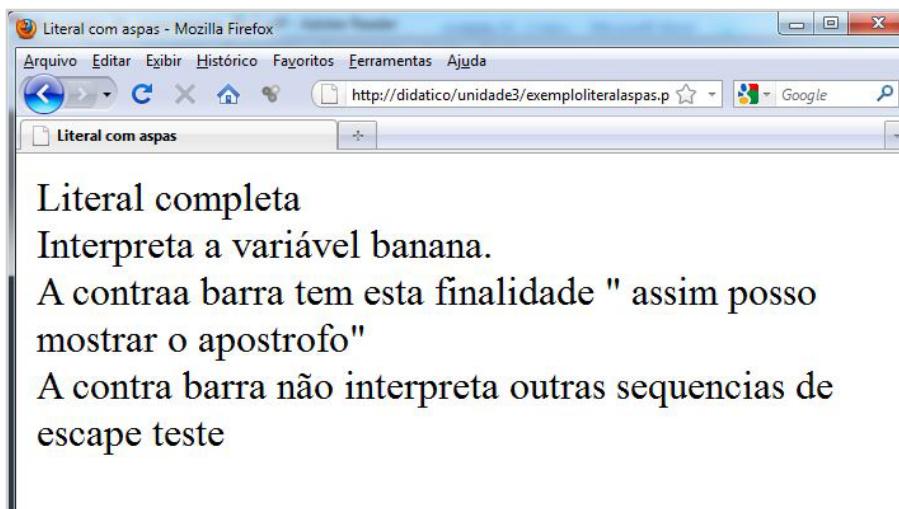


Figura 3.9 – Exemplo de literal com aspas.

Fonte: Elaborada pelo autor (2010).



O manual do PHP possui uma lista de sequência de escape que pode ser usadas, além de outras sintaxes de literais: <[http://br2.php.net/manual/pt\\_BR/language.types.string.php](http://br2.php.net/manual/pt_BR/language.types.string.php)>.

### c. Variáveis predefinidas

O PHP possui uma grande gama de variáveis predefinidas para qualquer *script* que ele execute. Elas auxiliam para a recuperação de informações importantes como de formulário, transferência de arquivo, *cookies*, entre outros.

As variáveis predefinidas mais utilizadas são:

`$_GET`: recupera informações do formulário HTML.

`$_POST`: recupera informações do formulário HTML.

`$_FILE`: é utilizada quando existe um *upload* de arquivos, assim o PHP pode acessar todas as informações pertinentes a este arquivo.

`$_SESSION`: é utilizada para registrar e recuperar variáveis de sessão.

`$_COOKIE`: possui todas as informações registradas em *cookie*. Assim, é possível recuperar suas informações.

As variáveis `$_GET`, `$_POST`, `$_FILE` e `$_SESSION` ficam disponíveis enquanto a sessão permanecer ativa. A variável `$_COOKIE` independe da sessão.



Cookies são arquivos que contém informações trocadas entre o browser e o servidor Web. A principal função é manter a persistência das seções do protocolo HTTP. Os cookies possuem um tempo de vida, não são permanentes. Eles são muito utilizados para manter um perfil do usuário a uma determinada página. Eles são vulneráveis, mas não representam risco de segurança ou privacidade ao usuário.

Todas as variáveis predefinidas são na realidade um *array* (vetor) de registro. Mesmo que elas só contenham uma informação, devem ser tratadas como um *array* em quaisquer circunstâncias.



Para saber mais sobre variáveis predefinidas, acesse o endereço <[http://www.php.net/manual/pt\\_BR/reserved.variables.php](http://www.php.net/manual/pt_BR/reserved.variables.php)>.

## d. Escopo de variáveis

O PHP tem por definição que o escopo é a abrangência em que a variável pode ser acessada em um *script*. O escopo pode ser definido em três níveis básicos no PHP, são eles:

- **Variáveis globais:** são visíveis em todo o *script*, porém não são visíveis dentro das funções.
- **Variáveis de funções:** são visíveis somente na função que a declarou.
- **Varáveis globais em funções:** são variáveis globais que são definidas como globais dentro das funções. Para isso, nas funções deve ser precedida pela palavra reservada global.

```
<?php
$var1=0;
Funcao teste() {
    Global $var1;
    ...
}
?>
```



Para saber mais sobre variáveis escopo de variáveis, acesse o endereço <[http://www.php.net/manual/pt\\_BR/language.variables.scope.php](http://www.php.net/manual/pt_BR/language.variables.scope.php)>.

## e. Constantes

As constantes são elementos de programação que não sofrem variação durante o *software*. Elas são usadas para manter informações padrões como configurações e elementos que não sofrem variação. No PHP as constantes podem ser representadas pela função *define* e teríamos:

```
define("SIMBOLOMOEDA","R");
define("DTINSTALACAO","2009-05-01");
define("COEFICIENTE",3.8);
```



---

As constantes definidas pela função *define* não necessitam do símbolo \$ e podem ser acessadas de qualquer parte do *script*.

---

## e. Data

O PHP não possui um tipo de dados que seja data. Na realidade, uma data no PHP é uma *string*. Porém, existem funções específicas que auxiliam no tratamento das datas.

A função *date* formata uma data e hora local conforme os parâmetros utilizados.

```
<?
echo date("d/m/Y");
?>
```

Retorna a data no formato DD/MM/YYYY. Lembre-se que o retorno sempre será uma *string*. Já a função *mktime* retorna um inteiro que representa um tempo no formato UNIX. Esse tempo pode ser formatado para uma saída *string*. Então para imprimir uma data 01/01/1998 13:32:12, basta o código:

```
<?php
echo date("d/m/Y H:i:s", mktime(13, 32, 12, 12, 32, 1997));
?>
```



Você pode encontrar outras funções de datas no endereço <[http://www.php.net/manual/pt\\_BR/ref.datetime.php](http://www.php.net/manual/pt_BR/ref.datetime.php)>.

## Seção 3 – Operadores

Operadores são símbolos responsáveis pelas manipulações de valores e variáveis nas linguagens. Já vimos alguns operadores em alguns exemplos. Muitos já são de conhecimento comum. Alguns operadores podem aceitar um, dois ou três argumentos, porém o mais comum é utilizar operador que aceitam dois operadores.

### a. Operadores de caractere

É possível utilizar o operador de concatenação para unir *strings* e formar um resultado de forma que se estabeleça um padrão semelhante ao da adição de números. Observe:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Operador de Caracter</title>
</head>
```

```
<body>
<?php
$parte1="Universidade do ";
$parte2="Sul de ";
$resultado=$parte1.$parte2."Santa Catarina";
echo $resultado;
?>
</body>
</tml>
```

O resultado será:

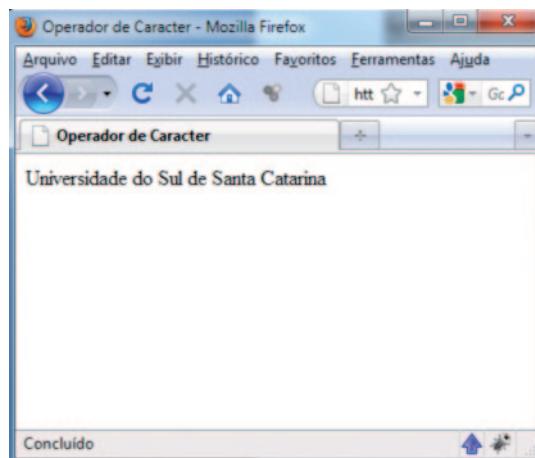


Figura 3.10 - Exemplo de operador de caractere.

Fonte: Elaborada pelo autor (2010).

## b. Operadores aritméticos

Os operadores aritméticos estão na categoria dos operadores básicos das linguagens de programação. Eles são simplesmente os operadores responsáveis pelas manipulações matemáticas.

Tabela 5 – Operadores aritméticos.

Operador	Nome	Exemplo
+	Adição	$\$a + \$b$
-	Subtração	$\$a - \$b$
*	Multiplicação	$\$a * \$b$
/	Divisão	$\$a / \$b$
%	Módulo	$\$a \% \$b$

Fonte: PHP e MySQL – desenvolvimento Web.

Os operadores de adição, subtração, multiplicação e divisão possuem seu funcionamento de forma esperada aos operadores matemáticos. O operador de módulo requer uma atenção especial, já que ele não existe nas operações matemáticas normais, e sim um resultado de uma divisão. Portanto, o operador de módulo retorna o resto de uma divisão.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>Operadores aritméticos</title>
</head>
<body>
<?php
$a=9;
$b=2;
echo "Adição: ".($a+$b)."  
";
echo "Subtração: ".($a-$b)."  
";
echo "Multiplicação: ".($a*$b)."  
";
echo "Divisão: ".($a/$b)."

```

```
echo "Módulo ou resto: ".$resultado;  
?>  
</body>  
</html>
```

O resultado será:

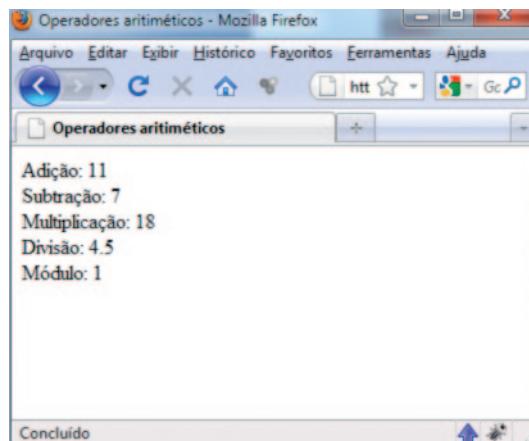


Figura 3.11 – Exemplo de operadores aritméticos.

Fonte: Elaborada pelo autor (2010).

A saída deste exemplo deverá ser o resto da divisão entre o 9 e 2 sendo, portanto, 1.

Os operadores aritméticos são usados em números. Caso você utilize esses operadores com *strings*, o PHP tentará converter para um inteiro ou decimal. Não sendo bem-sucedido nessa tentativa, ele assumirá que a *string* tem valor zero.

### c. Operadores de atribuição

O operador de atribuição básico é representado pelo símbolo de = e sempre deve ser referido como “é configurado ou definido como”. Em outras palavras, a variável \$a é definida como nove. Essa definição deve ser feita para não confundir com os operadores de comparação.

Além do operador básico, o PHP possui operadores de atribuição combinados. Assim, é possível realizar operações aritméticas juntamente com uma atribuição.

Você pode combinar os operadores aritméticos e o operador de caracteres com o de atribuição, assim teremos as seguintes combinações:

Operador	Utilização	Equivalente a
<code>+=</code>	<code>\$a += \$b</code>	<code>\$a = \$a + \$b</code>
<code>-=</code>	<code>\$a -= \$b</code>	<code>\$a = \$a - \$b</code>
<code>*=</code>	<code>\$a *= \$b</code>	<code>\$a = \$a * \$b</code>
<code>/=</code>	<code>\$a /= \$b</code>	<code>\$a = \$a / \$b</code>
<code>%=</code>	<code>\$a %= \$b</code>	<code>\$a = \$a % \$b</code>
<code>.=</code>	<code>\$a .= \$b</code>	<code>\$a = \$a . \$b</code>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Operadores de Atribuição</title>
</head>

<body>
<?php
$a = 3;
$a += 5; // configura $a para 8, como se dissessemos: $a = $a + 5;
echo $a;

echo "<br>";
```

```
$b = "Bom ";
$b .= "Dia!"; // configura $b para "Bom Dia!", como em $b = $b . "Dia!";
echo $b;
echo "<br>";

$a=23;
$b=3;
$a += $b;
echo "Adição: $a<br>";
$a -= $b;
echo "Subtração: $a<br>";
$a *= $b;
echo "Multiplicação: $a<br>";
$a /= $b;
echo "Divisão: $a<br>";
$a %= $b;
echo "Módulo ou resto: $a<br>";
?>
</body>
</html>
```

O resultado:

```

Operadores de Atribuição - Mozilla Firefox
Arquivo Editar Exibir Histórico FAVORITOS Ferramentas Ajuda
http://didatico/unidade3 Google
Operadores de Atribuição
8
Bom Dia!
Adição: 26
Subtração: 23
Multiplicação: 69
Divisão: 23
Módulo ou resto: 2

Concluído

```

Figura 3.12 - Exemplo de operadores de atribuição.

Fonte: Elaborada pelo autor (2010).

#### d. Operadores de comparação

Como o próprio nome já diz, esses operadores tem a finalidade de comparar valores. Os elementos que utilizam esses operadores retornam um valor lógico (verdadeiro ou falso):

Tabela 6 – Operadores de comparação.

Operador	Nome	Exemplo
<code>==</code>	Igual a	<code>\$a == \$b</code>
<code>====</code>	Idêntico	<code>\$a === \$b</code>
<code>!=</code>	Não igual	<code>\$a != \$b</code>
<code>&lt;&gt;</code>	Não igual	<code>\$a &lt;&gt; \$b</code>
<code>&lt;</code>	Menor que	<code>\$a &lt; \$b</code>
<code>&gt;</code>	Maior que	<code>\$a &gt; \$b</code>
<code>&lt;=</code>	Menor que ou igual a	<code>\$a &lt;= \$b</code>
<code>&gt;=</code>	Maior que ou igual a	<code>\$a &gt;= \$b</code>

Fonte: PHP e MySQL – desenvolvimento Web.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>Operadores de Comparaçāo</title>
</head>

<body>
<?php
$a="3";
$b=3;

if ($a==$b) echo "sāo idēnticas";
else echo "nāo sāo idēnticas";
echo "<br><strong>Teste igualdade</strong><br>";
if ($a==$b) echo "sāo iguais";
else echo "nāo sāo iguais";
?>

</body>
</html>
```

O resultado:

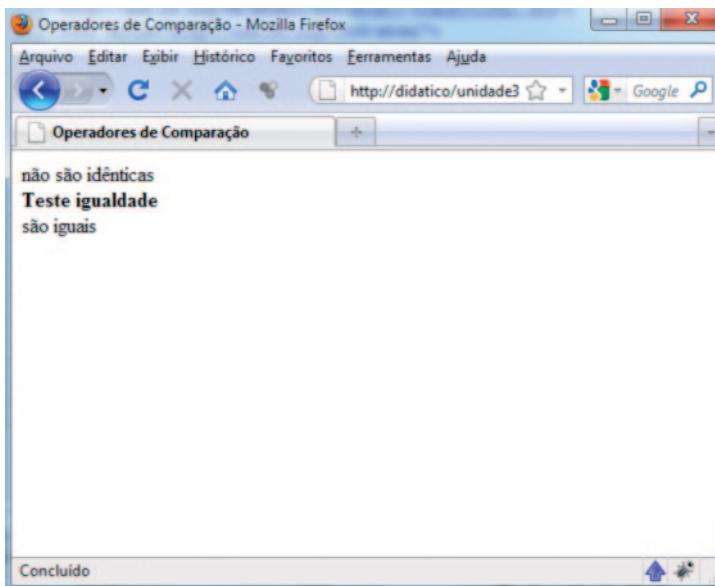


Figura 3.13 - Exemplo de operadores de comparação.

Fonte: Elaborada pelo autor (2010).




---

Operador de comparação de igualdade deve-se ter cuidado para não confundi-lo com o operador de comparação. Eles possuem funções diferentes e a não utilização correta causa um erro de lógica difícil de ser identificado.

---

O operador de comparação idêntico deve-se ser usado com cautela, já que ele não compara somente os valores como o de igualdade, mas compara o tipo também. Portanto, os valores iguais podem ser diferentes por causa do tipo definido para as variáveis.

```
<?php
$a="3";
$b=3;
if ($a===$b) echo "são idênticas";
else echo "não são idênticas";
echo "<br><strong>Teste igualdade</strong><br>";
if ($a==$b) echo "são iguais";
else echo "não são iguais";
?>
```

## e. Operadores de incremento e decremento

Esta classe de operadores se refere a uma forma de atalho para que se possa aumentar ou diminuir valores de variáveis.

Tabela 7 – Operadores de incremento/decremento.

Operador	Nome	Efeito
<code>++\$a</code>	Pré-incremento	Incrementa \$a em um, e então retorna \$a.
<code>\$a++</code>	Pós-incremento	Retorna \$a, e então incrementa \$a em um.
<code>--\$a</code>	Pré-decremento	Decrementa \$a em um, e então retorna \$a.
<code>\$a--</code>	Pós-decremento	Retorna \$a, e então decremente \$a em um.

Os operadores de incremento e decremento não podem ser usados com valores lógicos (verdadeiro/*true* ou falso/*false*). Quanto aos valores NULL, não terão resultado para o decremento, mas se for incrementado assumirá 1.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>Operadores de incremento de decremento</title>

</head>

<body>

<?php

$a=7;

echo "A variável \$a vale inicialmente: $a<br>";

++$a;

echo "A variável \$a vale agora: $a<br>";
```

```
$a++;
echo "A variável \$a vale finalmente: $a<br>";
$b=56;
echo "A variável \$b vale inicialmente: $b<br>";
--$b;
echo "A variável \$b vale agora: $b<br>";
$b--;
echo "A variável \$b vale finalmente: $b<br>";
?>
</body>
</html>
```

Resultando em:

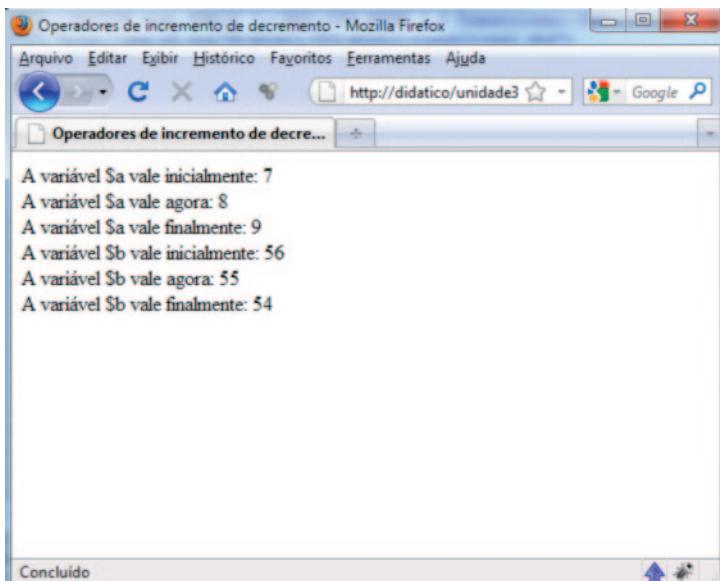


Figura 3.14 – Exemplo de operadores de incremento e decremento.

Fonte: Elaborada pelo autor (2010).



Observe mais exemplos que conta no manual do PHP <[http://www.php.net/manual/pt\\_BR/language.operators.increment.php](http://www.php.net/manual/pt_BR/language.operators.increment.php)>.

## f. Operadores lógicos

Os operadores lógicos têm a finalidade de combinar expressões de operadores lógicos. Desta forma, possibilita que várias combinações sejam feitas dentro da mesma estrutura.

Tabela 8 – Operadores lógicos.

Operador	Nome	Utilização	Resultado
!	NÃO	!\$b	Retorna verdadeiro se \$b for falso e vice-versa
&&	E	\$a && \$b	Retorna verdadeiro se \$a e \$b forem verdadeiros; caso contrário é falso
	OU	\$a    \$b	Retorna verdadeiro se \$a ou \$b ou ambos forem verdadeiros
AND	E	\$a and \$b	O mesmo que &&, mas com precedência mais baixa
OR	OU	\$a or \$b	O mesmo que   , mas com precedência mais baixa

Fonte: PHP e MySQL – desenvolvimento Web.

## Seção 4 – Estrutura de controle

As estruturas de controle são responsáveis pelas automatizações e restrições do fluxo da execução. Desta maneira, conseguimos manipular de forma mais efetiva as operações necessárias para a solução.

As estruturas de controle são elementos que permitem manter o controle sobre o fluxo da execução. Elas podem ser representadas de diversas formas, normalmente são nominadas pelo seu efeito. Basicamente podemos classificá-las de dois tipos: **tomada de decisão** e de **repetição**.

## a.Instrução de tomada de decisão: *if*

Esta instrução é responsável por comparações simples no fluxo. O *if* é a instrução básica para tomada de decisão. Ela testa uma condição e decide se deve ser executado bloco verdadeiro (quando a condição for verdadeira) ou o bloco falso (quando a condição for falsa). As instruções na condição *if* devem estar contidas dentro de parênteses.

O *if* utiliza os operadores lógicos e os operadores de comparação para construir as expressões de condição. Ainda, o bloco que define o escopo das instruções é compreendido entre chaves. Quando as chaves são omitidas, o interpretador comprehende que não existe um bloco de instrução. Consequentemente, somente a primeira instrução é executada.

```
<?php  
$a=23;  
if ($a>=18) {  
    echo "Idade: $a";  
    echo "<BR>";  
    echo "maior de idade<BR>";  
    if ($a<21)  
        echo "Você não é maior de 21 anos<BR>";  
    echo "Fim do processo";  
}  
?>
```

O *if* possui um comando de desvio, a instrução *else*. O *else* possui a finalidade de executar uma condição contraria a testada. Desta forma, o exemplo anterior ficaria:

```
<?php  
$a=23;  
if ($a>=18) {  
    echo "Idade: $a";  
    echo "<BR>";  
    echo "maior de idade<BR>";  
    if ($a<21)  
        echo "Você não é maior de 21 anos<BR>";  
    else  
        echo "Você é maior de 21 anos<BR>";  
    echo "Fim do processo";  
} else {  
    echo "Idade: $a";  
    echo "<BR>";  
    echo "menor de idade";  
}  
?>
```

O PHP suporta a instrução ***elseif***, que na prática significa encadear diversas instruções *if* e *else*. Combinando essas duas instruções em uma única estrutura:

```
<?php  
$a=4;  
$b=4;  
if ( $a > $b ) {  
    echo "a é maior que b" ;  
} elseif ( $a == $b ) {  
    echo "a é igual a b" ;  
} else {  
    echo "a é menor que b" ;  
}  
?>
```

Outra instrução que se compara ao *if* é o **operador ternário**. Esse operador é utilizado para comparação e atribuição. A sua sintaxe é:

```
condição ? valor_verdade : valor_falso
```

Assim, podemos exemplificar da seguinte forma:

```
<?php
$a=23;
$b=($a>=18)? "Maior de idade": "Menor de idade";
echo "idade: $a<br>\n$b";
?>
```

## b. Instrução de tomada de decisão – SWITCH

A instrução *switch* é semelhante à instrução *if*. Porém, ao contrário do *if* que aceita somente dois valores de resultado para a condição (verdadeiro ou falso), essa instrução pode aceitar mais dois valores.

É fundamental entender o funcionamento do *switch*. Basicamente, essa instrução busca comparação de valores, ou seja, a partir de uma variável busca-se uma lista de valores para encontrar o seu igual. Uma vez encontrado executa-se a instrução.

```
<?php
$a=4;
switch ($a) {
    case "1":
        echo "um";
        break;
    case "2":
```

```
echo "dois";
break;
case "3":
echo "três";
break;
case "4":
echo "quatro";
break;
default:
echo "padrão";
break;
}
?>
```

Uma situação especial no caso do *switch* é o uso da instrução **default**. Essa instrução é acionada quando não é encontrada nenhuma condição verdadeira dentro do case.

### c. Instrução de repetição – *while*

A instrução **while** tem um comportamento semelhante ao *if*. O *while* utiliza uma condição para que o bloco de repetição seja executado enquanto a condição se mantiver verdadeira.

O uso mais frequente desse laço é quando não podemos visualizar o número de iterações que o laço deve executar. Desta forma, o laço pode ser executado nenhuma ou **n** vezes.

```
<?php
echo "exemplo 1 <br>\n";
$i = 1;
while ($i <= 10) {
    echo ++$i."\n<br>\n";
}
echo "<br>exemplo 2 <br>";
$i = 1;
while ($i <= 10):
    echo $i."\n<br>\n";
    $i++;
endwhile;
?>
```

#### **d. Instrução de repetição – *for***

Este laço de repetição é para uso exclusivo de incremento ou decrecimento de variáveis. Assim, torna-se indispensável a informação exata de quantas iterações o laço deve executar. Utilizando o mesmo exemplo anterior podemos ter agora:

```
<?php
echo "exemplo 1 <br>\n";
for ($i=10;$i >= 1;$i--) {
    echo $i."\n<br>\n";
}
echo "<br>exemplo 2 <br>";
$i = 1;
for ($i=1;$i <= 10;$i++) {
    echo $i."\n<br>\n";
}
?>
```



Não existem diferenças de desempenho entre a instrução *for* e a instrução *while*. Você pode utilizar qualquer um dos dois em determinadas situações.

## e. Instrução de repetição – *do..while*

O PHP possui uma contração do laço *while*. Essa contração inverte a posição da condição a ser respeitada. No *while* a condição é testada no início do bloco, já na *do..while* a condição é executada no final do laço. Essa pequena diferença garante ao *do..while* que as instruções dentro do laço sejam executadas pelo menos uma única vez. Uma garantia que o *while* não pode dar.

```
<?php  
$i = 3;  
do {  
    echo $i++."<br>";  
} while ($i > 0);  
?>
```

## f. Funções

Funções são trechos de códigos destinados a executar uma tarefa específica. Qualquer código PHP pode ser escrito dentro de funções. Lembre-se que a variável de retorno não necessita ser declarada. Porém, é recomendado que você documente qual valor e tipo está sendo retornado em uma função.

Qualquer função possui a capacidade de retornar um valor, porém não é obrigatório. Uma função pode existir apenas para executar determinados códigos e não interagir com o restante do programa. O PHP aceita que somente uma variável ou constante seja retornada por uma função. Mas a variável poderá ser um *array*.

A utilização de argumentos em funções é extremamente útil, pois podemos encaminhar valores às funções para que elas possam executar suas tarefas. Os argumentos devem ser declarados logo após o nome da função, entre parênteses, e tornam-se variáveis pertencentes ao escopo local da função. Podemos utilizar três modos de argumentos em PHP.

- Argumento por cópia;
- Argumento por referência;
- Argumento *default*.

Os argumentos por cópia são valores enviados às funções onde a sua manipulação não se reflete no restante do programa. Já os argumentos por referência, afetam qualquer variável que possua a referência do argumento. O argumento *default* significa que, caso a chamada da função não contenha um valor o argumento, assume o valor preestabelecido.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>Funções no PHP</title>
</head>
<body>
<?php
function simples() {
    echo "Função sem argumentos e sem retorno<br />";
}
function retorno() {
    return "Função que contém um retorno<br />";
}

```

```
function argumentocopia ($var1,$var2, $vetor) {  
    echo "Função com argumentos por cópias<br />";  
    for ($i=0;$i<count($vetor);$i++)  
        echo "Posição $i valor: ".$vetor[$i]."<br />";  
    $var1 += 5;  
    $var2 = "Sem alteração";  
    echo "Os valores atuais das variáveis são: $var1, $var2<br />";  
}  
  
function argumentoreferencia (&$var1,$var2) {  
    echo "Função com argumentos por referencia<br />";  
    $var1 += 5;  
    $var2 = "Sem alteração";  
    echo "Os valores dentro da função nas variáveis são: $var1, $var2<br />";  
}  
  
function argumentodefault ($var1 = "Com argumentos", $var2="Sem argumentos") {  
    // pode ser utilizado com qualquer argumento  
    echo "Função com argumentos default<br />";  
    echo "O valor atual das variáveis são: $var1, $var2<br />";  
}  
simples();  
echo "<br />";  
echo retorno();  
echo "<br />";  
echo "<strong>Argumentos por cópia</strong><br />";  
  
$argumento1=10;  
$argumento2="Segundo";  
$argumento3[0]="Banana";  
$argumento3[1]="Maçã";
```

```
$argumento3[2] = "Morango";
$argumento3[3] = "Melão";

argumentocopia ($argumento1,$argumento2, $argumento3);
echo "<br />";
echo "As variáveis no programa não sofreram alteração<br />";
echo "Valor da variável argumento1: $argumento1, valor da variável
argumento2: $argumento2<br />";
for ($i=0;$i<count($argumento3);$i++)
    echo "Posição $i valor: ".$argumento3[$i]."<br />";
echo "<br />";

echo "<strong>Argumentos por referência</strong><br />";

$argumento1=45;
$argumento2="Dois";
echo "Os valores dos argumentos antes de encaminhar a função são:
$argumento1, $argumento2<br />";
argumentoreferencia ($argumento1,$argumento2);
echo "Após a função ser executada apenas a variável por referência terá
seu valor alterado: $argumento1, $argumento2<br />";
echo "<br />";

echo "<strong>Argumentos com valor default</strong><br />";
argumentodefault("Encaminhado Valor");
?>
</body>
</html>
```

Você pode mesclar esses tipos de argumentos, ou seja, em uma mesma função você pode conter todos os tipos de argumentos. Os valores *default* não são obrigatórios em todos os argumentos. Você pode utilizar em um único argumento e nos demais não.

Função sem argumentos e sem retorno

Função que contém um retorno

**Argumentos por cópia**

Função com argumentos por cópias

Posição 0 valor: Banana

Posição 1 valor: Maçã

Posição 2 valor: Morango

Posição 3 valor: Melão

Os valores atuais das variáveis são: 15, Sem alteração

As variáveis no programa não sofreram alteração

Valor da variável argumento1: 10, valor da variável argumento2: Segundo

Posição 0 valor: Banana

Posição 1 valor: Maçã

Posição 2 valor: Morango

Posição 3 valor: Melão

**Argumentos por referência**

Os valores dos argumentos antes de encaminhar a função são: 45, Dois

Função com argumentos por referencia

Os valores dentro da função nas variáveis são: 50, Sem alteração

Após a função ser executada apenas a variável por referência terá seu valor alterado: 50, Dois

**Argumentos com valor default**

Função com argumentos default

O valor atual das variáveis são: Encaminhado Valor, Sem argumentos

Figura3.15 - Exemplo de função no PHP.

Fonte: Elaborada pelo autor (2010).

## Seção 5 – Trabalhando com arquivos

Uma funcionalidade muito comum em diversas linguagens é a manipulação de arquivos. Normalmente é utilizada para ler ou escrever um arquivo de comunicação, ou seja, um arquivo que contém informações pertinentes a uma integração assíncrona.




---

Lembre-se sempre que manipular arquivos requer permissão de acesso ao arquivo ou diretório. Sempre verifique se os acessos no servidor estão corretos.

---




---

Para simplificar o estudo, vamos adotar como padrão a manipulação de arquivos somente. Mesmo assim você pode ampliar seus estudos consultando o manual do PHP no endereço <[http://br2.php.net/manual/pt\\_BR/ref.filesystem.php](http://br2.php.net/manual/pt_BR/ref.filesystem.php)>.

---

### *fopen*

A função *fopen* é responsável por abrir um URL. Ou seja, informando um endereço o PHP tentará encontrar qual o protocolo resolverá a URL. Assim como interpreta que a abertura de um arquivo físico é uma forma de protocolo.

A sintaxe dessa função é:

```
fopen(url,mode)
```

Podemos representar da seguinte forma:

```
<?php  
$arquivo = fopen('c:\teste.txt','r');  
?>
```

O **mode** é na realidade o modo em que você trabalhará a URL. Esses tipos de manipulação estão descritos na tabela a seguir:

Tabela 9 – Tipos de abertura de arquivos.

<b>mode</b>	<b>Descrição</b>
'r'	Abre somente para leitura; coloca o ponteiro do arquivo no começo do arquivo.
'r+'	Abre para leitura e escrita; coloca o ponteiro do arquivo no começo do arquivo.
'w'	Abre somente para escrita; coloca o ponteiro do arquivo no começo do arquivo e reduz o comprimento do arquivo para zero. Se o arquivo não existir, tenta criá-lo.
'w+'	Abre para leitura e escrita; coloca o ponteiro do arquivo no começo do arquivo e reduz o comprimento do arquivo para zero. Se o arquivo não existir, tenta criá-lo.
'a'	Abre somente para escrita; coloca o ponteiro do arquivo no final do arquivo. Se o arquivo não existir, tenta criá-lo.
'a+'	Abre para leitura e escrita; coloca o ponteiro do arquivo no final do arquivo. Se o arquivo não existir, tenta criá-lo.
'x'	Cria e abre o arquivo somente para escrita; coloca o ponteiro no começo do arquivo. Se o arquivo já existir, a chamada a <b>fopen()</b> falhará, retornando <b>FALSE</b> e gerando um erro de nível <b>E_WARNING</b> . Se o arquivo não existir, tenta criá-lo. Isto é equivalente a especificar as flags <b>O_EXCL O_CREAT</b> para a chamada de sistema <b>open(2)</b> .
'x+'	Cria e abre o arquivo para leitura e escrita; coloca o ponteiro no começo do arquivo. Se o arquivo já existir, a chamada a <b>fopen()</b> falhará, retornando <b>FALSE</b> e gerando um erro de nível <b>E_WARNING</b> . Se o arquivo não existir, tenta criá-lo. Isso é equivalente a especificar as flags <b>O_EXCL O_CREAT</b> para a chamada de sistema <b>open(2)</b> .

Em contrapartida, a função *fclose* fecha o arquivo aberto. Normalmente após a execução do *script* o sistema fecha as seções abertas. Porém, é aconselhável sempre fechar manualmente para evitar que o arquivos seja corrompido quando fechado pelo encerramento do programa.

## **Fwrite**

A função *fwrite* é responsável em escrever no arquivo aberto. Essa é uma das principais funções dentro da categoria de manipulação de arquivos. Com ela podemos imprimir todas as informações necessárias dentro um arquivo especificado.

```
<?php

$filename = 'teste.txt';
$conteudo = "Acrescentar isso ao arquivo\n";

// Primeiro vamos ter certeza de que o arquivo existe e pode ser alterado
if (is_writable($filename)) {
    // caso exita vamos adicionar
    if (!$handle = fopen($filename, 'a')) {
        echo "Não foi possível abrir o arquivo ($filename)";
        exit;
    }
} else {
    // caso não exista vamos criá-lo
    if (!$handle = fopen($filename, 'w+')) {
        echo "Não foi possível abrir o arquivo ($filename)";
        exit;
    }
}

// Escreve $conteudo no nosso arquivo aberto.
if (fwrite($handle, $conteudo) === FALSE) {
    echo "Não foi possível escrever no arquivo ($filename)";
    exit;
}
echo "Sucesso: Escrito ($conteudo) no arquivo ($filename)";
fclose($handle);
?>
```

Observe que nesse exemplo foi forçado o teste para verificar se o arquivo existe e pode ser gravado. Utilizando o mesmo exemplo e abrindo o arquivo no modo w+ já seria o suficiente para criar ou reescrever o arquivo.



Lembre-se sempre que o manual do PHP possui referências e exemplos nos quais você pode testar. Acesse [http://br2.php.net/manual/pt\\_BR/function.fwrite.php](http://br2.php.net/manual/pt_BR/function.fwrite.php).

## Fgets

O *fgets* é a função de leitura de uma linha. Ou seja, com ela lemos uma linha inteira e acrescentamos a uma variável. Uma vez na variável podemos interpretar e manipular a informação conforme a necessidade.

```
<?php  
//para este exemplo é necessário que o arquivo arquivodeentrada.txt  
//exista no mesmo diretório que o script está salvo  
if ($arquivo = @fopen("arquivodeentrada.txt", "r")) {  
    $contador=1;  
    // feof verifica se chegou a fim do arquivo  
    while (!feof($arquivo)) {  
        $leitura = fgets($arquivo);  
        echo "Linha ".$contador++.": $leitura<br>";  
    }  
    fclose($arquivo);  
} else {  
    echo "Não foi possível abrir o arquivo";  
}  
?>
```

A manipulação é extremamente importante, pois é muito utilizada para a troca de informações *off-line*. Em diversas situações, existe a necessidade de exportar determinadas informações. Isso pode ocorrer por meio de um arquivo formatado ou até mesmo por XML. A função básica é criar o arquivo e ler linha por linha a fim de identificar seu conteúdo.



## Síntese

Neste capítulo você será apresentado à linguagem serve-side PHP. O PHP é uma linguagem simples de ser desenvolvida em função da sua estrutura. Tanto é possível utilizar técnicas avançadas como orientação a objetos quanto simplificar o desenvolvimento utilizando uma programação estruturada.

Como era de se esperar, o interpretador é extremamente dinâmico. Não é necessário se preocupar com declarações de variáveis. O PHP se encarrega de interpretar o tipo de dados a ser trabalhado. Já quanto aos vetores, possui um leque de opções que supre praticamente quaisquer necessidades que nos deparemos.

Ainda oferece a possibilidade de trabalhar com arquivos de uma forma simples. Os arquivos são muito utilizados como relatórios e comunicação de dados de uma forma off-line, ou seja, a comunicação não se dá em tempo real.



## Atividades de autoavaliação

Ao final de cada unidade, você será provocado a realizar atividades de autoavaliação. O gabarito está disponível no final do livro-didático. Mas se esforce para resolver as atividades sem ajuda do gabarito, pois assim você estará estimulando a sua aprendizagem.

1. Faça um script para validar um formulário utilizando a linguagem PHP. O formulário deve conter nome, endereço, nome da mãe, idade e e-mail. Faça uma página de resposta para que o PHP leia e escreva na página o resultado do formulário. A idade deve ser superior a dois anos e o e-mail deve ser validado.

2. Acrescente ao exercício anterior a criação de um arquivo texto das informações provenientes do formulário. O arquivo texto deve conter os seguintes padrões:

Nome

Endereço

Nome da mãe

Idade

E-mail

Escreva um terceiro para ler essas informações do arquivo e organizá-los em uma tabela do HTML. Publique sua resposta na ferramenta Exposição do EVA.



### Saiba mais

Se você desejar, aprofunde os conteúdos estudados nesta unidade ao consultar as seguintes referências:

Manual PHP, 2010. Disponível em <[http://br2.php.net/manual/pt\\_BR/index.php](http://br2.php.net/manual/pt_BR/index.php)>. Acesso em: 5 abr. 2010.

PHP Brasil, 2010. Disponível em <[www.phpbrasil.com](http://www.phpbrasil.com)>. Acesso em: 5 abr. 2010.

Truquese Dicas – Tutoriais de Javascript, 2010. Disponível em <[www.truquesedicas.com/tutoriais/php/index.htm](http://www.truquesedicas.com/tutoriais/php/index.htm)>. Acesso em: 2 mar. 2010.

## UNIDADE 4

# 4

## Sistemas Web com banco de dados (PHP + MYSQL)



### Objetivos de aprendizagem

Ao final desta unidade, você terá subsídios para:

- Delimitação de aplicações Web com acesso a banco de dados.
- Saber realizar consultas.
- Trabalhar com manipulação de dados.
- Utilizar sessões de aplicações Web.



### Seções de estudo

**Seção 1** Banco de dados

**Seção 2** Banco de dados e acessos

**Seção 3** Trabalhando com tabelas

**Seção 4** API do PHP

**Seção 5** Envio e recuperação de dados

**Seção 6** Sessões e *cookies*



## Para início de estudo

Quando falamos em sistemas de *software*, é natural pensar em armazenamento e recuperação de dados. Embora possamos classificar a maioria dos sistemas desta maneira, existem sistemas que tratam de problemas cujo foco não é o armazenamento e a recuperação.

A manipulação de dados pode acontecer de qualquer forma que se permita salvar esses dados e, normalmente, essa ação utiliza um banco de dados. Ao utilizar um banco de dados, você tem vantagens, como acesso rápido aos dados, facilidade em extrair conjuntos de dados, mecanismos predefinidos para lidar com acesso concorrente, entre outros.

Veremos como o PHP trabalha com um banco de dados, mais especificamente com o MySQL. O seu escopo de trabalho, como manipular dados, e como montar uma sessão de acesso.

Vale registrar que neste ponto será abordado de forma superficial as funcionalidades do banco de dados. O foco principal de estudo é **como utiliza-lo para desenvolver sistemas dinâmicos**.

## Seção 1 – Banco de dados

Um banco de dados tem a finalidade de proporcionar mecanismos que facilitem as transações de armazenamento de dados. Esses dados são formatados em tabelas cada qual contendo colunas e linhas.

Essas tabelas podem se relacionar entre si utilizando colunas chamadas **chaves**, que nada mais são que os identificadores dos dados. Além de se saber identificar e trabalhar com eles, você deve saber projetar um banco de dados para Web.

## Banco de dados na Web

A diferença crucial entre um desenvolvimento com banco de dados entre um sistema *desktop* e um sistema Web é que no sistema *desktop* normalmente há uma conexão única com o banco de dados. Enquanto a aplicação permanecer aberta, a conexão não se desfaz. Quando desenvolvemos em um sistema Web, o funcionamento é um pouco diferente, pois toda a operação de conexão com o banco de dados se dá no servidor.

Quem faz a chamada é uma aplicação externa que não mantém uma conexão com esse servidor de forma permanente.

Podemos considerar que a cada resposta do servidor Web a conexão se desfaz. Desta forma, não há necessidade de se manter uma conexão aberta entre a linguagem *server-side* e o banco de dados. Em termos gerais, você pode perceber que toda a página que necessitar conectar ao banco de dados deverá abrir uma nova conexão e ao final essa conexão será fechada. Ainda nesta unidade, retomaremos esse assunto para seu melhor entendimento.

## Modelo relacional

O modelo relacional ainda é o principal modelo utilizado hoje em dia, mesmo tendo sido criado na década de 1970. Como já foi comentado, o sistema gerenciador de banco de dados utiliza o armazenamento de dados em tabelas que possuem relações entre si.

**Assim, a ideia do modelo relacional é que os dados correlatos devem estar contidos na mesma tabela e utilizamos relacionamentos entre tabelas para agruparmos dados menos correlatos.**



Considerando este exemplo: um cadastro de clientes deve-se ter as informações pessoais de cada cliente, porém, nesta mesma tabela, acrescentar as informações de endereço torna-se redundante. Por causa disso, criamos uma segunda tabela com os endereços comuns (rua, CEP, bairro, cidade, estado). Na tabela cliente, apenas informamos a referência do endereço, e não todo ele.

Observe que nesse exemplo podemos usar mais relacionamentos. Realizar a análise para diminuir a redundância de informações chama-se **normalização de dados**.

É importante destacar também que os relacionamentos utilizam tipos específicos de campos na tabela. Esses tipos são denominados de **chaves primárias e estrangeiras**.

Podemos resumir que uma coluna, ou campo, é um pedaço simples de dados definido por um tipo, tamanho ou outras características. Já a linha, ou registro, é um simples conjunto ordenado de colunas. Portanto, uma tabela contém um número fixo de colunas e qualquer número de linhas.

**Podemos concluir que um banco de dados é um conjunto de TABELAS.**

## MySQL

O MySQL é um banco de dados SQL multiusuário e multi-*threading* (executa eventos em paralelo).

Normalmente os sistemas que utilizam o MySQL não possuem um grande volume de informações. Ele acaba sendo utilizado em todo o seu potencial para aplicações com baixo a médio volume de dados.

O MySQL é utilizado em diversos sistemas e tem como características marcantes **estabilidade e agilidade nas operações**. Hoje, possui controle de transação, suporte a *procedure*, entre outras funcionalidades, e utiliza como base o modelo relacional de gerenciamento de dados e sua linguagem padrão é o SQL. Ainda mantém uma característica que o tornou popular: **facilidade de gerenciamento**.

O MySQL possui algumas particularidades que facilitam o desenvolvimento de aplicações. Perceba que, além das informações comuns do SQL, como NOT NULL ou PRIMARY KEY, ao final do comando existem algumas

informações adicionais que são de suma importância. Essas informações são os chamados *Storage Engines*.

## Storage engines

*Storage engine* refere-se ao tipo de armazenamento que o banco de dados utilizará. Não são todos os bancos de dados que podem alterar o tipo de armazenamento conforme a necessidade. O MySQL permite você alterar esse tipo de armazenamento para cada tabela criada. Portanto, podemos escolher qual a melhor forma de armazenamento para cada aplicação a ser desenvolvida.



Você pode estar se questionando: "para que mudar o tipo de armazenamento?" Simplesmente para adequar a realidade do problema.

Cada *storage engine* possui algumas características nas quais pode se fazer necessário para melhorar a solução do problema. Desta forma, para chegar à melhor solução, é necessário que se conheça as principais características inerentes aos *storage engines*:

- **Transacional:** característica em que a tabela aceita múltiplos acessos. Isso significa que uma ação de um usuário ou aplicação não interfere com a operação do outro. Pode executar comandos em blocos (transação) em vez de executar um comando SQL por vez.
- **Armazenamento:** o MySQL permite gravar a tabela 100% em memória, isto é, nada no disco. Pode-se gravar dados em uma TABLESPACE, ou seja, um arquivo onde se armazena todas as tabelas do banco. Pode-se utilizar ainda uma forma de arquivamento extremamente rápida, conhecida como ISAM, para gravar dados e recuperá-los de forma ágil. Outro tipo de *engine* pode gravar os dados de forma compactada, economizando muito espaço em disco. Ainda, pode-se gravar em formato CSV (arquivo com delimitador), que facilita bastante a integração com equipamentos de rede e telefonia, por exemplo. E mais, em vez de ler e gravar os

dados no servidor onde o MySQL está instalado, pode-se espalhar os dados por vários computadores para se criar um cluster de alta disponibilidade e alta performance.

- **Índices:** dependendo do motor, temos índices do tipo B-TREE, B+TREE, RTREE (*spatial*) ou FULL TEXT (que indexa palavras, em vez da coluna ou do campo inteiro, e permite buscas como fazemos no Google, digitando palavras fora de ordem).
- **Integridade Referencial:** são as *Foreign Key* (chaves estrangeiras). Não são todos os *engines* que a utiliza. Como todos os recursos que estamos discutindo, dependendo da aplicação ou finalidade da tabela isso não é necessário, e é uma funcionalidade que pesa para o banco de dados. Às vezes, não ter este recurso pode ser uma vantagem em termos de velocidade.
- **Lock:** é a capacidade de poder travar um único registro (linha ou tupula), vários registros ou a tabela inteira. Cada *engine* tem um ou mais tipo de travamentos à disposição da tabela.
- **Backup On-Line:** o MySQL permite realizar *backups* em processo *batch* ou ainda *on-line*. Algumas *storage engines* permitem tal funcionalidade.
- **Auto Recovery:** há *engines* que, caso haja uma corrupção de índice, por exemplo, você pode utilizar o comando REPAIR TABLE. Há outros, no entanto, que no máximo você será avisado, por meio do *log* de erro, que o MySQL server encontrou uma inconsistência e já consertou.

A imagem a seguir ilustra a arquitetura do MySQL, conectores, *engines* e o gerenciador de banco de dados propriamente dito.

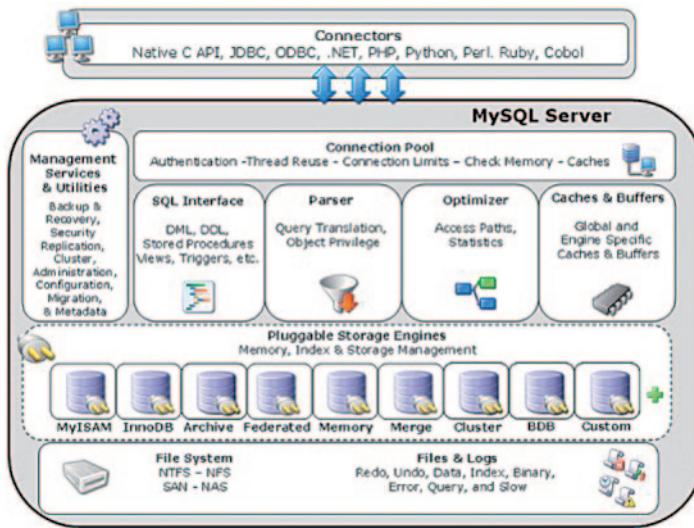


Figura 4.1 – Arquitetura do MySQL.

Fonte: <[http://forge.mysql.com/wiki/MySQL\\_Internals\\_Custom\\_Engine](http://forge.mysql.com/wiki/MySQL_Internals_Custom_Engine)>. Acesso em 06 jan. 2011.

- *Management Server & Utilities* – são as ferramentas de administração, backup, segurança, configuração, migração, entre outras.
- *Connection Pool* – é a camada que faz a autenticação, gerencia a *thread* (processo), limites de conexão, memórias e caches (atalho de processo).
- *SQL Interface* – camada que controla as estruturas do SQL como procedimentos, *views*, *triggers*, entre outros.
- *Parser* – verifica a sintaxe do comando SQL e os privilégios de acesso.
- *Optimizer* – monta as estatísticas do banco de dados para eventuais relatórios de desempenho.
- *Caches & buffers* – camada onde se cria atalhos e alocação de memórias para comandos e estruturas específicas.
- *Pluggable Storage Engines* – camada responsável pelo gerenciamento e controle da estrutura de armazenamento.
- *File System* – controla o arquivamento físico conforme a formatação do sistema operacional.

- *Files & logs* - gerencia arquivos físicos diversos de controles (*logs*, índices, erros etc) .

As *storages engines* mais comuns são a **InnoDB** e a **MyISAM**. Veremos agora as principais características destes *storages engines*.

## InnoDB

O InnoDB foi desenvolvido para obter o máximo de performance ao processar grande volume de dados e é usado na produção de vários *sites* com banco de dados grandes e que necessitam de alto desempenho.

Este *storage* provê o MySQL com um mecanismo de armazenamento seguro com transações com *commit*, *rollback* e recuperação em caso de falhas. O InnoDB faz bloqueio a nível de registro e também fornece uma leitura sem bloqueio em SELECT em um estilo consistente com Oracle. Esses recursos aumentam a performance e a concorrência de multiusuários. InnoDB é o primeiro gerenciador de armazenamento no MySQL que suporta restrições FOREIGN KEY.

Tecnicamente, InnoDB é um banco de dados completo colocado sob o MySQL. InnoDB tem sua própria área de *buffer* para armazenar dados e índices na memória principal. InnoDB armazena suas tabelas e seus índices em uma *tablespace*, o qual pode consistir de vários arquivos. Isso é diferente, por exemplo, de tabelas MyISAM, em que cada tabela é armazenada como um arquivo separado. Tabelas InnoDB podem ser de qualquer tamanho, mesmo em sistemas operacionais em que o sistema de arquivo é limitado a 2 GB.

## MyISAM

As tabelas MyISAM possuem algumas características que podem auxiliar na solução do problema. Você pode compactar tabelas MyISAM para utilizar menos espaço. Você pode INSERIR novas linhas em uma tabela que não tenha blocos livres no meio

do arquivo de dados, na mesma hora, outras *threads* (processo de conexão) são lidas da tabela (inserção concorrente).

Um bloco livre pode vir de uma atualização de uma linha de tamanho dinâmico com muitos dados para uma linha com menos dados ou ao deletarmos linhas. Quando todos os blocos livres são usados, todas as inserções futuras serão concorrentes de novo.

Todo dado é armazenado com *byte* mais baixo primeiro, ou seja, o dado que possui menor tamanho físico. Isso torna a máquina e o sistema operacional independentes. A única exigência para a portabilidade do arquivo binário é que a máquina utilize inteiros com sinais em complemento de dois e formato de pontos flutuante IEEE.

Todas as chaves numéricas estão armazenadas com o *byte* mais alto em primeiro para conseguir melhor compactação do índice.

Não há uma grande perda de velocidade em realizar esse tipo de armazenamento. O código atual busca-valor-coluna também não é crítico em relação ao tempo comparado a outro código.

O tratamento interno para uma coluna com a propriedade de AUTO\_INCREMENT (numeração automática sequencial para uma chave primária) em uma Engine MyISAM é de atualizar automaticamente após INSERT/UPDATE seus índices de controle. Ele fará colunas AUTO\_INCREMENT mais rápidas (pelo menos 10%) e números antigos não serão reutilizados.

Ao inserir ordenadamente (como quando se utiliza colunas AUTO\_INCREMENT), a árvore-chave será separada de forma que o nó do mais alto contenha apenas uma chave. Isso aumentará a utilização de espaço na árvore de chaves. As colunas BLOB e TEXT podem ser indexadas, assim como os valores NULL são permitidos em colunas indexadas.

## Seção 2 – Banco de dados e acessos

Relembramos alguns conceitos de banco de dados e conhecemos algumas características inerentes ao MySQL que nos ajudaram em um desenvolvimento. Portanto, alguns procedimentos são ideais para que possamos trabalhar.



Não vamos entrar na instalação do MySQL, uma vez que, pela diversidade de sistemas operacionais e constante atualização banco, fica inviável a manutenção de um tutorial para instalação. Qualquer dúvida, deve-se solicitar ao professor da disciplina.

### Criando uma base de dados

O conjunto de caracteres aceito em um banco de dados é denominado ***collation***. No MySQL não podem ser nulos. Os comandos CREATE DATABASE e ALTER DATABASE possuem cláusulas opcionais para especificarem o *collation* de banco de dados:

```
CREATE DATABASE db_name  
[DEFAULT CHARACTER SET character_set_name [COLLATE collation_  
name]]  
  
ALTER DATABASE db_name  
[DEFAULT CHARACTER SET character_set_name [COLLATE collation_  
name]]  
  
Exemplo:  
  
CREATE DATABASE db_name  
DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

O *collation* do banco de dados utiliza valores padrões caso não sejam informados durante a criação deste banco. O valor padrão é definido na configuração do MySQL. Assim, para você criar seu primeiro banco, deve acessar o MySQL com o usuário de administrador (normalmente é o **login root**) por meio de um cliente SQL conectado ao banco de dados.

Em seguida, crie uma base de dados (ou *schema*, ou banco de dados).

O comando SQL para isso é *create database*.



Vamos supor que o nome da base de dados seja prg3.

Então o comando ficaria:

`CREATE DATABASE prg3 ;`



Lembre-se que o SQL não é *case-sensitive*, portanto não diferencia maiúsculas de minúsculas.

Após executar esse comando, o banco de dados será criado com as configurações padrões. Caso queira trocar o nome do banco ou *collation* criada, basta utilizar o comando `ALTER DATABASE`. Vamos supor que fosse necessário alterar o comando o *collation* para Sueco, sem diferenciar maiúsculas ou minúsculas. Faríamos o seguinte comando:

```
ALTER DATABASE prg3 DEFAULT CHARACTER SET latin1 COLLATE latin1_general_ci;
```

Lembre-se que o *collation* do banco de dados não impede que a tabela seja criada em outra coleção. Apenas indica qual o padrão a ser utilizado pelo banco. Veja a seguir uma tabela com os *collations* mais utilizados.

Collation	Descrição
<b>big5_bin</b>	Chinês Tradicional, Binário
<b>binary</b>	Binário
<b>cp1250 (Windows Central European)</b>	
<b>cp1250_bin</b>	Europeu Central (multilinguagem), Binário
<b>cp1250_croatian_ci</b>	Croata, Não diferencia maiúsculas/minúsculas
<b>cp1250_czech_cs</b>	Tcheco, Diferencia maiúsculas/minúsculas
<b>cp1250_general_ci</b>	Europeu Central (multilinguagem), Não diferencia maiúsculas/minúsculas
<b>cp1250_polish_ci</b>	Polonês, Não diferencia maiúsculas/minúsculas

Collation	Descrição
<b>cp852 (DOS Central European)</b>	
<b>cp852_bin</b>	Europeu Central (multilinguagem), Binário
<b>cp852_general_ci</b>	Europeu Central (multilinguagem), Não diferencia maiúsculas/minúsculas
<b>cp866 (DOS Russian)</b>	
<b>cp866_bin</b>	Russo, Binário
<b>cp866_general_ci</b>	Russo, Não diferencia maiúsculas/minúsculas
<b>latin1 (cp1252 West European)</b>	
<b>latin1_bin</b>	Oeste Europeu (multilinguagem), Binário
<b>latin1_danish_ci</b>	Dinamarquês, Não diferencia maiúsculas/minúsculas
<b>latin1_general_ci</b>	Oeste Europeu (multilinguagem), Não diferencia maiúsculas/minúsculas
<b>latin1_general_cs</b>	Oeste Europeu (multilinguagem), Diferencia maiúsculas/minúsculas
<b>latin1_german1_ci</b>	Alemão (dicionário), Não diferencia maiúsculas/minúsculas
<b>latin1_german2_ci</b>	Alemão (Agenda de telefones), Não diferencia maiúsculas/minúsculas
<b>latin1_spanish_ci</b>	Espanhol, Não diferencia maiúsculas/minúsculas
<b>latin1_swedish_ci</b>	Suéco, Não diferencia maiúsculas/minúsculas
<b>latin2 (ISO 8859-2 Central European)</b>	
<b>latin2_bin</b>	Europeu Central (multilinguagem), Binário
<b>latin2_croatian_ci</b>	Croata, Não diferencia maiúsculas/minúsculas
<b>latin2_czech_cs</b>	Tcheco, Diferencia maiúsculas/minúsculas
<b>latin2_general_ci</b>	Europeu Central (multilinguagem), Não diferencia maiúsculas/minúsculas
<b>latin2_hungarian_ci</b>	Húngaro, Não diferencia maiúsculas/minúsculas
<b>latin5 (ISO 8859-9 Turkish)</b>	
<b>latin5_bin</b>	Turco, Binário
<b>latin5_turkish_ci</b>	Turco, Não diferencia maiúsculas/minúsculas
<b>latin7 (ISO 8859-13 Baltic)</b>	
<b>latin7_bin</b>	Báltico (multilinguagem), Binário
<b>latin7_estonian_ci</b>	Estoniano, Diferencia maiúsculas/minúsculas

Collation	Descrição
<b>latin7_general_ci</b>	Báltico (multilinguagem), Não diferencia maiúsculas/minúsculas
<b>latin7_general_cs</b>	Báltico (multilinguagem), Diferencia maiúsculas/minúsculas
<b>macce (Mac Central European)</b>	
<b>macce_bin</b>	Europeu Central (multilinguagem), Binário
<b>macce_general_ci</b>	Europeu Central (multilinguagem), Não diferencia maiúsculas/minúsculas
<b>macroman (Mac West European)</b>	
<b>macroman_bin</b>	Oeste Europeu (multilinguagem), Binário
<b>macroman_general_ci</b>	Oeste Europeu (multilinguagem), Não diferencia maiúsculas/minúsculas
<b>sjis (Shift-JIS Japanese)</b>	
<b>sjis_bin</b>	Japonês, Binário
<b>sjis_japanese_ci</b>	Japonês, Não diferencia maiúsculas/minúsculas
<b>swe7 (7bit Swedish)</b>	
<b>swe7_bin</b>	Suéco, Binário
<b>swe7_swedish_ci</b>	Suéco, Não diferencia maiúsculas/minúsculas
<b>utf8 (UTF-8 Unicode)</b>	
<b>utf8_bin</b>	Unicode (multilinguagem), Binário
<b>utf8_czech_ci</b>	Tcheco, Não diferencia maiúsculas/minúsculas
<b>utf8_danish_ci</b>	Dinamarquês, Não diferencia maiúsculas/minúsculas
<b>utf8_esperanto_ci</b>	Esperanto, Não diferencia maiúsculas/minúsculas
<b>utf8_estonian_ci</b>	Estoniano, Não diferencia maiúsculas/minúsculas
<b>utf8_general_ci</b>	Unicode (multilinguagem), Não diferencia maiúsculas/minúsculas
<b>utf8_hungarian_ci</b>	Húngaro, Não diferencia maiúsculas/minúsculas
<b>utf8_icelandic_ci</b>	Islandês, Não diferencia maiúsculas/minúsculas
<b>utf8_latvian_ci</b>	Letônio, Não diferencia maiúsculas/minúsculas
<b>utf8_lithuanian_ci</b>	Lituânia, Não diferencia maiúsculas/minúsculas
<b>utf8_persian_ci</b>	Pérsia, Não diferencia maiúsculas/minúsculas
<b>utf8_polish_ci</b>	Polonês, Não diferencia maiúsculas/minúsculas
<b>utf8_roman_ci</b>	Oeste Europeu, Não diferencia maiúsculas/minúsculas

Collation	Descrição
<b>utf8_romanian_ci</b>	Romêno, Não diferencia maiúsculas/minúsculas
<b>utf8_slovak_ci</b>	Eslováquio, Não diferencia maiúsculas/minúsculas
<b>utf8_slovenian_ci</b>	Eslovênia, Não diferencia maiúsculas/minúsculas
<b>utf8_spanish2_ci</b>	Espanhol Traditional, Não diferencia maiúsculas/minúsculas
<b>utf8_spanish_ci</b>	Espanhol, Não diferencia maiúsculas/minúsculas
<b>utf8_swedish_ci</b>	Suéco, Não diferencia maiúsculas/minúsculas
<b>utf8_turkish_ci</b>	Turco, Não diferencia maiúsculas/minúsculas
<b>utf8_unicode_ci</b>	Unicode (multilinguagem), Não diferencia maiúsculas/minúsculas

Fonte: <<http://www.mysql.com>>.

## Criando usuário

Um fator importante para o desenvolvimento Web é a segurança dos seus dados. Vale a pena ressaltar que, por causa da arquitetura da Web, o sistema fica mais exposto a pessoas mal intencionadas. Por isso, uma atenção à segurança empregada é fundamental.

O MySQL usa o comando *Grant* para criar e conceder privilégios para os usuários. Esse procedimento é importante para que possamos aumentar a segurança da aplicação. O MySQL não valida somente o *login* e a senha, mas também o endereço de onde se está conectando. Assim, mesmo que alguém mal intencionado descubra o *login* e a senha, ele somente poderá utilizar pelo servidor que ela está configurada. Para ser mais claro, vamos observar o registro de *login* do usuário do MySQL. A sintaxe completa do comando é:

```

GRANT priv_type [(column_list)] [, tipo_priv [(column_list)] ...]

    ON {tbl_name | * | *.* | db_name.*}

    TO user_name [!IDENTIFIED BY [PASSWORD] 'password']

        [, user_name [!IDENTIFIED BY [PASSWORD] 'password'] ...]

    [REQUIRE

        NONE |

        [{SSL| X509}]

    [CIPHER cipher [AND]]

    [ISSUER issuer [AND]]

    [SUBJECT subject]]

    [WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR # |

            MAX_UPDATES_PER_HOUR # |

            MAX_CONNECTIONS_PER_HOUR #]]

```

Porém, podem existir casos em que a revogação dos direitos ou privilégios do usuário se faz necessária. Para isso, o comando a ser utilizado é o **REVOKE**. A sua sintaxe é:

```

REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]

    ON {tbl_name | * | *.* | db_name.*}

    FROM user_name [, user_name ...]

```

Esse comando permite manipular os privilégios em quatro níveis: global, banco de dados, tabelas, colunas.

## 1. Nível global

Neste nível, os privilégios são aplicados a todos os bancos de dados de um servidor. Podem ser concedidos com o comando **GRANT ALL ON \*.\*** e revogados com o comando **REVOKE ALL ON \*.\***.

## 2. Nível dos bancos de dados

Aplicam-se a todas as tabelas em um determinado banco de dados. A sintaxe para esta segurança são **GRANT ALL ON db.\*** e **REVOKE ALL ON db.\*** e concederão e revogarão apenas privilégios de banco de dados.

## 3. Nível das tabelas

O nível de tabelas aplica-se às colunas em uma determinada tabela. Para este privilégio, teremos de detalhar mais a sintaxe do comando utilizando **GRANT ALL ON db.table** e **REVOKE ALL ON db.table** e concederão e revogarão apenas privilégios de tabelas.

## 4. Nível das colunas

Privilégios de colunas aplicam-se a uma única coluna em uma determinada tabela. Para este comando, é necessário especificar cada acesso individualmente para cada coluna.

Lembre-se que para o privilégio de **DELETE** sempre se refere a todo o registro. Não é possível conceder um delete no nível de coluna.

Além dos níveis, podemos ter o tipo de privilégios os quais podemos relacionar na tabela a seguir.

Privilégios	Descrição
<b>ALL [PRIVILEGES]</b>	Configura todos os privilégios simples, exceto <b>WITH GRANT OPTION</b> .
<b>ALTER</b>	Permite o uso de <b>ALTER TABLE</b> .
<b>CREATE</b>	Permite o uso de <b>CREATE TABLE</b> .
<b>CREATE TEMPORARY TABLES</b>	Permite o uso de <b>CREATE TEMPORARY TABLE</b> .
<b>DELETE</b>	Permite o uso de <b>DELETE</b> .
<b>DROP</b>	Permite o uso de <b>DROP TABLE</b> .
<b>EXECUTE</b>	Permite que o usuário execute <i>STORED PROCEDURES</i> .
<b>FILE</b>	Permite o uso de <b>SELECT ... INTO OUTFILE</b> e <b>LOAD DATA INFILE</b> .
<b>INDEX</b>	Permite o uso de <b>CREATE INDEX</b> e <b>DROP INDEX</b> .
<b>INSERT</b>	Permite o uso de <b>INSERT</b>
<b>LOCK TABLES</b>	Permite o uso de <b>LOCK TABLES</b> em tabelas nas quais se tem o privilégio <b>SELECT</b> .
<b>PROCESS</b>	Permite o uso de <b>SHOW FULL PROCESSLIST</b> .
<b>REFERENCES</b>	Para o futuro.
<b>RELOAD</b>	Permite o uso de <b>FLUSH</b> .
<b>REPLICATION CLIENT</b>	Dá o direto ao usuário de perguntar onde o <i>slave/master</i> está. Utilizando em replicação de banco de dados.
<b>REPLICATION SLAVE</b>	Necessário para a replicação dos slaves (para ler logs binário do master).
<b>SELECT</b>	Permite o uso de <b>SELECT</b> .
<b>SHOW DATABASES</b>	<b>SHOW DATABASES</b> exibe todos os bancos de dados..
<b>SHUTDOWN</b>	Permite o uso de <b>mysqladmin shutdown</b> .
<b>SUPER</b>	Permite a conexão (uma vez) mesmo se <code>max_connections</code> tiverem sido alcançados e executa o comando <b>CHANGE MASTER, KILL thread, mysqladmin debug, PURGE MASTER LOGS e SET GLOBAL</b> .
<b>UPDATE</b>	Permite o uso de <b>UPDATE</b>
<b>USAGE</b>	Sinônimo para "sem privilégios".
<b>GRANT OPTION</b>	Sinônimo para <b>WITH GRANT OPTION</b> .

Os únicos valores do tipo de privilégio que você pode especificar para uma tabela são: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, GRANT, INDEX e ALTER.

Já para colunas você pode especificar para uma coluna são: SELECT, INSERT, UPDATE e REFERENCES, como o mencionado anteriormente.

Atenção:

- O MySQL permite que você crie privilégios a nível de banco de dados mesmo se o banco de dados não existir para tornar fácil de se preparar para o uso do banco de dados. O MySQL não revogará automaticamente qualquer privilégio, mesmo se você apagar uma tabela ou um banco de dados.
- Você pode configurar privilégios globais utilizando a sintaxe ON \*.\*.
- Você pode configurar privilégios de bancos de dados utilizando a sintaxe ON nome\_bd.\*.
- Se você especificar ON \* e estiver com algum banco de dados aberto, serão configurados os privilégios somente para este banco de dados.
- Se você especificar ON \* e você não tem possui um banco de dados aberto, afetarão os privilégios globais.

Os privilégios para uma tabela ou coluna são formados por meio do OU lógico dos privilégios em cada um dos quatro níveis de privilégios. Por exemplo, um usuário tem um privilégio global *select*, isso não pode ser negado por uma entrada no nível de banco de dados, tabela ou coluna. Os privilégios para uma coluna podem ser calculados da seguinte forma:

privilégios globais OU (privilégios de banco de dados E privilégios de máquina) OU privilégios de tabela OU privilégios de coluna.

Na maioria dos casos, os direitos a um usuário são atribuídos em apenas um dos níveis de privilégios, portanto a situação não é tão complicada como mostrado anteriormente.

Um novo usuário é criado ou, se você possui privilégios de concessão globais, a senha do usuário será especificada utilizando a cláusula IDENTIFIED BY, se uma for dada. Se o usuário já possui uma senha, ela é trocada pela nova.

Se você não quiser enviar a senha em texto puro, você pode usar a opção PASSWORD seguida de uma senha embaralhada da função SQL PASSWORD().



---

Se você criar um novo usuário, mas não especificar uma cláusula IDENTIFIED BY, o usuário não possuirá uma senha. Isso não é seguro.

---

Somente serão apresentadas as tabelas ou os banco de dados pelo comando SHOW TABLES (por exemplo) que o usuário têm acesso. Se um usuário não tem privilégios em uma tabela, ela não é mostrada quando o usuário solicita uma lista de tabelas ou de banco de dados.

A cláusula WITH GRANT OPTION dá ao usuário habilidade de fornecer a outros usuários quaisquer privilégios que ele tenha em um nível específico de privilégio. Tenha sempre cuidado ao fornecer o privilégio Grant.

Você não pode conceder a outro usuário um privilégio que não possua; o privilégio GRANT possibilita fornecer somente os privilégios que possuir.

Esteja ciente que, quando conceder a um usuário o privilégio GRANT em um nível particular de privilégios, qualquer privilégio que o usuário já possua nesse nível também pode ser concedido por esse usuário.

Suponha que você conceda a um usuário o privilégio INSERT em um banco de dados. Se você conceder o privilégio SELECT no banco de dados e especificar WITH GRANT OPTION, o usuário, além de poder repassar o privilégio SELECT, poderá repassar o *insert*. Se você concede o privilégio UPDATE para o usuário no banco de dados, o usuário poderá conceder os privilégios INSERT, SELECT e UPDATE.

Quando o MySQL inicia, todos os privilégios são lidos na memória. Privilégios de bancos de dados, tabelas e colunas são iniciados um vez, e privilégios ao nível de usuário fazem efeito na próxima vez que o usuário conectar. Modificações nas tabelas de permissões que você realiza utilizando GRANT ou REVOKE são percebidas pelo servidor imediatamente. Quaisquer outras alterações deverão ser acrescidas do comando FLUSH PRIVILEGES, assim dizemos ao servidor para recarregar as tabelas de permissões.

Vamos criar um usuário para que possamos iniciar a construção de uma aplicação. Vamos supor que o nome do usuário seja **web** e a senha seja **senha123**. O comando ficaria:

```
GRANT SELECT , INSERT ,UPDATE ,DELETE, RELOAD, REFERENCES, SHOW DATABASES, LOCK TABLES ON prg3.* to web@localhost identified by 'senha123';
```

Esse exemplo é o mais simples em criação de usuários e estamos concedendo privilégio de selecionar, inserir, alterar e excluir os dados da base de dados prg3 para todas as tabelas (prg3.\*). O *login* do usuário ficou ‘web’ e somente poderá ser acessado do próprio servidor que o MySQL está instalado.



Para que esse usuário funcione na aplicação, é obrigatório que o servidor Web também esteja instalado na mesma máquina. Caso esteja em outra, não terá permissão para se conectar no banco de dados.

Perceba que a arquitetura da internet possibilita essa configuração, mesmo que várias pessoas acessem um site qualquer. O site localiza-se em um servidor e todas as conexões que saírem dele sempre serão feitas pelo servidor Web, mesmo que a solicitação seja feita por um usuário na internet.

- Lembre-se que, por causa da arquitetura da internet, deve-se sempre ter o servidor Web muito bem configurado para que não deixem pessoas mal intencionadas acessar serviços derivados dele.

- Sempre tenha um projeto definido para o banco de dados. Caso o seu banco não esteja bem definido, será difícil armazenar e recuperar as informações

## Seção 3 – Trabalhando com tabelas

Já criamos o banco de dados e o usuário que será acessado pela aplicação. Agora veremos a criação das tabelas desse banco de dados. Para isso, o comando a ser utilizado é o CREATE TABLE e sua sintaxe completa é:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nome_tabela [(definição_create,...)]
[table_options] [select_statement]
```

ou

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nome_tabela [(]LIKE nome_antigo_tabela[]);
```

### **definição\_create:**

```
nome_coluna tipo [NOT NULL | NULL] [DEFAULT valor_padrão] [AUTO_INCREMENT]
|[PRIMARY] KEY [COMMENT 'string'] [definição_referência]
| [CONSTRAINT [symbol]] PRIMARY KEY (index_col_name,...)
| KEY [nome_indice] (index_nome_coluna,...)
| INDEX [nome_indice] (index_nome_coluna,...)
| [CONSTRAINT [symbol]] UNIQUE [INDEX] [index_name] (index_col_name,...)
| FULLTEXT [INDEX] [nome_indice] (index_nome_coluna,...)
| [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,...)
| [definição_referência]
| CHECK (expr)
```

**tipo:**

```
TINYINT[(tamanho)] [UNSIGNED] [ZEROFILL]
| SMALLINT[(tamanho)] [UNSIGNED] [ZEROFILL]
| MEDIUMINT[(tamanho)] [UNSIGNED] [ZEROFILL]
| INT[(tamanho)] [UNSIGNED] [ZEROFILL]
| INTEGER[(tamanho)] [UNSIGNED] [ZEROFILL]
| BIGINT[(tamanho)] [UNSIGNED] [ZEROFILL]
| REAL[(tamanho,decimais)] [UNSIGNED] [ZEROFILL]
| DOUBLE[(tamanho,decimais)] [UNSIGNED] [ZEROFILL]
| FLOAT[(tamanho,decimais)] [UNSIGNED] [ZEROFILL]
| DECIMAL(tamanho,decimais) [UNSIGNED] [ZEROFILL]
| NUMERIC(tamanho,decimais) [UNSIGNED] [ZEROFILL]
| CHAR(tamanho) [BINARY | ASCII | UNICODE]
| VARCHAR(tamanho) [BINARY]
| DATE
| TIME
| TIMESTAMP
| DATETIME
| TINYBLOB
| BLOB
| MEDIUMBLOB
| LONGBLOB
| TINYTEXT
| TEXT
| MEDIUMTEXT
| LONGTEXT
| ENUM(value1,value2,value3,...)
| SET(value1,value2,value3,...)
```

**index\_nome\_coluna:**

nome\_coluna [(tamanho)] [ASC | DESC]

### **definição\_referência:**

```
REFERENCES nome_tabela [(index_nome_coluna,...)]
    [MATCH FULL | MATCH PARTIAL]
    [ON DELETE opção_referência]
    [ON UPDATE opção_referência]
```

### **opção\_referência:**

RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

### **opções\_tabela:**

table\_option [table\_option] ...

### **opções\_tabela:**

```
ENGINE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }
| AUTO_INCREMENT = #
| AVG_ROW_LENGTH = #
| CHECKSUM = {0 | 1}
| COMMENT = 'string'
| MAX_ROWS = #
| MIN_ROWS = #
| PACK_KEYS = {0 | 1 | DEFAULT}
| PASSWORD = 'string'
| DELAY_KEY_WRITE = {0 | 1}
| ROW_FORMAT = { DEFAULT | DYNAMIC | FIXED | COMPRESSED }
| RAID_TYPE = { 1 | STRIPED | RAID0 } RAID_CHUNKS=# RAID_CHUNKSIZE=#
| UNION = (table_name,[table_name...])
| INSERT_METHOD = { NO | FIRST | LAST }
| DATA DIRECTORY = 'caminho absoluto para o diretório'
| INDEX DIRECTORY = 'caminho absoluto para o diretório'
| DEFAULT CHARACTER SET character_set_name [COLLATE collation_name]
```

Obviamente, não utilizamos normalmente essa sintaxe completa, mas uma sintaxe em que podemos definir o nome da tabela, os campos, o relacionamento e o tipo da tabela. Vale à pena ressaltar que a tabela temporária (quando especificamos a instrução TEMPORARY) é visível apenas para a conexão atual, e será automaticamente deletada quando a conexão é fechada. Isso significa que duas conexões diferentes podem usar o mesmo nome de tabela temporária sem conflitos com outras ou com uma tabela existente com o mesmo nome.

Para a arquitetura das aplicações Web, onde, via de regra, as conexões são finalizadas a cada retorno da requisição, o seu uso fica ainda mais limitado. Essa tabela temporária será acessada somente durante o processamento da página pelo servidor. Quando a requisição retorna ao usuário, a conexão fecha e a tabela é excluída.

Quando falamos de tipos de registros, podemos descrever as seguintes características:

1. **TINYINT[(M)] [UNSIGNED] [ZEROFILL]**

Um inteiro muito pequeno. A faixa desse inteiro com sinal é de -128 até 127. A faixa sem sinal é de 0 até 255.

2. **BIT, BOOL, BOOLEAN**

Estes são sinônimos para TINYINT(1).

3. **SMALLINT[(M)] [UNSIGNED] [ZEROFILL]**

Um inteiro pequeno. A faixa do inteiro com sinal é de -32768 até 32767. A faixa sem sinal é de 0 a 65535.

4. **MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]**

Um inteiro de tamanho médio. A faixa com sinal é de -8388608 a 8388607. A faixa sem sinal é de 0 a 16777215.

5. **INT[(M)] [UNSIGNED] [ZEROFILL]**

Um inteiro de tamanho normal. A faixa com sinal é de -2147483648 a 2147483647. A faixa sem sinal é de 0 a 4294967295.

6. **INTEGER[(M)] [UNSIGNED] [ZEROFILL]**

Este é um sinônimo para INT.

## 7. BIGINT[(M)] [UNSIGNED] [ZEROFILL]

Um inteiro grande. A faixa com sinal é de -9223372036854775808 a 9223372036854775807. A faixa sem sinal é de 0 a 18446744073709551615.

Existem algumas coisas sobre campos BIGINT sobre as quais você deve estar ciente:

- Todas as operações aritméticas são feitas usando valores BIGINT ou DOUBLE com sinal, não devemos utilizar inteiros sem sinal maiores que 9223372036854775807 (63 bits), exceto com funções ded bit! Se você fizer isso, alguns dos últimos digitos no resultado podem estar errados por causa de erros de arredondamento na conversão de BIGINT para DOUBLE.
- Quando usar operadores (+, -, \*, etc.) em que ambos os operandos são inteiros.
- Você pode armazenar valores inteiro exatos em um campo BIGINT armazenando-os como *string*, como ocorre nestes casos, não haverá nenhuma representação intermediaria dupla.
- ‘-’, ‘+’, e ‘\*’ serão utilizados em cálculos aritiméticos BIGINT quando ambos os argumentos forem valores do tipo INTEGER! Isso significa que, se você multiplicar dois inteiros grandes (ou obter resultados de funções que retornam inteiros), você pode obter resultados inesperados quando o resultado for maior que 9223372036854775807.

## 8. FLOAT(precisão) [UNSIGNED] [ZEROFILL]

Um número de ponto flutuante. Um ponto flutuante é a definição dada ao processador interpretar os números reais. Podemos considerar que engloba o tipo ponto flutuante qualquer número que possa ser representado por meio de uma notação científica (Nº intero multiplicado por uma base elevado a um expoente). Esses tipos são como os tipos FLOAT e DOUBLE descritos logo a seguir. FLOAT(X) tem a mesma faixa que os tipos correspondentes FLOAT e DOUBLE, mas o tamanho do display e o número de casas decimais

é indefinido. Note que o uso de FLOAT pode trazer alguns problemas inesperados, como nos cálculos, já que em MySQL todos são feitos com dupla precisão.

9. FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]

Um número de ponto flutuante pequeno (precisão simples). Os valores permitidos estão entre -3.402823466E+38 e -1.175494351E-38, 0 e entre 1.175494351E-38 e 3.402823466E+38. Se UNSIGNED for especificado, valores negativos não são permitidos. O M é a largura do *display* e o D é o número de casas decimais. FLOAT sem um argumento ou FLOAT(X) onde X <=24 tende a um número de ponto flutuante de precisão simples.

10. DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]

Um número de ponto flutuante de tamanho normal (dupla precisão). Valores permitidos estão entre -1.7976931348623157E+308 e -2.2250738585072014E-308, 0 e entre 2.2250738585072014E-308 e 1.7976931348623157E+308. Se UNSIGNED for especificado, valores negativos não são permitidos. O M é a largura do *display* e o D é número de casa decimais. DOUBLE sem argumento ou FLOAT(X) onde 25 <= X <= 53 são números de ponto flutuante de dupla precisão.

11. DOUBLE PRECISION[(M,D)] [UNSIGNED]  
[ZEROFILL], REAL[(M,D)] [UNSIGNED]  
[ZEROFILL]

Estes são sinônimos para DOUBLE.

12. DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]

Um número de ponto flutuante não empacotado. Comporta-se como um campo CHAR: ``não empacotado'' significa que o número é armazenado como uma *string*, usando um caractere para cada dígito do valor. O ponto decimal e, para números negativos, o sinal de menos ('-'), não são contados em M (mas é reservado espaço para isso). Se D for 0, os valores não terão ponto decimal ou parte fracionária. A faixa máxima do valor DECIMAL é a mesma do DOUBLE, mas a faixa atual para um campo DECIMAL dado pode

ser limitado pela escolha de M e D. Se UNSIGNED é especificado, valores negativos não são permitidos. Se D não for definido, será considerado como 0. Se M não for definido, é considerado como 10.

13. DEC[(M[,D])] [UNSIGNED] [ZEROFILL],  
 NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL],  
 FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]  
 Este é um sinônimo para DECIMAL.

#### 14. DATE

Uma data. A faixa suportada é entre ‘1000-01-01’ e ‘9999-12-31’. MySQL mostra valores DATE no formato ‘AAAA-MM-DD’, mas permite a você atribuir valores a campos DATE utilizando tanto *strings* quanto números.

#### 15. DATETIME

Uma combinação de hora e data. A faixa suportada é entre ‘1000-01-01 00:00:00’ e ‘9999-12-31 23:59:59’. MySQL mostra valores DATETIME no formato ‘AAAA-MM-DD HH:MM:SS’, mas permite a você atribuir valores a campos DATETIME utilizando *strings* ou números.

#### 16. TIMESTAMP[(M)]

Um *timestamp*. A faixa é entre ‘1970-01-01 00:00:00’ e algum momento no ano 2037.

## Propriedades das colunas

As colunas ainda possuem propriedades que ajudam a restringir o seu conteúdo assim como acrescenta regras ao seu banco de dados:

- Se nem NULL nem NOT NULL forem especificados, a coluna é tratada como se NULL fosse especificado.

Uma coluna *integer* pode ter o atributo adicional AUTO\_INCREMENT. Quando você insere um valor de NULL ou 0 em uma coluna AUTO\_INCREMENT indexada, a coluna é definida com o valor da próxima sequência. Normalmente

ele é valor+1, onde valor é o maior valor para a coluna *column* atualmente na tabela. A sequência de AUTO\_INCREMENT começa com 1.

- Se você deletar a linha contendo o valor máximo para uma coluna AUTO\_INCREMENT, o valor não será reutilizado por tabelas MyISAM ou InnoDB. Se você deletar todas as linhas na sua tabela com DELETE FROM nome\_tabela (sem um WHERE) no modo AUTOCOMMIT, a sequência será reiniciada em todos os tipos de tabela, exceto InnoDB.



Apenas poderá existir uma coluna AUTO\_INCREMENT por tabela. Ainda deve ser indexado e não terá valor DEFAULT tampouco valores negativos.

Em tabelas MyISAM e BDB, você pode especificar colunas AUTO\_INCREMENT secundárias em uma chave multicolumna.

Um valor padrão (DEFAULT) tem de ser constante, ele não pode ser uma função ou uma expressão.

Se nenhum valor DEFAULT é especificado para uma coluna, o MySQL atribuirá um automaticamente, como a seguir:

- Se a coluna aceitar NULL como um valor, o valor padrão é NULL.
- Se a coluna é declarada como NOT NULL, o valor padrão depende do tipo de coluna:
  - Para tipos numéricos não declarados com o atributo AUTO\_INCREMENT, o padrão é 0.
  - Para uma coluna AUTO\_INCREMENT, o valor padrão é o próximo valor na sequência.
  - Para tipos *date* e *time* diferentes de TIMESTAMP, o padrão é o valor zero apropriado para o tipo. Para a primeira coluna TIMESTAMP na tabela, o padrão é a data e hora atuais.

- Para tipos *strings* diferentes de ENUM, o valor padrão é uma *string* vazia.
- Para ENUM, o padrão é o primeiro valor enumerado.

**Valores padrões devem ser constantes.** Isso significa, por exemplo, que você não pode definir o padrão de uma coluna *date* como o valor de funções como NOW() or CURRENT\_DATE.

Um **comentário** para uma coluna pode ser especificado com a opção COMMENT. O comentário é mostrado pela instrução SHOW CREATE TABLE e por SHOW FULL COLUMNS.

KEY é normalmente um sinônimo para INDEX. O atributo de chave PRIMARY KEY também pode ser especificado apenas como KEY.

No MySQL, uma chave UNIQUE só pode ter valores distintos. Um erro ocorre se você tentar adicionar uma nova linha com uma chave que coincida com uma já existente.

## Primary Key

PRIMARY KEY é uma chave única (KEY) onde todas as colunas-chave devem ser definidas como NOT NULL. Se elas não forem explicitamente declaradas como NOT NULL, isso será feito implicitamente e sem aviso. No MySQL, a chave é chamada PRIMARY. Uma tabela pode ter apenas uma PRIMARY KEY. Se você não tiver uma PRIMARY KEY e alguma aplicação perguntar pela PRIMARY KEY em sua tabela, o MySQL retornará à primeira chave UNIQUE, que não possui nenhuma coluna NULL, como a PRIMARY KEY.

Uma PRIMARY KEY pode ser um índice multicoluna. Porém, você não pode criar um índice multicoluna usando o atributo de chave PRIMARY KEY em uma especificação de coluna. Fazendo assim apenas colunas simples poderão ser marcadas como primárias. Você deve utilizar uma cláusula PRIMARY KEY(index\_nome\_coluna, ...) separada.

Um índice UNIQUE é aquele no qual todos os valores no índice devem ser distintos. A exceção a isso é que, se for permitido conter valores NULL em uma coluna no índice, ele pode conter múltiplos valores NULL. Essa exceção não se aplica a tabelas BDB, que permitem apenas um único NULL. Se a chave PRIMARY ou UNIQUE consistir de apenas uma coluna e ela é do tipo inteiro, você também poderá se referir a ela como \_rowid.

Se você não atribuir um nome ao índice que não é um PRIMARY KEY, ele terá o mesmo nome da primeira index\_nome\_coluna, com um sufixo opcional (\_2, \_3, ...) para torná-lo único. Você pode nomear índices para uma tabela usando SHOW INDEX FROM nome\_tabela.

Apenas os tipos de tabelas MyISAM, InnoDB, e BDB suportam índices em coluna que possam ter valores NULL. Nos outros casos, você deve declarar tais colunas NOT NULL ou um erro será retornado.

Com a sintaxe nome\_coluna(length) em uma especificação de índice, você pode criar um índice que utiliza apenas os primeiros length() bytes de uma coluna CHAR ou VARCHAR. Isso pode tornar o arquivo de índices muito menor.

Apenas os tipos de tabela MyISAM e InnoDB suportam índice em colunas BLOB e TEXT. Ao colocar um índice em uma coluna BLOB ou TEXT, você sempre DEVE especificar o tamanho do índice, até 255 bytes. Por exemplo:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Uma especificação index\_col\_name pode finalizar com ASC ou DESC. Essas palavras-chaves são permitidas para extensão futura para especificar o armazenamento do valor do índice em crescente ou decrescente. Atualmente elas são analisadas, mas ignoradas; valores de índice são sempre armazenados em ordem crescente.

Quando você utiliza ORDER BY ou GROUP BY com uma coluna TEXT ou BLOB, o servidor ordena valores usando apenas o número inicial de bytes, indicado pela variável do servidor max\_sort\_length.

Você também pode criar índices FULLTEXT especiais. Eles são usados para busca *full-text*. Apenas o tipo de tabela MyISAM suporta índices FULLTEXT. Eles só podem ser criados em colunas CHAR, VARCHAR e TEXT. A indexação sempre ocorre sobre toda a coluna; índices parciais não são suportados.

Tabelas InnoDB suportam verificação de chaves estrangeiras. Note que a sintaxe FOREIGN KEY no InnoDB é mais restrita que a sintaxe apresentada anteriormente. As colunas da tabela indicada devem ser nomeadas explicitamente. O InnoDB suporta ambas as ações ON DELETE e ON UPDATE em chaves estrangeiras. Para outros tipos de tabelas, MySQL Server analisa as sintaxes FOREIGN KEY, CHECK e REFERENCES no comando CREATE TABLE, mas sem tal ação ser tomada.

Para tabelas ISAM e MyISAM, cada coluna NULL tem um *bit* extra, arredondado para o *byte* mais próximo. O tamanho máximo de um registro em *bytes* pode ser calculado como a seguir:

```
tamanho da linha = 1
+ (soma do tamanho da coluna)
+ (números de coluna NULL + delete_flag 7)/8
+ (número de colunas de tamanho variável)
```

`delete_flag` é 1 para tabelas com formato de registro estático. Tabelas estáticas usam um *bit* no registro para um parâmetro que indica se a linha foi deletada. `delete_flag` é 0 para tabelas dinâmicas porque esse parâmetro é armazenado no cabeçalho da linha dinâmica.

Esses cálculos não se aplicam a tabelas InnoDB, para a qual o tamanho do armazenamento não é diferente para colunas NULL comparados a colunas NOT NULL.

Para ajudar a ter um melhor entendimento, vamos utilizar um conjunto pequeno de tabelas em que possamos sintetizar as informações e nos focar na lógica embutida.

Relembrando que o comando para a criação de tabelas é o *create table*. Ainda como forma didática, vamos supor que tenhamos

dois cadastros: livros e autores. Para criar essas tabelas, utilize os comandos a seguir.

```
CREATE TABLE prg3.autores (
    cod_autor int(11) NOT NULL AUTO_INCREMENT,
    nom_autor varchar(50) NOT NULL,
    PRIMARY KEY (cod_autor),
    KEY nom_autor (nom_autor)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Neste caso, estamos criando a tabela **autores** dentro da base de dados já criada **prg3**. Essa tabela possui dois campos: *cod\_autor*, *nom\_autor*. A chave primária é o campo *cod\_autor*. Vamos inserir os seguintes dados:

```
INSERT INTO autores (cod_autor, nom_autor) VALUES (NULL, 'Fulano de Tal');
INSERT INTO autores (nom_autor) VALUES ('Outro Autor');
```

Perceba a sutil diferença entre as duas instruções.

A primeira instrução informa que o campo *cod\_autor* receberá NULL para forçar a ação do AUTO\_INCREMENT. No segundo comando, o campo é omitido e com isso forçamos a interpretação do gerenciador que o campo está recebendo NULL e terá o mesmo comportamento da primeira instrução. Portanto, se selecionarmos os registros, deverá retornar os seguintes dados:

```
SELECT cod_autor, nom_autor FROM autores;
```

cod_autor	nom_autor
1	Fulano de Tal
2	Outro Autor

Já criamos uma tabela criada de autores, portanto nada mais natural que criar uma segunda tabela de livros.

```
CREATE TABLE IF NOT EXISTS livros (
    cod_livro int(11) NOT NULL AUTO_INCREMENT,
    nom_livro varchar(50) NOT NULL,
    cod_autor int(11) DEFAULT NULL,
    PRIMARY KEY (cod_livro),
    KEY nom_livro (nom_livro,cod_autor),
    KEY cod_autor (cod_autor)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Temos uma chave primária *cod\_livro*. O nome do livro (*nom\_livro*) que possui um índice composto com o *cod\_autor*. Por fim, o *cod\_autor*. Perceba que apenas existe o código do autor. Esse campo relacionará com a tabela autor para que possamos minimizar dados, redundância e possamos melhor organizar as informações. Vamos ao segundo passo, vamos relacionar as duas tabelas.

```
ALTER TABLE livros
    ADD CONSTRAINT livros_fk_1 FOREIGN KEY (cod_autor) REFERENCES
    autores (cod_autor) ON DELETE SET NULL ON UPDATE SET NULL;
```

Vale a ressalva que a criação do relacionamento poderia ser realizada no próprio comando CREATE.

Agora podemos inserir dados na tabela de livros:

```
INSERT INTO livros (cod_livro ,nom_livro ,cod_autor)
VALUES (NULL , 'PHP', '1'),
       (NULL , 'MySQL', '1');
```

Desta vez, em um único comando inserimos dois registros. No MySQL esse comando é muito útil para economizar instruções a serem realizadas.

Selecionaremos a tabela livros e relacionaremos a tabela autor. Além de um comando, já se torna base de um relatório plausível.

```
SELECT cod_livro, nom_livro, nom_autor  
FROM livros a, autores b  
WHERE a.cod_autor = b.cod_autor;
```

cod_livro	nom_livro	nom_autor
2	MySQL	Fulano de Tal
1	PHP	Fulano de Tal

Quando igualamos os valores dos campos relacionados, otimizamos a busca e condicionamos o resultado à verificação válida dos registros. Ou seja, somente retornará as informações que se relacionarem.

Vimos que o **MySQL** é um banco de dados consistente e simplificado. A sua instalação, o seu gerenciamento e os seus controles são postos de tal forma que facilitam o trabalho do desenvolvedor.

Você pode pesquisar mais o MySQL no endereço: <<http://www.mysql.com>>.

## Seção 4 – API do PHP

O PHP utiliza interfaces que são chamadas de API para realizar a conexão com o banco de dados. Ele possui uma gama de APIs que facilitam a conexão com diversos bancos de dados.

Estas APIs caracterizam-se por serem extremamente eficientes a ágeis quanto a sua resposta de requisitos. Porém, o principal problema encontrado é quando você necessitar trocar o banco de dados. Vamos ver dois exemplos comparativos para que fique mais claro. O primeiro sendo o PHP:

```
<?php
$con = mysql_connect($host, $user, $pass);
mysql_select_db($dbnome);
$query = "select * from tabela";
$qry = mysql_query($query, $con);
While ( $row = mysql_fetch_assoc($qry) {
    echo $row["primeiroNome"];
    echo $row["segundoNome"];
}
mysql_free_result($qry);
mysql_close($con);
?>
```

## Perl com DBI:

```
Use DBI;
my $con = DBI -> connect("dbi:mysql:$ dbnome", $user, $pass);
my $qry = $con -> do ("select * from tabela");
while (my $row = $qry->fetchrow_hashref) {
    print $row->( primeiroNome);
    print $row->( segundoNome);
}
$qry->finish;
$con->disconnect;
```

Perceba que o nível de complexidade é semelhante entre as duas instruções. Porém, o exemplo da linguagem Perl é mais portátil. Fica comprehensível se tivermos de trocar o banco de dados.

Imagine que necessitamos trocar o banco de dados do MySQL para o Oracle, todas as instruções de API do MySQL teriam de ser trocadas para as chamadas da API do Oracle. No Perl

onde utiliza um método orientado a objetos bastaria adequar um comando:

```
my $con = DBI -> connect("dbi:oracle:$ dbnome", $user, $pass);
```

Na verdade, não utilizaremos um método orientado a objetos. Você poderia se perguntar por que não utilizar esse método. A resposta é “O PHP não é uma linguagem orientada a objetos”, mas possui recursos para orientação a objetos.

Para solucionar esse problema, você pode criar suas próprias classes de conexões ou simplesmente funções genéricas para melhor utilizar as APIs, além de buscar soluções públicas, como as classes PEAR.

Você pode pesquisar mais a PEAR no endereço: <<http://pear.php.net/>>.

Como o mencionado, você pode criar funções ou classes para melhor solucionar o seu problema. Um exemplo de funções seria:

```
<?php  
  
function conect($tipo, $host, $usr, $pass, $banco) {  
  
    GLOBAL $BC = $tipo;  
  
    switch ($BC) {  
  
        case "mysql":  
  
            $res= mysql_connect($host, $usr, $pass);  
  
            mysql_select_db($banco, $res);  
  
        case "oracle":  
  
            $res = oci_connect($usr, $pass, $host);  
  
    }  
  
    return $res;  
}  
  
?>
```

Desta forma, alternando-se o tipo pode-se utilizar diversas APIs com o mesmo comando. Iremos nos aprofundar um pouco mais na API do MySQL mais à frente.

## Conexão

O PHP possui conexões nativas com diversos bancos de dados. O MySQL foi uma das primeiras API a ser desenvolvida. Desta forma, não necessita de uma configuração para acessar o banco. Basta que o endereço do servidor do banco de dados tenha permissão para conexão.

O comando para realizar esse procedimento é o **mysql\_connect**.

```
mysql_connect(servidor,login,senha);
```

Discriminando o comando, temos os seguintes conceitos:

- **servidor**: endereço do servidor do banco de dados (caso seja sua própria máquina, o endereço é localhost ou 127.0.0.1);
- **login**: nome de usuário utilizado para acessar o banco de dados;
- **senha**: senha do usuário que fará a conexão.

O retorno do comando é um identificador para a conexão em caso de sucesso ou *false* em caso de falha. O identificador deve ser armazenado dentro de uma variável PHP para utilizar em outros comandos e diferenciar de uma segunda conexão paralela se existir.

Vamos conectar no banco de dados utilizando o usuário ‘web’. O comando ficaria assim:

```
$con = mysql_connect('localhost','web','senha123');
```

A variável \$con receberá o identificador da conexão, que poderemos utilizar para identificá-la quando necessário. Feita a conexão, devemos selecionar o banco de dados em que deveremos trabalhar. Você pode passar o nome do banco de dados como quarto parâmetro da conexão ou utilizar a função `mysql_select_db()` para isso.

Selecionando a base de dados, ficaria assim:

```
mysql_select_db('prg3',$con);
```

As conexões que são abertas durante o *script* abertas pelo *mysql\_connect* são fechadas quando este *script* termina o processamento. Porém, podem ocorrer situações em que você necessite encerrar a conexão. Para isso, deve-se utilizar o comando *mysql\_connect*.

```
mysql_select_db('prg3',$con);
(...)

mysql_close($con);
```

Vamos então construir uma página onde recuperaremos apenas as informações já cadastradas em um banco de dados. Para isso, criaremos uma tabela no banco de dados que você já deve ter criado. O banco de dados ‘prg3’. Criando a tabela ‘livrarias’:

```
CREATE TABLE prg3.livrarias (
    cod_livraria int(11) NOT NULL AUTO_INCREMENT,
    nom_livraria varchar(50) NOT NULL,
    PRIMARY KEY (cod_livraria ),
    KEY nom_autor (nom_livraria)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Inserir algumas livrarias de exemplo:

```
INSERT INTO livrarias (cod_livraria, nom_livraria) VALUES
(NULL, 'Livraria 1'),
(NULL, 'Livraria 2'),
(NULL, 'Livraria 3'),
(NULL, 'Livraria 4'),
(NULL, 'Livraria 5'),
(NULL, 'Livraria 6');
```

O código para mostrar uma relação de livrarias desse exemplo seria:

```
<html>
<head>
<title>Relação de Livrarias</title>
</head>
<body>
<h1><strong>Relação de Livrarias</strong></h1>

<?php
// a conexão com o servidor bando de dados é realizada
$link = mysql_connect('localhost','web','123');
// o banco de dados é selecionado
mysql_select_db('prg3');

// a consulta é montada e encaminhada ao servidor
$result= mysql_query("select cod_livraria, nom_livraria from livrarias");

?>
<!--
Montamos a tabela em HTML para montar o relatório
-->

<table border="1" cellspacing="0" cellpadding="0" style='border-collapse:collapse; border:none'>
<tr>
<td width="81" valign="top" style="border-top:solid black 1.0pt; border-left:none; border-bottom:solid black 1.0pt; border-right:none; padding:0cm 5.4pt 0cm 5.4pt">
<b>Código</b>
</td>
<td width="579" valign="top" style="border-top:solid black 1.0pt; border-left:none; border-bottom:solid black 1.0pt; border-right:none; padding:0cm 5.4pt 0cm 5.4pt">
<b>Nome</b>
```

```
</td>
</tr>
<?php
// controle utilizado no HTML 4.0 para alternar a cor da tabela do HTML
$fundo="white";
/*
este laço de repetição é executado enquanto existirem registros a serem
recuperados
a função mysql_fetch_array retorna os registro do banco e atribui um
vetor de resultado
*/
while ($relacao = mysql_fetch_array($result)) {
    if ($fundo=="white") {
        $fundo="silver";
    } else {
        $fundo="white";
    }
?>
<tr>
<td width="81" valign="middle" style="border:none; border-
bottom:solid black 1.0pt; background:<?=$fundo?>; padding:0cm 5.4pt
0cm 5.4pt">
<!-- vetor de resposta da relação
-->
<b><?=$relacao["cod_livraria"]; ?></b>
</td>
<td width="579" valign="top" style="border:none; border-bottom:solid
black 1.0pt; background:<?=$fundo?>;padding:0cm 5.4pt 0cm 5.4pt">
<?=$relacao["nom_livraria"]; ?>
</td>
</tr>
<?php
```

```

}
?>
</table>
</body>
</html>

```

O resultado desse código é:

Relação de Livrarias	
Código	Nome
1	Livraria 1
2	Livraria 2
3	Livraria 3
4	Livraria 4
5	Livraria 5
6	Livraria 6

Figura 4.2 – Exposição da query.

Fonte: Elaborada pelo autor (2010).

Nesse exemplo podemos notar que, de forma simples, podemos montar uma consulta ao banco e retornar uma relação de forma a ser interpretada. Perceba que a consulta foi usado os campos, e não utilizao o asterisco.

Quando você usa o asterisco nas consultas SQLs, você força um processamento maior do gerenciador. No nosso exemplo, seria algo insignificante, mas a prática de utilizar sempre os campos é excelente e evita que você se esqueça de utilizar.

Vamos agora incrementar um pouco o exemplo.

Perceba que estamos utilizando a conexão de forma a estar embutido na própria página. Isso deixa exposto o acesso ao nosso banco de dados por terceiros. A forma mais simples de realizar um incremento na segurança de uma página é retirar o servidor, o *login*, a senha e o banco de dados da página. Podemos realizar essa tarefa utilizando o comando include. Esse comando busca um arquivo e interpreta o seu conteúdo como parte da página. Assim, teremos acesso ao servidor, ao *login*, à senha e ao banco de dados sem que esteja presente na página do programa. Vamos construir um arquivo qualquer, como exemplo será

denominado com o nome de variaveis.bd salvo na própria pasta da página. Esse arquivo será texto puro. Não tem função de programação, apenas para que o PHP interprete de forma que possamos utilizar como variáveis, atribuiremos as variáveis como no PHP, mas deverá conter as *tags*.

```
<?php  
$user="web";  
$pass="123";  
$serv="localhost";  
$bd="prg3";  
?>
```

O programa fica desta forma:

```
<?php  
/* com o comando include, todo o texto contido no arquivo passa a fazer  
parte do programa. No nosso caso, como o arquivo possui comando  
válidos no PHP, são incorporados de forma natural.  
*/  
include("variaveis.bd");  
// a conexão com o servidor bando de dados é realizada  
$link = mysql_connect($serv, $user, $pass);  
// o banco de dados é selecionado  
mysql_select_db($bd);  
// a consulta é montada e encaminhada ao servidor  
$result= mysql_query("select cod_livraria, nom_livraria from livrarias");  
?>
```

O restante da página não se altera. Veja que as variáveis que estão dentro do arquivo variáveis.bd são incorporadas de tal forma que possui o mesmo comportamento das variáveis atribuídas na página. Para ficar mais claro, o código a seguir representa esta semelhança.

```
<?php  
/* O comando include não está sendo utilizado, neste momento está  
sendo usada a atribuição direta de variáveis. Mas, neste caso, voltamos  
ao problema inicial das variáveis de segurança estarem abertas na  
página.  
*/  
  
$user="web";  
$pass="123";  
$serv="localhost";  
$bd="prg3";  
  
// a conexão com o servidor bando de dados é realizada  
$link = mysql_connect($serv, $user, $pass);  
  
// o banco de dados é selecionado  
mysql_select_db($bd);  
  
// a consulta é montada e encaminhada ao servidor  
$result= mysql_query("select cod_livraria, nom_livraria from livrarias");  
?>
```

Para aumentar a segurança, você pode armazenar o arquivo com as configurações do servidor de banco de dados fora da pasta de publicação da Web. Ou seja, o arquivo de parametrização não estará acessível pela internet.

## Tratamento de erros

Vamos continuar com o mesmo exemplo e incrementá-lo um pouco mais, acrescentando um tratamento de erros de alguns níveis como mensagem personalizada e o próprio erro do banco de dados.

```
<?php

/* continuando com o comando include

*/
include("variaveis.bd");

/* a conexão com o servidor bando de dados é realizada, caso ocorra
algum problema o programa irá parar e emitir a mensagem: "Erro na
conexão com o banco de dados".

Nenhum outro comando será executado.

*/
$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o
banco de dados");

// vamos selecionar o banco de dados errado o que resultará em um erro

mysql_select_db("bancoerrado") or die ("<BR>Erro na seleção do banco
de dados > ".mysql_errno($link) . ":" . mysql_error($link). "\n<BR>" );

// a consulta é montada e encaminhada ao servidor

$result= mysql_query("select cod_livraria, nom_livraria from livrarias")
or die("Erro na execução da consulta > ". mysql_errno($link) . ":" . mysql_
error($link));

?>
```

A função *mysql\_error* retorna o último erro ocorrido no banco de dados. Já a função *mysql\_errno* retorna o número do erro. Esse número é importante identificar, pois é mais fácil a busca e a identificação da solução do erro. O resultado do exemplo será:

## Relação de Livrarias

Erro na seleção do banco de dados > 1049: Unknown database 'bancoerrado'

Figura 4.3 – Tratamento de erros.

Fonte: Elaborada pelo autor (2010).

Outra forma de montar um tratamento de erro seria a construção de algoritmo para que ocorra um desvio durante a execução. Desta forma, é possível manter um controle maior sobre a aplicação e ainda respostas mais amigáveis ao usuário. Um exemplo desse tipo de tratamento de erro você pode ver a seguir.

```
<?php

/* continuando com o comando include
*/
include("variaveis.bd");

/* a conexão com o servidor bando de dados é realizada, caso ocorra
algum problema o programa irá parar e emitir a mensagem: "Erro na
conexão com o banco de dados".

Nenhum outro comando será executado.

*/
$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o
banco de dados");

// vamos selecionar o banco de dados errado o que resultará em um erro
if (!(mysql_select_db("bancoerrado"))){

    echo "<p>Erro na seleção do banco de dados</p>\n";
    echo "<b>".mysql_errno($link) . ":" . mysql_error($link). "</b>\n<BR>";
} else {

    // a consulta é montada e encaminhada ao servidor
    $result= mysql_query("select cod_livraria, nom_livraria from livrarias");

    ?>
    <!--
    Montamos a tabela em HTML para montar o relatório
    -->
```

```
<table border="1" cellspacing="0" cellpadding="0" style='border-collapse:collapse; border:none'>

<tr>

    <td width="81" valign="top" style="border-top:solid black 1.0pt; border-left:none; border-bottom:solid black 1.0pt; border-right:none; padding:0cm 5.4pt 0cm 5.4pt">

        <b>C&oacute;digo</b>

    </td>

    <td width="579" valign="top" style="border-top:solid black 1.0pt; border-left:none; border-bottom:solid black 1.0pt; border-right:none; padding:0cm 5.4pt 0cm 5.4pt">

        <b>Nome</b>

    </td>

</tr>

<?php

// controle utilizado no HTML 4.0 para alternar a cor da tabela do HTML

$fundo="white";

while ($relacao = mysql_fetch_array($result)) {

    if ($fundo=="white") {

        $fundo="silver";

    } else {

        $fundo="white";

    }

?>

<tr>

    <td width="81" valign="middle" style="border:none; border-bottom:solid black 1.0pt; background:<?=$fundo?>; padding:0cm 5.4pt 0cm 5.4pt">

        <!-- vetor de resposta da rela o

-->

        <b><?=$relacao["cod_livraria"]; ?></b>

    </td>
```

```
<td width="579" valign="top" style="border:none; border-
bottom:solid black 1.0pt; background:<?=$fundo?>;padding:0cm 5.4pt
0cm 5.4pt">

<?=$relacao["nom_livraria"]; ?>

</td>
</tr>
<?php
}
?>
</table>
<?php
}
?>
```

## Seção 5 – Envio e recuperação de dados

Já observamos alguns recursos que a API do PHP com o MySQL disponibiliza para que possamos desenvolver sistemas completos. Porém, essas APIs teriam uma utilização reduzida, sem a interação com as próprias páginas HTML.

Via de regra, as páginas funcionarão com programas independentes um dos outros. Cada requisição ao servidor executarão as instruções e retornarão suas respostas, uma vez que retornar a resposta o programa encerra, ao menos na visão do servidor.

## Formulários para manipulação de dados

Vamos montar agora um *script* simples de edição e pesquisa de duas tabelas do MySQL. Vamos utilizar como base a tabela de livros e autores já mostrados em outros exemplos. Primeiramente montaremos uma página em HTML para funcionar como um menu de operações.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html lang="pt-br">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<head>
<title>Livros e Autores</title>
</head>
<body>
<h1>Livros e Cadastros</h1>
<hr />
<h2>Menu de Operações</h2>
<p><a href="listautor.php" target="_self">Listar Autores</a><br />
<a href="listlivros.php">Listar Livros</a><br />
<a href="pesqregistros.php">Pesquisar Dados</a></p>
</body>
</html>
```

Utilizaremos três operações: Listar Livros, Listar Autores e Pesquisar Dados. Cada *link* apontado para uma página diferente, como resultado fica assim:



Figura 4.4 – Formulando menus.

Fonte: Elaborada pelo autor (2010).

Veja que essa página de menu é apenas HTML, não foi utilizado nenhum comando de PHP ou JavaScript. Lembre-se que é uma escolha para esse exemplo. Nada impede que possamos incrementar a navegação.

Vamos trabalhar agora com a Lista de Autores. Essa página aumenta um pouco a complexidade do desenvolvimento. A ideia é listar todos os registros cadastrados na tabela ‘autores’. Para cada registro listado, o usuário pode alterar ou excluir o registro. Além disso, um *link* para inserir um novo registro.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd"><html lang="pt-br">

<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Autores</title>
</head>
<body>
<h1>Autores</h1>
<hr />
<!--
link para cadastrar o autor
-->
<p><a href="cadautor.php">Cadastrar</a></p>
<?php
// include com as variáveis do banco de dados
include("variaveis.bd");
$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o
banco de dados");
mysql_select_db($bd) or die ("Erro na seleção do banco de dados".
mysql_errno($link) . ":" . mysql_error($link). "\n<br>");
// a consulta é montada e encaminhada ao servidor
$result= mysql_query("select cod_autor, nom_autor from autores");
?>
<table width="40%" border="1">
<tr>
<th scope="col">C&oacute;digo</th>
```

```
<th scope="col">Nome</th>
<th colspan="2" scope="col">Op&ccedil;&otilde;es</th>
</tr>
<?php
// lembre-se que você pode mesclar as TAGs entre HTML e o PHP
// Teste para verificar se relacionou ao menos um registro na consulta
// O '@' é utilizado para inibir qualquer mensagem erro.
if (@mysql_num_rows($result)>=1) {
    // o laço será executado enquanto existir registros
    while ($relacao = mysql_fetch_array($result)) {
        // construção em PHP para manipular o CSS
        if ($fundo=="white") {
            $fundo="silver";
        } else {
            $fundo="white";
        }
        // construido a linha dos resultados da consulta
    ?>
    <tr style="background:<?=$fundo?;>">
    <td><?=$relacao["cod_autor"]; ?></td>
    <td><?=$relacao["nom_autor"]; ?></td>
    <!-- nestes dois links é enviado o código do registro
        assim as páginas que os links estão apontando poderão identificar qual
        registro deve ser manipulado
    -->
    <td><a href="altautor.php?cod=<?=$relacao["cod_autor"]?>">Alterar</a></td>
    <td><a href="excautor.php?cod=<?=$relacao["cod_
        autor"]?>">Excluir</a></td>
    </tr>
<?php
```

```

} } else {

    // caso não seja encontrado registros esta linha é montada

?>

<tr style="background:silver;">

<td>&nbsp;</td>

<td>&nbsp;</td>

<td>&nbsp;</td>

<td>&nbsp;</td>

</tr>

<?php

}

?>

</table>

<p><a href="menu.html">Voltar</a></p>

<p>&nbsp;</p>

<p>&nbsp;</p>

</body>

</html>

```

Veja que nesse código realizamos uma consulta ao banco de dados. Além disso, foi manipulado o HTML para se modificar dependendo de cada resultado. Assim, teríamos a página com os seguintes dados.

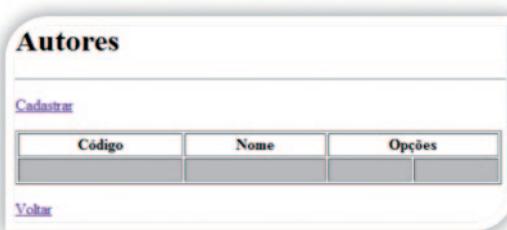


Figura 4.5 – Recuperando dados do MySQL.  
Fonte: Elaborada pelo autor (2010).

Vamos agora montar a página que irá inserir os dados na tabela.

Como anteriormente mostrado, essa tabela de autores está criada com o campo *cod\_autor* sendo *auto\_increment*. Assim, podemos omitir esse campo no cadastro, já que nenhuma informação do usuário deve ser inserida nesse campo. Portanto, teremos apenas um campo que receberá informações do usuário. Outro fator é que será utilizado uma validação de campo em JavaScript.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Cadastrar Autor</title>
<script language="javascript" type="text/javascript" charset="utf-8">
function valida_form(form) {
    // verificando a quantidade de caracteres inseridos no campo 'nom'
    if (form.nom.value.length<4) {
        // se for inferior a quatro caracteres uma mensagem é mostrada
        alert("O nome do Autor deve conter no mínimo 4 caracteres!");
        // retorna falso para o formulário inibindo o seu envio para o
        // servidor. O comando return encerra a função
        return false;
    }
    return true;
}
</script>
</head>
<body>
<?php
// verificação se o formulário foi submetido, ou seja, enviado ao servidor
if (@$_POST["salvar"]!="") {
// include com as variáveis do banco de dados
```

```

include("variaveis.bd");

$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o
banco de dados");

mysql_select_db($bd) or die ("Erro na seleção do banco de dados".
mysql_errno($link) . ":" . mysql_error($link). "\n<br>");

// a inserção de dados é montada e encaminhada ao servidor

$result= mysql_query("insert into autores (nom_autor) values ('".$_
POST["nom"]."')") or die ("Erro ao cadastrar autor");

?>

<script language="javascript" type="text/javascript" charset="utf-8">
// interação do PHP com o JavaScript

alert ("Registro inserido com sucesso! Código gerado: <?=mysql_insert_"
id(?>);

</script>

<?php

}

?>

<h1>Cadastrar Autor</h1>

<hr>

<!--

o atributo onSubmit="return valida_form(this);'' está chamando uma
função JavaScript.

Além disso, o comando return força que o formulário só seja enviado se a
função retornar verdadeiro (true)

-->

<form action="" method="post" enctype="multipart/form-data"
name="frmcad" target="_self" onSubmit="return valida_form(this);">

<table width="40%" border="1">

<tr>

<td width="28%"><label for="nom">Nome</label></td>

<td width="72%"><input type="text" name="nom" id="nom"
value=""></td>

</tr>

```

```
<tr>
<td colspan="2" align="center">
    <input type="submit" name="salvar" id="Cadastrar"
    value="Cadastrar">&nbsp;&nbsp;
    <input type="reset" name="Limpar" id="Limpar" value="Limpar">
</td>
</tr>
</table>
<p><a href="listautor.php">Voltar</a></p>
</form>
</body>
</html>
```

Alguns pontos podem ressaltar nesse exemplo. O primeiro é com relação à validação de campos. A validação efetuada foi realizada apenas em JavaScript. Isso acarreta em baixa segurança quanto às validações de informações. Como já falamos, o JavaScript é uma linguagem *client-side*, isso significa que a interpretação fica a cargo do *browser* do usuário. Portanto, torna-se facilmente manipulável pelo próprio usuário. A forma mais simples é desabilitando a interpretação do JavaScript pelo *browser*. Assim, a validação irá parar de funcionar. Outra forma seria a utilização de algumas ferramentas de desenvolvimento *on-line*, ou seja, pode ser alterada a programação em tempo de execução. Poderia citar o Firebug do Firefox, que permite alterar todas as informações das linguagens *client-side*. Para aumentar a segurança, é necessário utilizar uma validação de dados também na linguagem *Server-side*. Nossa exemplo bastaria acrescentar o campo que deva ser obrigatório acrescido da função *strlen* que conta o número de caracteres em uma *string*:

```
(...)
<?php
// verificação se o formulário foi submetido, ou seja, enviado ao servidor
if ((@$_POST["salvar"]!="") && (strlen(@$_POST["nom"])>3)) {
// include com as variáveis do banco de dados
include("variaveis.bd");

$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com
o banco de dados");
(...)
```

Com essa simples alteração, não será permitida a inclusão de um registro que não tenha pelo menos quatro caracteres no nome. Normalmente quando há uma validação em JavaScript, não é feito um tratamento nas linguagens *service-side*, apenas verifica se foi validado corretamente pelo JavaScript, mas fica a cargo de cada desenvolvedor.

Outro ponto a ser comentado é quanto à integração do PHP com o JavaScript. Veja que no exemplo, após a inserção do registro, é emitida uma mensagem em JavaScript avisando que foi inserido com sucesso e ainda mostrando o código que foi inserido. Esse código pode ser resgatado pela função *mysql\_insert\_id*. Perceba que a interação se dá por meio de escrita, ou seja, o PHP manda escrever alguma informação dentro do código JavaScript. Ele, por sua vez, realiza a interpretação de sua programação sem tomar conhecimento da existência do PHP.

Ainda relacionado ao código, foi utilizado uma programação embutida na própria página do formulário. Esse método requer que seja sempre verificado se o formulário foi enviado ao servidor ou simplesmente a página foi carregada a primeira vez. Como resultado, temos a seguinte página:

A imagem mostra uma interface web com o título "Cadastrar Autor". Abaixo do título, há um formulário com um campo de texto rotulado "Nome" que contém o valor "Jô Soares". Ao lado do campo, há dois botões: "Cadastrar" e "Limpar". Abaixo do formulário, há um link "Voltar".

Figura 4.6 – Cadastrando dados no banco de dados.

Fonte: Elaborada pelo autor (2010).

A mensagem de retorno como:

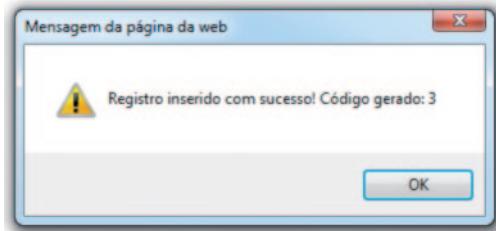


Figura 4.7 – Mensagem de retorno em JavaScript.

Fonte: Elaborada pelo autor (2010).

Após inserir alguns registros, retornaremos a página de listagem, desta vez mostrará os dados inseridos:



Código	Nome	Opções	
1	Fulano de Tal	<a href="#">Alterar</a>	<a href="#">Excluir</a>
3	Jô Soares	<a href="#">Alterar</a>	<a href="#">Excluir</a>
2	Outro Autor	<a href="#">Alterar</a>	<a href="#">Excluir</a>

Figura 4.8 – Listagem completa das informações.

Fonte: Elaborada pelo autor (2010).

Com os dados inseridos, podemos trabalhar nas outras páginas de manipulação (Alterar e Excluir). A primeira a trabalharmos será a página de alteração de dados. A funcionalidade é muito semelhante à página de inserção. Porém, é necessário recuperar a informação e realizar a alteração. Uma parte desse desenvolvimento já está acoplada à página de Listar Autores. Quando o usuário clicar no link Alterar, já está sendo enviado o código do registro que será alterado. Assim, a página de alteração já saberá qual registro deve ser manipulado.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Alterar Autor</title>
<script language="javascript" type="text/javascript" charset="utf-8">
function valida_form(form) {
    if (form.nom.value.length<4) {
        alert("O nome do Autor deve conter no mínimo 4 caracteres!");
        return false;
    }
    return true;
}
</script>
</head>
<body>
<?php
// include com as variáveis do banco de dados
include("variaveis.bd");
// include com as variáveis do banco de dados
$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o
banco de dados");
mysql_select_db($bd) or die ("Erro na seleção do banco de dados".
mysql_errno($link) . ":" . mysql_error($link). "\n<br>");
// Validação de dados e método
if ((@$_POST["cod"]!="") && (strlen(@$_POST["nom"])>3) && (@$_
POST["cod"]==@$_GET["cod"])){
/*
comando de alteração de dados
```

deve-se preferencialmente filtrado pela chave primaria, assim garante que apenas um registro será alterado

```
*/  
  
$result= mysql_query("update autores set nom_autor='".$_  
POST["nom"]." where cod_autor=".$_GET["cod"]) or die ("Erro ao  
cadastrar autor");  
  
?>  
  
<script language="javascript" type="text/javascript" charset="utf-8">  
    alert ("Registro alterado com sucesso! Código alterado: <?=$_  
GET["cod"]?>");  
  
</script>  
  
<?php  
  
}  
  
/* verifica o registro que foi selecionado na página de listagem  
todo a informação mostrada na url é interpretada pelo PHP como um  
envio de informação pelo método GET*/  
  
if (@$_GET["cod"]!="") {  
  
    $result= mysql_query("select cod_autor, nom_autor from autores  
    where cod_autor=".$_GET["cod"]);  
  
    $relacao = mysql_fetch_array($result);  
  
}  
  
?>  
  
<h1>Alterar Autor</h1>  
  
<hr>  
  
<form action="" method="post" enctype="multipart/form-data"  
name="frmcad" target="_self" onSubmit="return valida_form(this);">  
  
<table width="40%" border="1">  
  
<tr>  
  
<td><label for="cod">Código</label></td>  
  
<td>  
  
<!--o atributo readonly deixa o campo somente leitura  
o atributo value carrega um valor padrão para o campo-->
```

```
<input type="text" name="cod" id="cod" value=<?=@$relacao["cod_<br/>autor"]; ?>" readonly="true"></td></tr><tr><td width="28%"><label for="nom">Nome</label></td><td width="72%"><input type="text" name="nom" id="nom" value=<?=@$relacao["nom_autor"]; ?>"></td></tr><tr><td colspan="2" align="center"><input type="submit" name="salvar" id="Alterar" value="Alterar" />&nbsp;&nbsp;<input type="reset" name="Limpar" id="Limpar" value="Limpar" /></td></tr></table><p><a href="listautor.php">Voltar</a></p></form></body></html>
```

Veja que nessa página está sendo realizado um acesso a banco de dados antes do envio das informações do formulário. Isso se faz necessário para que seja realizada uma consulta sobre o código enviado pela página anterior. Desta forma, pode alimentar os campos do formulário para serem alterados.

**Alterar Autor**

Código	3
Nome	Jô Soares
<input type="button" value="Alterar"/> <input type="button" value="Limpar"/>	
<a href="#">Voltar</a>	

Figura 4.9 – Alterando valores.

Fonte: Elaborada pelo autor (2010).

O campo código não pode ser alterado, para isso impossibilitamos a alteração dele no campo. As validações de JavaScript permanecem assim como a resposta da alteração. Então, com esse exemplo, é possível perceber que entre uma página de inclusão e outra de alteração existem pequenas alterações no código, mas que fazem muita diferença no final. Uma delas é a reposição da conexão com o banco de dados. Como ela é necessária nos dois momentos (carregar a páginas e envio para o servidor), é necessário que seja feita fora dos laços de decisão.

Uma validação que aparece nesse exemplo é sobre o método. Ou seja, temos de validar se o código consultado é o mesmo que foi alterado. Podemos fazer isso verificando se código enviado pela url da página é o mesmo utilizado no formulário. Qualquer discrepância dessas informações é em função de o usuário estar tentando burlar o sistema.



Essa forma de segurança utilizada soluciona o problema desse exemplo. Existem diversas formas de aumentar a segurança da aplicação. Uma delas é utilizando chaves criptografadas de verificação. Pesquise sobre o assunto!

A segunda página de manipulação de dados é a exclusão de registro. Essa página não há uma entrada de informações pelo usuário. Para este exemplo, foi adotada uma interação maior com o JavaScript. Uma confirmação de exclusão por meio do JavaScript é feita e somente então é excluído o registro.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Exclui Autor</title>
<script language="javascript" type="text/javascript" charset="utf-8">
function valida_form(form) {
    // verifica se o usuario quer mesmo excluir o usuário
    if (confirm("Deseja Realmente Excluir o Registro?")) {
        return true;
    }
    return false;
}
</script>
</head>
<body>
<?php
// include com as variáveis do banco de dados
include("variaveis.bd");
$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o
banco de dados");
mysql_select_db($bd) or die ("Erro na seleção do banco de dados".
mysql_errno($link) . ":" . mysql_error($link). "\n<br>");
if ((@$_POST["cod"]!="") && (@$_POST["cod"]==@$_GET["cod"])){
    // query para excluir as informações
    $result= mysql_query("delete from autores where cod_autor=".$__
POST["cod"]) or die ("Erro ao cadastrar autor");
?
<script language="javascript" type="text/javascript" charset="utf-8">
```

```
        alert ("Registro excluído com sucesso! Código excluído: <?=$_GET["cod"]?>");

        location.href="listautor.php";

    </script>

<?php

}

if (@$_GET["cod"]!="") {

    $result= mysql_query("select cod_autor, nom_autor from autores
where cod_autor=".$_GET["cod"]);

    $relacao = mysql_fetch_array($result);

}

?>

<h1>Alterar Autor</h1>

<hr>

<form action="" method="post" enctype="multipart/form-data"
name="frmcad" target="_self" onSubmit="return valida_form(this);">

<table width="40%" border="1">

<tr>

<td>C&oacute;digo</td>

<td>

<!--campo oculto para segurança das informações-->

<input type="hidden" name="cod" id="cod"
value="<?=@$relacao["cod_autor"]; ?>">

<?=@$relacao["cod_autor"]; ?></td>

</tr>

<tr>

<td width="28%">Nome</td>

<td width="72%"><?=@$relacao["nom_autor"]; ?></td>

</tr>

<tr>

<td colspan="2" align="center">
```

```

<input type="submit" name="salvar" id="Excluir"
value="Excluir">&nbsp;&nbsp;

<input type="reset" name="Limpar" id="Limpar" value="Limpar">

</td>

</tr>

</table>

<p><a href="listautor.php">Voltar</a></p>

</form>

</body>

</html>

```

Veja que não há uma diferença expressiva entre a exclusão e a alteração de dados. As principais alterações se darão quanto à regra de negócio contida na programação, logicamente, não levando em consideração os aspectos visuais da página.

Todas as páginas partindo dessa solução serão construídas de forma muito semelhante. Pouca alteração da lógica de programação será feita. Porém, pode ainda existir algum insegurança quando se trabalha com relacionamentos. Por causa disso, o exemplo a seguir ilustra a página de alteração de livros. Esta tabela possui um relacionamento com autores. A solução é essencialmente a mesma utilizada para a tabela autores: uma página para listagem dos registros, uma para cadastro, uma para alteração e uma para exclusão.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">

<title>Alterar Livro</title>

<script language="javascript" type="text/javascript" charset="utf-8">

function valida_form(form) {

```

```
if (form.nom.value.length<4) {  
    alert("O nome do Livro deve conter no mínimo 4 caracteres!");  
    return false;  
}  
  
return true;  
}  
  
</script>  
</head>  
<body>  
<?php  
// include com as variáveis do banco de dados  
include("variaveis.bd");  
  
$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o  
banco de dados");  
  
mysql_select_db($bd) or die ("Erro na seleção do banco de dados".  
mysql_errno($link) . ":" . mysql_error($link). "\n<br>");  
  
// valida as informações encaminhadas  
  
if ((@$_POST["cod"]!="") && (strlen(@$_POST["nom"])>3) && (@$_  
POST["cod"]==@$_GET["cod"]) && (@$_POST["cod_autor"]!="")) {  
  
// adequação das informações em branco para nulo  
  
if (@$_POST["cod_autor"]!="") {  
    $cod_autor=$_POST["cod_autor"];  
}  
else {  
    $cod_autor="NULL";  
}  
  
$result= mysql_query("update livros set nom_livro=".$_POST["nom"].",  
cod_autor=".$_POST["cod_autor"]." where cod_livro=".$_POST["cod"]) or  
die ("Erro ao cadastrar livro");  
?  
<script language="javascript" type="text/javascript" charset="utf-8">  
alert ("Registro alterado com sucesso! Código gerado: <?=$_  
POST["cod"]?>");
```

```
</script>
<?php
}

if (@$_GET["cod"]!="") {

    $result= mysql_query("select cod_livro, nom_livro, cod_autor from
livros where cod_livro=".$_GET["cod"]);

    $relacao = mysql_fetch_array($result);

}

?>

<h1>Alterar Livro</h1>

<hr>

<form action="" method="post" enctype="multipart/form-data"
name="frmcad" target="_self" onSubmit="return valida_form(this);">

<table width="40%" border="1">

<tr>

    <td width="28%"><label for="cod">C&oacute;digo</label></td>

    <td width="72%"><input type="text" name="cod" id="cod"
value=<?=$relacao["cod_livro"]?>" readonly="true"></td>

</tr>

<tr>

    <td width="28%"><label for="nom">Nome</label></td>

    <td width="72%"><input type="text" name="nom" id="nom"
value=<?=$relacao["nom_livro"]; ?>"></td>

</tr>

<tr>

    <td width="28%"><label for="cod_autor">Autor</label></td>

    <td width="72%">

        <select name="cod_autor">

            <option value="">-----</option>

        <?php

            //consulta para montar as informações de relacionamento
```

```
$result_autores= mysql_query("select cod_autor, nom_autor from
autores");

while ($relacao_autores = mysql_fetch_array($result_autores)) {
    //seleciona o autor cadastrado para o livro
    if ($relacao_autores["cod_autor"]==$relacao["cod_autor"]) {
        $select="selected";
    } else {
        $select="";
    }
}

?>

<option value=<?=$relacao_autores["cod_autor"]?>"<?=$select?>><?=$relacao_autores["nom_autor"]; ?></option>

<?php
}

?>

</select>

</td>

</tr>

<tr>

<td colspan="2" align="center">
    <input type="submit" name="salvar" id="Alterar"
value="Alterar">&nbsp;&nbsp;
    <input type="reset" name="Limpar" id="Limpar" value="Limpar">
</td>
</tr>

</table>

<p><a href="listlivros.php">Voltar</a></p>

</form>

</body>

</html>
```

Apesar de existir o relacionamento entre as tabelas, a solução nesse caso foi de compor as consultas para cada campo, ou seja, uma consulta que recupera a informação de livros e outra que recupera as informações de autores, selecionando o autor cadastrado no livro. Essa solução é mais bem empregada para o uso do campo *combox* do formulário. O restante do código se torna muito semelhante à página de alterar autores.



Toda a forma de manipulação de dados é sempre a mesma: incluir, alterar, excluir. Quando se define uma solução para essa manipulação, uma página ela tende a se propagar para todo o sistema. O cuidado é para as informações ou detalhes específicas de cada página.

O resultado desse código é uma página com três campos: código, nome do livro e autor.

Código	2
Nome	Completo MySQL
Autor	Outro Autor

[Alterar](#) [Limpar](#)

[Voltar](#)

Figura 4.10 – Alterando no relacionamento entre tabelas.

Fonte: Elaborada pelo autor (2010).



Procure desenvolver as outras páginas (listar, inclusão e exclusão) dos registros de livros.

Vamos construir agora uma página de consulta. Esta página consiste em o usuário informar algum dado e o sistema procurar as informações e listá-las na tela. Para aumentar um pouco a complexidade, vamos realizar a consulta sobre as duas tabelas (autores e livros). Para os registros encontrados, faremos um *link* apontando para a página de alteração de informações. Desta forma, maximizaremos a utilidade da consulta.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Alterar Livro</title>
<script language="javascript" type="text/javascript" charset="utf-8">
function valida_form(form) {
    // teste para obrigar um caracter em algum dos campos
    if ((form.nom_livro.value.length<1) && (form.nom_autor.value.length<1)) {
        alert("O nome do Livro deve conter no mínimo 1 caracter!\n Ou\o
nO nome do Autor deve conter no mínimo 1 caracter!");
        return false;
    }
    return true;
}
</script>
</head>
<body>
<?php
if (@$_POST["pesquisar"]!="") {
    include("variaveis.bd");
    $link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o
banco de dados");
    mysql_select_db($bd) or die ("Erro na seleção do banco de dados".
mysql_errno($link) . ":" . mysql_error($link). "\n<br>");
    // esta variável tem a função de montar o filtro da consulta
    $pesquisa="";
    if (@$_POST["nom_livro"]!="") {
        // primeiro filtro é montado se existir algum caracter
```

```

$pesquisa="nom_livro like '%".$_POST["nom_livro"]."%'";
}

if (@$_POST["nom_autor"]!="") {

    // verifica se o primeiro filtro foi usado. Caso sim, acrescenta a
    // condição AND (E) para que o segundo filtro seja utilizado

    if ($pesquisa!="") { $pesquisa.=" and "; }

    // segundo filtro é montado se existir algum caracter

    $pesquisa.="nom_autor like '%".$_POST["nom_autor"]."%'";
}

// consulta é montada em uma variável para facilitar o desenvolvimento

$query="select a.cod_livro, a.nom_livro, b.cod_autor, b.nom_autor from
livros a, autores b where a.cod_autor=b.cod_autor and ".$pesquisa;

// consulta é executada

$result= mysql_query($query);

}

?>

<h1>Pesquisar Registros</h1>

<hr>

<form action="" method="post" enctype="multipart/form-data"
name="frmcad" target="_self" onSubmit="return valida_form(this);">

<table width="40%" border="1">

<tr>

    <td width="28%"><label for="nom_livro">Livro</label></td>

    <td width="72%"><input type="text" name="nom_livro" id="nom_
livro" value=""></td>

</tr>

<tr>

    <td width="28%"><label for="nom_autor">Autor</label></td>

    <td width="72%"><input type="text" name="nom_autor" id="nom_
autor" value="">

</td>

</tr>

</table>

```

```
<tr>
    <td colspan="2" align="center">
        <input type="submit" name="pesquisar" id="pesquisar"
        value="Pesquisar">&nbsp;&nbsp;
        <input type="reset" name="Limpar" id="Limpar" value="Limpar">
    </td>
</tr>
</table>
</form>
<hr>
<table width="54%" height="82" border="1">
    <tr>
        <th scope="col">C&acute;digo Livro</th>
        <th scope="col">Nome Livro</th>
        <th scope="col">C&acute;digo Autor</th>
        <th scope="col">Nome Autor</th>
    </tr>
<?php
// 
if (@mysql_num_rows($result)>=1) {
    $fundo="white";
    while ($relacao = mysql_fetch_array($result)) {
        if ($fundo=="white") {
            $fundo="silver";
        } else {
            $fundo="white";
        }
    }
?>
<tr style="background:<?=$fundo?;>">
    <td><label title="Alterar os dados deste livro"><a href="altilivros.
php?cod=<?=$relacao["cod_livro"]; ?>"><?=$relacao["cod_livro"]; ?>
```

```
</a></label></td>

<td><?=$relacao["nom_livro"]; ?></td>
<td><label title="Alterar os dados deste Autor"><a href="altautor.php?cod=<?=$relacao["cod_autor"]; ?>"><?=$relacao["cod_autor"]; ?>
</a></label></td>
<td><?=$relacao["nom_autor"]; ?></td>
</tr>
<?php
} } else {
?>
<tr style="background:silver;">
<td colspan="4">
<?php
// caso não encontre informações, mostra uma mensagem
if (@$_POST["pesquisar"]!="") echo "Não foram encontrado registros
na pesquisa.";
else echo " ";
?>
</td>
</tr>
<?php
}
?>
</table>
<p><a href="menu.html">Voltar</a></p>
</body>
</html>
```

Como já mencionado, o controle da página é semelhante às outras páginas do sistema. A diferença está na construção da consulta de forma dinâmica. Conforme a combinação de

informações digitadas no campo, a consulta se altera. Desta forma, aumenta a complexidade do desenvolvimento, mas em contrapartida a satisfação do usuário também é ampliada. O usuário não precisa ficar navegando para filtrar várias informações.

Código Livro	Nome Livro	Código Autor	Nome Autor
1	Ajax	2	Paulo Coelho
1	Completo PHP	3	Paulo Coelho

Figura 4.11 – Pesquisa de informações.  
Fonte: Elaborada pelo autor (2010).

Outro atrativo da página de consulta é a possibilidade de se realizar alterações dos registros. Caso o usuário clique no *link* do código do livro, será encaminhado à página de alteração de livros. Assim, como se clicar no código do autor, será encaminhado à página de alteração do autor.

Nesse simples exemplo, montamos um sistema completo em todas as suas principais características: cadastros, manipulação de dados, recuperação de informações. A maioria dos sistemas comerciais se baseia nessas operações. Como você pôde ver e comprovar, o desenvolvimento pode se tornar extenso, porém muito simples e intuitivo. O importante é o desenvolvedor se concentrar no problema para entendê-lo e posteriormente desenvolver uma solução.

## Seção 6 – Sessões e *cookies*

Você já entendeu as operações envolvidas em um desenvolvimento de um sistema. Ainda assim algumas perguntas ficaram no ar. Como posso identificar se o usuário já visitou a minha página? Como manter informações? Para respondermos essas perguntas, precisamos entender um pouco mais sobre o protocolo HTTP.

O protocolo HTTP não possui informação de estado, ou seja, não possui formas predefinidas de se dar continuidade entre as transações. Quando o usuário navega entre as páginas, o HTPP não possui formas de se identificar se a navegação se deu pelo mesmo usuário. Uma necessidade fundamental para uma segurança em sistema Web é conseguir controlar a sessão dos usuários. Em outras palavras, conseguir monitorar um usuário durante uma única sessão em um *site* Web.

Todas as sessões no PHP se baseiam em um registro único de sessão chamado de ID. Esse ID é uma numeração aleatória e criptografada. Esta geração é realizada pelo PHP e armazenado no cliente para o tempo de vida de uma sessão. As formas de manter esse ID é por meio de *cookies* no computador do usuário ou enviado com a URL.

Muito provavelmente você já deve ter utilizado *sites* que mantinham sessões para identificar e monitorar a sua navegação. O ID da sessão funciona como uma chave que dá acesso às variáveis privadas da sua navegação. Essas variáveis são armazenadas no servidor e somente pode ser acessadas quando o mesmo ID é transmitido para o servidor. O ID é a única informação visível no cliente.

### ***Cookie***

*Cookie* é um arquivo texto que contém uma pequena parte de informações que os programas armazenam. O *cookie* basicamente possui nome, valor, tempo de vida, caminho no servidor, domínio, protocolo de segurança.

No PHP para um *cookie* ser criado usamos a função *setcookie()*. As informações do *cookie* fazem parte do cabeçalho de uma página. Quando o usuário faz uma requisição, a página vem dividida em duas partes, o cabeçalho, que contém as informações sobre a requisição e o conteúdo, que é onde é a saída HTML do processamento. Acabam se tornando muito comuns os erros de cabeçalho quando tentamos alterar essas informações após ser enviado o conteúdo da página (códigos HTML). Todas as informações de cabeçalho necessitam ser passadas antes de qualquer conteúdo. Por esse motivo, é necessário ter cuidado para usar o *setcookie* antes de qualquer saída HTML. Essa saída pode ser uma *tag* ou um simples espaço que está no começo do arquivo antes da abertura da tag <?php.

```
<?php  
//teste para verificar se o formulário foi enviado  
if (@$_POST["nome"]!="") {  
    //criação do cookie  
    setcookie("primeirocookie", $_POST["nome"], time() + 3600);  
}  
?  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html lang="pt-br">  
    <head>  
        <meta http-equiv="Content-Type" content="text/html;  
charset=iso-8859-1">  
        <title>Grava Cookie</title>  
    </head>  
    <body>  
        <form name="form1" method="post" action="">  
            <label>Nome  
                <input type="text" name="nome" id="nome">  
            </label>  
            <br> <input type="submit" name="enviar" value="Enviar">  
    </body>
```

```

</form>

<p><a href="leituracookie.php">Ler Cookie</a><br>
<a href="apagarcookie.php">Apagar Cookie </a></p>

</body>

</html>

```

Como já mencionado, a função *setcookie* aceita vários parâmetros, sendo os mais importantes os três primeiros:

- o nome do *cookie* (primeirocookie)
- o valor do *cookie* (`$_POST[nome]`)
- tempo de expiração do *cookie* (`time() + 3600` é um cálculo que retorna o tempo no timestamp unix, que é a quantidade de segundos que passou desde o dia 01/01/1970)

O tempo de expiração determina para o *browser* até quando ele deve usar aquele *cookie*. O cálculo é em segundos, portanto a função *time* foi utilizada para retornar a hora em segundos neste momento. Adicionado 3600 segundos, o que quer dizer que o *cookie* será válido por mais uma hora.




---

Existem vários parâmetros que podem ser manipulados. Pesquise mais sobre o comando no manual <[http://br.php.net/manual/pt\\_BR/function.setcookie.php](http://br.php.net/manual/pt_BR/function.setcookie.php)>.

---

Para ler um *cookie*, é muito simples. Todos os *cookies* enviados para a página estarão disponíveis na variável `$_COOKIE`. Mas um *cookie* só estará disponível na próxima vez que você acessar uma página. Por isso, vamos criar uma página chamada para realizar essa leitura:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html lang="pt-br">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Leitura do Cookie</title>
</head>

<body>
<p>Lendo o cookie</p>
<p>
<?php
// verifica a existencia do cookie
if (@$_COOKIE["primeirocookie"]!="") {
    //escreve o valor do cookie
    echo "Leitura Cookie: " . $_COOKIE["primeirocookie"];
} else {
    echo "Não foi encontrado Cookie";
}
?>
</p>
<p><a href="gravarcookie.php">Criar Cookie</a><br>
<a href="apagarcookie.php">Apagar Cookie </a></p>
</body>
</html>
```

Veja que dentro de uma hora, toda vez que você carregar essa página, retornará sempre o mesmo valor, a não ser que você altere o valor do *cookie*.

Para apagar um *cookie*, também utilizamos a função *setcookie*, porém temos de passar uma *string* vazia “ ou false, e no tempo de expiração temos de passar uma hora atrasada.

```
<?php

    // passando o valor de vazio e atrasando uma hora da hora atual
    setcookie("primeirocookie", "", time() - 3600);

?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html lang="pt-br">
    <head>
        <meta http-equiv="Content-Type" content="text/html;
        charset=iso-8859-1">
        <title>Apagar Cookie</title>
    </head>

    <body>
        <p>Cookie Apagado</p>
        <p><a href="gravarcookie.php">Criar Cookie</a></p>
    </body>
</html>
```

Veja que o resultado é uma exclusão completa do *cookie*.

Os *cookies* possuem alguns problemas como o navegador não aceitá-los. Hoje é muito difícil encontrar um navegador que não aceite um *cookie*. Porém, pode estar configurado para não aceitar. Portanto, tenha cuidado na sua utilização.

## Sessão

As sessões em PHP possuem alguns passos simples de serem utilizados:

1. Iniciar uma sessão.
2. Registrar variáveis de sessão.
3. Utilizar essas variáveis.
4. Remover registros de variáveis e destruir a sessão.

Esse passos não precisam estar todos em uma mesma página, mas devem estar dentro da sua aplicação. Vamos então montar um exemplo em que montaremos uma sessão semelhante aos exemplos do *cookie*.

```
<?php  
  
//iniciar a sessão  
session_start();  
  
if (@$_POST["nome"]!="") {  
    //criação da variável  
    $_SESSION["primeirasessao"]=$_POST["nome"];  
}  
  
?>  
  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
  
<html lang="pt-br">  
  <head>  
    <meta http-equiv="Content-Type" content="text/html;  
    charset=iso-8859-1">  
    <title>Grava Sess&atilde;o</title>  
  </head>  
  <body>  
    <form name="form1" method="post" action="">
```

```

<label>Nome
<input type="text" name="nome" id="nome">
</label>
<br> <input type="submit" name="enviar" value="Enviar">
</form>
<p><a href="leiturasessao.php">Ler Sessão</a><br>
<a href="apagarsessao.php">Apagar Sessão </a></p>
</body>
</html>

```

Perceba que a sessão não possui tantos parâmetros quanto os *cookies*. Basicamente temos de iniciar a sessão e registrar as variáveis. As variáveis de sessão são acessadas pelo *array* (vetor) *\$\_SESSION*. Vamos agora montar a leitura desta sessão.

```

<?php
//toda a página que utiliza uma variável de sessão deve ser iniciada com
session_start()

session_start();
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html lang="pt-br">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Leitura da Sessão</title>
</head>
<body>
<p>Lendo a Sessão</p>
<p>
<?php
// verifica a existência do cookie

```

```
if (@$_SESSION["primeirasessao"]!="") {  
    //escreve o valor do cookie  
    echo "Leitura Sessão: " . $_SESSION["primeirasessao"];  
} else {  
    echo "Não foi encontrada Sessão";  
}  
?  
</p>  
<p><a href="gravarsessao.php">Criar Sessão</a><br>  
<a href="apagarsessao.php">Apagar Sessão </a></p>  
</body>  
</html>
```

Veja que a primeira instrução da página deve ser sempre o *session\_start*. Somente assim é possível acessar as variáveis de sessão registradas no servidor. Para apagar, vejamos o exemplo a seguir.

```
<?php  
    session_start();  
    // destruindo todas as variáveis de sessão do ID atual  
    session_destroy();  
?  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html lang="pt-br">  
<head>  
<meta http-equiv="Content-Type" content="text/html;  
charset=iso-8859-1">  
<title>Apagar Sessão</title>  
</head>  
<body>
```

```

<p>Sessão Apagada</p>
<p><a href="gravarsessao.php">Criar Sessão</a></p>
</body>
</html>

```

Mesmo que você deseje somente excluir ou destruir as variáveis de sessão é necessário antes iniciá-la para que o servidor possa identificar seu uso.

Já vimos *sessões* e *cookies*, mas, afinal, como utilizamos isso em um sistema de segurança?

A resposta é que como monitoramos a sessão do usuário é possível saber se ele realizou uma autenticação com o servidor e se ela foi realizada com sucesso. Antes de realizar esta tarefa, é necessário criar uma tabela de usuários.

```

CREATE TABLE prg3.usuario (
    login VARCHAR( 10 ) NOT NULL ,
    senha VARCHAR( 32 ) NOT NULL ,
    PRIMARY KEY ( login ) ,
    INDEX ( senha )
) ENGINE = MYISAM ;

```

Além da criação da tabela, é necessário cadastrar o primeiro usuário para se dar acesso ao sistema.

```
INSERT INTO usuario ( login, senha ) VALUES ('teste', MD5( 1 ));
```

Agora já está criada a base de dados necessária para se manter os usuários do sistema. O próximo passo é criar uma tela de *login* no sistema. Essa página deve validar as informações do usuário com o banco de dados.

```
<?php

// inicia a sessão
session_start();

// verifica se o formulário foi enviado

if ((@$_POST["login"]!="") && (@$_POST["senha"]!="")) {

    // carrega informações de banco
    include("variaveis.bd");

    $link = mysql_connect($serv, $user, $pass) or die("Erro na conexão
com o banco de dados");

    mysql_select_db($bd) or die ("Erro na seleção do banco de dados".
mysql_errno($link) . ":" . mysql_error($link). "\n<br>");

    // realiza a busca na tabela. A função md5 realiza a criptografia da
informação

    $result= mysql_query("select login from usuario where login=". $_
POST["login"]." and senha=".md5($_POST["senha"])."");

    if (@mysql_num_rows($result)==1) {

        // registra a sessão
        $_SESSION["login"]=$_POST["login"];
        header('Location: menu.php');

    } else {

        $msg="Login ou Senha n&atilde;o conferem!!";
    }
}

?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html lang="pt-br">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Login</title>
</head>
```

```
<body>

<h1>Login no Sistema </h1>

<hr>

<?php

if (@$msg!="") {

    echo "<h1 style='color:#F00'>$msg</h1>\n";
    echo "<hr>";

}

?>

<form name="form1" method="post" action="">

<table width="40%" border="1">

<tr>

<td>Login</td>

<td><label>

<input type="text" name="login" id="login">

</label></td>

</tr>

<tr>

<td>Senha</td>

<td><label>

<input type="password" name="senha" id="senha">

</label></td>

</tr>

<tr>

<td colspan="2" align="center"><label>

<input type="submit" name="enviar" id="enviar"
value="Enviar">&nbsp;&nbsp;

<input type="reset" name="limpar" id="limpar" value="Limpar">

</label></td>

</tr>
```

```
</table>
</form>
</body>
</html>
```

Com essa página de *login*, foi verificado *login* a senha do usuário. Se for validado, é registrada uma variável de sessão identificada como ‘login’. Após isso, redirecionaremos a página para o menu.php, que nada mais é do que a página de menu dos cadastros de livros e autores explicada em outro exemplo. Essa página recebe uma pequena alteração no código. Acrescentando na primeira linha do código da página a seguinte *tag* PHP:

```
<?php
include("logado.php");
?>
```

Vamos incluir um *script* de validação de sessão. Essa forma é feita para facilitar o desenvolvimento. Uma vez montado, basta acrescentar em todas as páginas do sistema que funcionará perfeitamente. Vamos então montar o *script* para validar a sessão:

```
<?php
// Iniciando a sessão
session_start();
# Se tiver sessão de usuário registrada...
if (@$_SESSION["login"]=="")
{
    // Mensagem de erro
    $mensagem = 'Você não efetuou o login.';
    // redireciona a navegação para a página de erro
    header('Location: erro.php?msg='.$mensagem);
    exit();
?>
```

Bem simples, foi verificado se a variável de sessão existe. Caso não, redireciona para uma página de erros já customizando a mensagem de erro. Essa página deverá ter um código assim:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html lang="pt-br">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=iso-8859-1">
    <title>Erro</title>
  </head>
  <body>
    <h1>Aten&ccedil;&atilde;o</h1>
    <hr />
    <h1 style="color:#F00"><?php
      echo addslashes(mb_convert_encoding(@$_GET["msg"],"iso-8859-
      1","utf-8"));
    ?>
    </h1>
    <hr />
    <p><a href="login.php">Login</a></p>
  </body>
</html>
```

Por último, o sistema deve ter uma página de *logout*, ou seja, de saída do sistema. Para maximizar, o uso vamos utilizar um redirecionamento para a página de erro, customizando a mensagem para a saída do sistema.

```
<?php  
// Inicia a sessão  
session_start();  
  
// Exclui os dados de login  
unset($_SESSION["login"]);  
  
// Elimina a sessão  
session_destroy();  
  
// Enviando o usuário pra página de login  
header('Location: erro.php?msg=Logout efetuado com sucesso!');  
?>
```



## Síntese

Nesta unidade você pôde acompanhar o desenvolvimento de um sistema completo, acessando um banco de dados, manipulando informações e ainda com um certo grau de segurança. Você aprendeu sobre MySQL e suas excentricidades e por que ele é amplamente utilizado em desenvolvimento Web. Viu ainda que com a utilização de uma boa solução pode simplificar o desenvolvimento do sistema.

Aprendeu como o controle de sessão é fundamental para que possamos manter a segurança da aplicação e abre novos horizontes para uma customização da própria aplicação para cada usuário, já que sendo possível identificá-lo torna-se possível criar possibilidades onde possa tornar a aplicação mais amigável conforme o usuário.



## Atividades de autoavaliação

- Desenvolva um cadastro de livros utilizando o MySQL e o PHP. Este cadastro deve conter uma tabela de livros contendo os seguintes campos:

Código

Nome

Autor

Editora

Número de páginas

Ainda deve conter uma página de listagem geral deste cadastro e uma página com uma busca por nome. Estas páginas devem ser navegáveis entre si, ou seja, construa um menu de navegação entre as páginas.

- Monte uma página de *login* (*login* e senha) para acessar uma aplicação fictícia. Esta página de *login* deve verificar no banco de dados se o usuário está cadastrado ou não. Caso esteja, registre uma sessão para o usuário e permita que ele cadastre outros usuários. Caso não esteja, não permitir o acesso a página de cadastro por meio da sessão (não registrando a sessão).



## Saiba mais

Se você desejar, aprofunde os conteúdos estudados nesta unidade ao consultar as seguintes referências:

Manual PHP, 2010. Disponível em: <[http://br2.php.net/manual/pt\\_BR/index.php](http://br2.php.net/manual/pt_BR/index.php)>. Acesso em: 5 abr. 2010.

PHP Brasil, 2010. Disponível em: <[www.phpbrasil.com](http://www.phpbrasil.com)>. Acesso em: 5 abr. 2010.

T r u q u e s e D i c a s – Tutoriais de Javascript, 2010.  
Disponível em: <[www.truquesedicas.com/tutoriais/php/index.htm](http://www.truquesedicas.com/tutoriais/php/index.htm)>. Acesso em: 2 mar. 2010.

Manual do MySQL, 2010. Disponível em: <<http://dev.mysql.com/doc/refman/5.1/en/index.html>>. Acesso em: 5 abr. 2010.



# Extensible Markup Language (XML)



## Objetivos de aprendizagem

Ao final desta unidade, você terá subsídios para:

- Identificar as características do desenvolvimento Web.
- Conhecer métodos de comunicação entre as linguagens.
- Saber as definições de elementos.
- Conhecer as validações de documentos.



## Seções de estudo

**Seção 1** Extensible Markup Language (XML)

**Seção 2** Definição de Tipos de Documentos (DTD)

**Seção 3** XML-Schema

**Seção 4** RSS



## Para início de estudo

Atualmente, o XML é tratado como um padrão de integração de informações em diversos níveis. Sua capacidade de estruturação das informações abre um leque gigantesco de opções que vem sendo cada vez mais explorados. Hoje, bem mais do que um padrão de integração, tornou-se ferramenta indispensável para quem necessita estar sempre informado.

## Seção 1 – Extensible Markup Language (XML)

*Extensible Markup Language* (XML) é linguagem de marcação de dados (*meta-markup language*) que provê um formato para descrever dados estruturados. Isso facilita declarações mais precisas do conteúdo e resultados mais significativos de busca por meio de múltiplas plataformas. Além disso, XML é uma linguagem de transmissão de informações pela internet.



---

XML é uma linguagem de marcação de dados extensível. Ela provê um formato para descrever dados estruturados que facilita declarações mais precisas do conteúdo.

---

**O XML permite a definição de um número infinito de *tags* ou elementos.** Essa característica é muito importante já que o volume de informações tende ao crescimento imensurável. As *tags* no HTML podem ser usadas para a definição de formatação dos caracteres e parágrafos, o XML utiliza as *tags* para uma única finalidade: **estruturar os dados**.

O XML tem como fundamento a descrição de informações. Essa característica a torna essencial para o armazenamento, a recuperação e a transmissão de informações. O grande diferencial dessa linguagem é de permitir armazenar em um mesmo agrupamento os dados e os metadados, que nada mais são do

que um detalhamento dos dados. O seu formato é textual, que nos permite uma leitura mais simples das informações nele formatadas.

Podemos dizer que é uma linguagem inteligente, pois se adéqua facilmente a qualquer nível de complexidade. Em uma complexidade simples, seria:

```
<?xml version="1.0" encoding="utf-8"?>
<AUTOMOVEL>Carro</AUTOMÓVEL>
```

Agora, se for necessário aumentar a complexidade, basta acrescentar *tags* que possam ser utilizadas para os metadados:

```
<?xml version="1.0" encoding="utf-8"?>
<AUTOMOVEL>
  <TIPO>Carro</TIPO>
  <FABRICANTE>Ferrari</FABRICANTE>
  <MODELO ANO="01/01/2009">F-50</MODELO>
</AUTOMÓVEL>
```

Como se pode observar, o número de informações se alterou, porém essa alteração é extremamente simples e facilmente absorvida pelo XML.

Outra vantagem dessa linguagem é a **portabilidade**.

Quando se utiliza arquivos binários para armazenamento de dados, eles podem ser lidos somente por aplicações específicas que foram previamente desenvolvidas para extrair as informações contidas. Grande parte dessas aplicações são proprietárias – o que força, na maioria das vezes, realizar uma engenharia reversa para que se possa extrair as informações por outro aplicativo.



Um exemplo claro é o editor de texto da Microsoft, o Word. Você digita todo o seu documento e monta as informações e formata da forma que achar melhor. Quando você salva o documento, há duas opções: salvar no formato Word ou escolher uma das diversas opções que a ferramenta disponibiliza. Supondo que

você salve no formato padrão, você leva o documento para outro computador que não possui o Word. Você dificilmente conseguirá abrir esse documento. Não somente essa ferramenta, mas diversos softwares proprietários assumem que sempre você terá um software compatível com padrão estabelecido por ele.

Com o XML esse problema não ocorre. Você pode inserir diversas *tags* para caracterizar inúmeros dados. Porém, o *software* que lerá este XML, apenas interpretará o que ele estiver programado para utilizar. Os dados que não identificar são ignorados, não interferindo na composição da informação. Também não se faz necessário um escopo de delimitação, ou seja, uma legenda para conseguirmos identificar onde determinado dado inicia e onde termina.

Uma das formas comuns de transmissão de dados é por meio de arquivos de largura fixa. Esses arquivos consistem em ter as informações em um determinado tamanho. Não podendo extrapolar o limite tampouco reduzir o espaço reservado para a informação. Para uma importação de clientes, teríamos a seguinte tabela:

Coluna	Posição Inicial	Tamanho
CPF	1	11
NOME	12	30
ENDEREÇO	41	50

Assim, teríamos o seguinte registro:

0001110011JOAQUIM DA SILVA                  RUA BRASIL, NUMERO 101

Torna-se clara a limitação dessa técnica: caso um nome extrapole as 30 posições ou ainda o endereço, a única alternativa é cortar a informação, isto é, omitir os caracteres que extrapolarem o tamanho definido.

Outro problema que o arquivo de largura fixa ainda possui é a inclusão de um caractere. Imagine o problema se a Receita Federal resolve aumentar o número de dígitos do CPF? Passar de 11 para 12 algarismos, com certeza seria o caos para milhares de sistemas com esse tipo de integração. Com o XML, esse problema não existe:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<CLIIENTE>
<CPF>00011110011</CPF>
<NOME>JOAQUIM DA SILVA</NOME>
<ENDERECO>RUA BRASIL, NUMERO 101</ENDERECO>
</CLIIENTE>
```

Tratando-se de uma estrutura simplificada, é possível que seja realizada uma validação no arquivo antes de ser iniciado o processamento. Comparativamente, seria como você abrir uma planilha eletrônica no bloco de notas. Fica completamente impossível identificar o seu conteúdo, porém foi gasto um tempo preciso para abrir um arquivo em um *software* que não será possível manipular.

O padrão XML definido pela W3C possui as seguintes especificações:

- *Extensible Markup Language* (XML) 1.0: define a sintaxe da XML.
- *XML Pointer Language* (XPointer) e *XML Linking Language* (XLink): definem um padrão para representar os *links* entre os recursos. Além dos *links* simples, como a tag <A> da HTML, a XML possui mecanismos para ligar recursos múltiplos e diferentes. A XPointer descreve como endereçar um recurso, e a XLink descreve como associar dois ou mais recursos.
- *Extensible Style Language* (XSL): define a linguagem de folhas de estilos padrão para a XML.

Como você pode perceber o XML começa com uma instrução de processamento: <?xml ...?>. Essa é a declaração XML. Embora não seja obrigatória, a sua presença explícita identifica o documento como um documento XML e indica a versão da XML com a qual ele foi escrito.

Não há declaração do tipo do documento. Diferentemente da SGML, a XML não requer uma declaração de tipo de documento. Entretanto, uma declaração de tipo de documento

pode ser fornecida. Além disso, alguns documentos precisarão de uma para serem entendidos sem ambiguidade.

Exemplos de utilização dos elementos vazios:

```
<?xml version="1.0" encoding="iso-8859-1"?>  
<CLIIENTE>  
<CPF>00011110011</CPF>  
<NOME>JOAQUIM DA SILVA</NOME>  
<ENDERECO/>  
</CLIIENTE>
```

O Elemento `<ENDERECO/>` tem uma sintaxe modificada. Enquanto que as maiorias dos elementos em um documento envolvem algum conteúdo, **elementos vazios são simplesmente marcadores onde alguma coisa ocorre** (pode se comparar a *tag* `<hr>` do HTML que se trata de um separador).

O final `/>` na sintaxe modificada indica a um programa que processa o documento XML que o elemento é vazio e uma marca de fim correspondente não deve ser procurada. Visto que os documentos XML não requerem uma declaração de tipo de documento, sem essa pista seria impossível para um analisador XML determinar quais marcas são intencionalmente vazias e quais teriam sido deixadas vazias por um erro.

A XML suavizou a distinção entre elementos declarados como vazios e os que não possuem conteúdo. A linguagem permite usar uma marca de elemento vazio para qualquer um desses casos. Também é válido usar um par de marcas início-fim para elementos vazios: `< ENDERECO ></ ENDERECO >`.

Os documentos XML são compostos de marcas e conteúdos. Existem seis tipos de marcações que podem ocorrer em um documento XML:

- a) elementos;
- b) referências a entidades;
- c) comentários;

- d) instruções de processamento;
- e) sessões marcadas;
- f) declarações de tipos de documento.

## a. Elementos

Os elementos são as *tags* mais comuns na forma de marcação. Eles são delimitados pelos sinais de menor e maior ('<' e '>'), a maioria dos elementos identificam a natureza do conteúdo que a envolvem. Alguns elementos podem ser vazios, como visto anteriormente; neste caso eles não têm conteúdo. O elemento não sendo vazio inicia-se com uma marca de início, <elemento>, e termina com uma marca de término, </elemento>.

## Atributos

Os atributos são associados aos elementos. Eles possuem valores e são localizados na abertura do elemento. Por exemplo:

```
<input type="text">
```

O elemento *input* cujo atributo *type* possui o valor *text*. Em XML, todos os valores de atributos devem estar entre aspas. Normalmente atributos vazios são omitidos do elemento.

## b. Referências a entidades

Alguns caracteres são reservados para identificar o início de uma marcação. O sinal de menor, <, por exemplo, identifica o início de uma marca de início ou término desta. Para inserir esses caracteres em seu documento como conteúdo, deve haver uma alternativa para representá-los. Em XML, entidades são usadas para representar esses caracteres especiais.

As entidades devem ter um único nome. Para ser utilizada, a entidade deve simplesmente ser referenciada pelo nome. As

referências das entidades iniciam com o E comercial ('&') e terminam com um ponto e vírgula (';'). Por exemplo, a entidade *lt* insere um literal < em um documento. Para representar o sinal de maior dentro do XML, deverá usar &lt; resultando no seguinte código:

```
<?xml version="1.0" encoding="utf-8"?>  
<operacao>6 &lt; 7</operacao>
```

### c. Comentários

Os comentários são idênticos aos utilizados no HTML. Iniciam-se com <!-- e terminam com -->. Os comentários podem conter qualquer caractere, a única exceção se refere à literal “--”. É possível colocar comentários entre marcas em qualquer lugar do XML. Comentários não fazem parte de um conteúdo textual de um documento XML. Um processador XML não necessita reconhecer esse bloco na aplicação.

### Sessões CDATA

Apesar de ser possível converter diversos tipos de caracteres para uma entidade, existem casos que esse processo pode se tornar complexo. Para que possamos escrever diversos tipos de caracteres sem nos preocupar em converter os elementos, utilizamos as sessões CDTAT. Em um documento, uma sessão CDATA instrui o analisador para ignorar a maioria dos caracteres de marcação. Vamos levar em consideração uma instrução PHP dentro de um XML. Ele pode conter caracteres que o analisador XML normalmente reconheceria como elementos, assim teremos a seguinte forma:

```
<?xml version="1.0" encoding="utf-8"?>
<operacao><![CDATA[
if ((@$_POST["CAMPO1"]>0)||(@$_POST["CAMPO2"]>0)&&(@$_
POST["CAMPO3"]>0)) {
    echo "ok";
}
]]>
</operacao>
```

Em todo o conteúdo da sessão (<![CDATA[, e o fim da seção, ]]>), os caracteres são passados diretamente para a aplicação, sem interpretação. Elementos, referências a entidades, comentários e instruções de processamento são todos irreconhecíveis e os caracteres que os compõem são passados literalmente para a aplicação. As únicas literais que não é possível ser contida nas sessões CDATA são as próprias marcações de início e fim da sessão.

O XML é um documento que pode ser gerado por diversos *softwares* e linguagens diferentes. Essa portabilidade permitiu sua ampla utilização entre diversas tecnologias. Para ilustrar essa flexibilidade, veja como é simples a criação de um XML utilizando o PHP. Neste exemplo o documento é gerado de forma semelhante a um arquivo texto de largura fixa ou limitado por caractere utilizando a função *fopen*.

```
<HTML>
<HEAD>
<TITLE>Documento PHP</TITLE>
</HEAD>
<BODY>
<?php
/*
CONFIGURAÇÕES
```

Primeiramente você deve a permissão no sistema operacional para que “usuários da Web” ou o usuário que detém o serviço do servidor Web possa criar e apagar arquivos no diretório/pasta onde esse arquivo está salvo.

```
*/  
  
$teste= array(array('1A','Zehh','10'),  
             array('1A','Tatiana','9'),  
             array('1B','João','3'),  
             array('1B','Bruna','5'),  
             array('1B','Bianca','8'));  
  
// Vetor simbolizando a leitura de uma fonte de dados  
  
$arquivo=fopen("arquivoteste.xml","w+");  
  
// fopen é a função que abre/cria um arquivo; ou uma url  
  
// w+ Abre para leitura e escrita; coloca o ponteiro do arquivo no começo  
do arquivo e reduz o comprimento do arquivo para zero. Se o arquivo  
não existir, tenta criá-lo.  
  
// Para saber mais sobre esta função pode acessar http://www.php.net/  
manual/pt_BR/function.fopen.php  
  
$xml=<?xml version="1.0" encoding="ISO-8859-1"?>\n";  
  
$xml.= "<escola>\n";  
  
// $xml é uma variável onde está sendo armazenado informações de  
texto, neste caso o que será escrito no arquivo  
  
$sala=0;  
  
$controle=false;  
  
// variável de controle para este caso trabalhando com vetor  
  
for ($i=0; $i<count($teste);$i++) {  
  
    if ($sala!=$teste[$i][0]) {  
  
        $sala=$teste[$i][0];  
  
        $controle=true;  
  
        if ($i!=0) {  
  
            $xml.= "</salas>\n";  
  
        }  
  
    } else {  
  
        $controle=false;  
  
    }  
  
    if ($controle==true) {
```

```
$xml.= "<salas nome=\"$sala\">\n";
$controle=false;
}

$xml.= "<aluno>
<nome><![CDATA[\".$teste[$i][1].\"]]></nome>
<nota valor=\"\".$teste[$i][2].\"\" />
</aluno>\n";
}

$xml.= "</salas>
</escola>";
fwrite($arquivo, $xml);
// escreve no arquivo designado o conteúdo de uma variável o texto
fclose($arquivo);
// fecha o arquivo
echo "arquivo gerado!!<br>";
/*
?>

<a href="lerxml.php?arquivo=arquivoteste.xml">Clique aqui
para ler o conteúdo</a><br>

<a href="arquivoteste.xml" target="_blank">Veja o xml</a>
</BODY>
</HTML>
```

Você pode perceber que, apesar de o arquivo resultante ser um documento XML, as funções empregadas neste caso são as mesmas se caso fôssemos construir um documento texto. Isso permite que linguagens que ainda não possuem bibliotecas próprias para o XML possam construir arquivos de comunicação nesse formato. Vejamos outro exemplo com os mesmos dados, porém utilizando uma biblioteca própria para o XML.

```
<HTML>
<HEAD>
<TITLE>Documento PHP</TITLE>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<BODY>
<?php
$teste= array(array("1A","Zehh","10"),
             array("1A","Tatiana","9"),
             array("1B","João","3"),
             array("1B","Bruna","5"),
             array("1B","Bianca","8"));

$xml = new DOMDocument("1.0", "ISO-8859-1"); // cria o cabeçalho do XML
$xml->preserveWhiteSpace = false; //ignora todos os espaços em brancos
$xml->formatOutput = true; //gera o código formatado
$root = $xml->createElement("escola"); //cria o primeira tag (raiz)
$sala=0;
// variável de controle para este caso trabalhando com vetor
for ($i=0; $i<count($teste);$i++) {
    // Força o agrupamento das salas
    if ($teste[$i][0]!=$sala) {
        $salasXML = $xml->createElement("salas");
        $salaXML_no=$root->appendChild($salasXML);
        $sala=$teste[$i][0];
    }
    // cria os elementos do xml aluno, nome e nota
    $alunoXML = $xml->createElement("aluno");
    // o elemento ou tag nome terá o nome do aluno como texto
    $nomeXML = $xml->createElement("nome");
    // cria uma sessão CDATA que permite caracteres especiais
    $cdataNome = $xml->createCDATASection($teste[$i][1]);
```

```

//cria o elemento nota sem texto
$notaXML = $xml->createElement("nota");

//acrescenta a tag aluno como filho da tag sala
$salunoXML_no=$salaXML_no->appendChild($salunoXML);

//acrescenta a tag nome como filha da tag aluno
$nomeXML_no=$salunoXML_no->appendChild($nomeXML);

    //acrescenta a sessão CDATA com o nome do aluno dentro da tag
    nome
    $nomeXML_no->appendChild($cdataNome);

    // acrescenta a tag nota como filha da tag aluno
    $notaXML_no=$salunoXML_no->appendChild($notaXML);

    // adiciona o atributo valor para a tag nota
    $notaXML_no->setAttribute("valor",$teste[$i][2]);

/*
    IMPORTANTE

    Uma vez realizado o setAttribute não pode ser acrescentado mais
    elementos a tag

*/
//Caso for o último elemento da tag sala adiciona o atributo nome para
a sala
if ($teste[$i][0]!=$teste[$i+1][0]) {
    $salaXML_no=$salaXML_no->setAttribute("nome",$teste[$i][0]);
}
}

$xml->appendChild($root);
$xml->save("testeXMLDom.xml"); //salvar arquivo
?>
<a href="lerxml.php?arquivo=testeXMLDom.xml">Clique aqui para ler o
conteúdo</a><br>
<a href="testeXMLDom.xml" target="_blank">Veja o xml</a>
</BODY>
</html>

```

Neste exemplo uma classe foi utilizada para a geração do XML, permitindo uma construção lógica utilizando todo o potencial do XML.



Existem diversos outros elementos que podem ser acrescentado ao XML utilizando a classe DOM. Pesquise um pouco mais sobre essa poderosa classe do PHP.



Existe um segredo para a utilização da classe DOM. O *encoding* do XML para aceitar acentuação deve ser **ISO-8859-1**. Porém, para que a classe consiga converter o *charset*, da página deve ser **UTF-8**. Caso contrário, a classe não conseguirá salvar o arquivo XML.

Já pode observar duas formas de criar um documento XML. Pode perceber que, apesar de os dois exemplos utilizarem os mesmos dados, a forma de construção se distingue muito. Vamos montar agora a leitura desse documento por meio do PHP.

```
<HTML>
<HEAD>
<TITLE>Lê o XML</TITLE>
</HEAD>
<body>
<?php
$arquivo = file_get_contents($_GET["arquivo"]);
// recupera a informação de um arquivo ou url e atribui para uma variável
// texto
$xml = simplexml_load_string($arquivo);
// transforma um texto no formato xml em objeto
for ($i=0;$i<count($xml->salas);$i++) {
    $nome_sala = $xml->salas[$i]["nome"];
    /*
        ...
    */
}
```

**IMPORTANTE**

Para usar esta classe, é necessário conhecer o arquivo xml que está retornando.

Vou explicar aqui não colocando a conceituação de orientação a objetos, mas explicando de forma simples e em linguagem usual.

As regras são as seguintes:

Para cada *tag XML*, é criado um objeto.

Os blocos de *tags* que contenham diversas sub-*tag* podem ser acessada como vetor, indicando o número da ocorrência de cada um.

Cada atributo da *tag* pode ser acessado pelo nome como um vetor.

\*/

```
echo "<p>Sala ".($i+1).": $nome_sala<br>";
for ($j=0;$j<count($xml->salas[$i]->aluno);$j++) {
    $nome_aluno_da_sala = $xml->salas[$i]->aluno[$j]->nome;
    $nota_aluno_da_sala = $xml->salas[$i]->aluno[$j]->nota["valor"];

    echo "Nome do aluno <b>".($j+1)."</b> sala <b>$nome_sala</b>:<br>
<em><strong>$nome_aluno_da_sala</strong></em><br>";
    echo "Nota do aluno <b>".($j+1)."</b> da sala <b>$nome_sala</b>:<br>
<em><strong>$nota_aluno_da_sala</strong></em><br><br>";
}

echo "</p>";
}
?>
<a href="javascript:back();">Voltar para a geração do xml</a>
</body>
```

</HTML>

Desta forma, você pode ler qualquer arquivo XML apenas adequando à manipulação das informações que forem resgatadas.

## ***Namespaces***

O *Namespace* em XML foi desenvolvido com o intuito de resolver o problema de conflitos de elementos. Em função do XML permitir a criação de qualquer nome de *tag* para os documentos, pode ocorrer de diferentes usuários em diferentes contextos criarem *tags* com os mesmos nomes. O *Namespace* permite realizar uma declaração o contexto utilizado sobre uma *tag*. Poderíamos utilizar o XML para registrar o valor de um livro da seguinte forma.

```
<?xml version="1.0" encoding="utf-8"?>
<livro>
    <título>XML uma visão do futuro!</título>
    <valor moeda="REAL">50,00</valor>
</livro>
```

Em outro contexto, agora é possível representar os dados do autor.

```
<?xml version="1.0" encoding="utf-8"?>
<autor>
    <título>Sr</título>
    <nome>Joaquim da Silva</nome>
</autor>
```

Quando lemos os dois documentos, percebemos que o ‘título’ do livro não possui relação com o ‘título’ do autor. Porém, um sistema automatizado não possui tal discernimento entre essas duas *tags*. Possivelmente ele interpretará as *tags* ‘título’ da mesma forma.

Podemos solucionar esse problema realizando uma declaração de *Namespace* para que todos os elementos dentro de um bloco possam ser referenciados a ele. Utilizamos a seguinte declaração.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<livro xmlns="urn.endereçodaaplicação.com:TituloDaAplicação">
    <título>XML uma visão do futuro!</título>
    <valor moeda="REAL">50,00</valor>
</livro>
```

Outra forma de declaração é a utilização de abreviações e atribuição de mais de um *Namespace* ao documento. Forçando que a aplicação tenha o comportamento idêntico a *tags* dentro de documentos diferentes.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<lv:livro xmlns:lv="urn.endereçodaaplicação.com:TituloDaAplicação"
xmlns:dinheiro="urn:Financeiro:Dinheiro">
    <lv:título>XML uma visão do futuro!</lv:título>
    <dinheiro:valor moeda="REAL">50,00</dinheiro:valor>
</lv:livro>
```

## f. Declarações de Tipos de Documentos

Um dos maiores poderes da XML é que ela permite que você crie seus próprios nomes para marcas. Dependendo da aplicação é provável não ser significativo que os elementos estejam dispostos em uma ordem lógica. Podem estar ordenadas de forma aleatória, teríamos o seguinte exemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<pessoas>
    <Joaquim>
        Diga
        <saudação>
            Olá querida
        </saudação>
        , Maria.
```

```
</Joaquim>  
<Pedro>  
<saudação>  
    Até mais tarde, Maria.  
</saudação>  
</Pedro>  
</pessoas>
```

Aos nossos olhos, podemos não encontrar o menor sentido nas instruções escritas. Mas XML processa de forma normal as informações. Não há nada de errado com esse documento XML. Pelo contrário, ele possui significado ao menos para a pessoa que o escreveu ou esteja escrevendo uma folha de estilos ou aplicação. Para processar o documento deve haver alguma restrição na sequência e alinhamento das marcas. Declarações são onde essas restrições podem ser expressas.

As declarações em XML permitem a um documento comunicar meta-informações ao analisador a respeito do seu conteúdo. Meta-informação inclui as sequências e os alinhamentos permitidos para as marcas, os valores de atributos e seus tipos e padrões, os nomes de arquivos externos que podem ser referenciados e se eles podem ou não conter XML.

Há quatro tipos de declarações em XML: declarações de tipos de elementos, declarações de listas de atributos, declarações de entidades e declarações de notações.

## Seção 2 – Definição de Tipos de Documentos DTD

A estrutura de um documento é definida pelos DTDs. DTD é onde se especifica quais os elementos e atributos são permitidos no documento. Embora não seja necessário que um documento XML tenha um DTD correspondente, recomenda-se, em alguns casos, a sua utilização para garantir a conformidade do documento.

O padrão DTD utiliza somente elementos definidos, diferentemente do XML simples, em que se pode acrescentar diversos elementos diferentes. No DTD isso não pode ocorrer. Todo o elemento deve ser definido.

Torna-se importante a sua utilização quando é necessária a extrema conformidade de informações, quando empresas trocam informações via XML com outras. Assim, para não haver discrepâncias de informações, é necessário definir quais elementos serão utilizados.

As DTDs especificam a estrutura de um documento, e são definidas por meio da gramática EBNF (*Extended Backus-Naur Form*), e não na sintaxe XML.

## Processamento de Documentos XML

Para processar documentos XML e verificar se a sintaxe e o tipo utilizados são válidas utiliza-se aplicações denominadas processadores de XML ou Parsers.

O Parser de validação percorre o documento para verificar se sua estrutura está de acordo com as regras de um DTD. Se o documento não estiver de acordo com a DTD, o Parser acusará um erro.

Existem Parsers que não fazem validação do tipo, mas apenas verifica se a sintaxe do documento está correta. Os Parsers são utilizados para transformar o documento XML em uma estrutura de árvore para que as aplicações possam acessar seu conteúdo.

## Definição de Tipo de Documento

A referência de um documento para uma DTD é feita em uma definição de tipo de documento. A DTD pode estar contida no documento (subconjunto interno) ou existir externamente como um arquivo separado (subconjunto externo).

Desta forma, as DTDs são introduzidas em um documento XML por meio da declaração DOCTYPE. Uma declaração de tipo de documento é colocada no prólogo de um documento XML, começando por <!DOCTYPE e terminando por >. O exemplo a seguir representa um subconjunto interno.

```
<?xml version="1.0"?>
<!DOCTYPE email [
  <!ELEMENT email (para,de,assunto,conteúdo)>
  <!ELEMENT para (#PCDATA)>
  <!ELEMENT de (#PCDATA)>
  <!ELEMENT assunto (#PCDATA)>
  <!ELEMENT conteúdo (#PCDATA)>
]>
<email>
  <para>Maria</para>
  <de>João</de>
  <assunto>Jantar</assunto>
  <conteúdo>O que há para o jantar?</conteúdo>
</email>
```

Os subconjuntos externos estão colocados fisicamente em outro arquivo que, em geral, termina com a extensão “.dtd”, embora essa extensão não seja necessária. Os subconjuntos externos são

especificados pela palavra-chave SYSTEM ou pela palavra-chave PUBLIC.

```
<?xml version="1.0"?>
<!DOCTYPE email SYSTEM "definicaoXML.dtd" >
<email>
<para>Maria</para>
<de>João</de>
<assunto>Jantar</assunto>
<conteúdo>O que há para o jantar?</conteúdo>
</email>
```

Veja que estamos apontando para uma definição externa chamada ‘definicaoXML.dtd’. O seu conteúdo pode ser representado da seguinte forma:

```
<?xml version="1.0"?>
<!DOCTYPE email [
  <!ELEMENT email (para,de,assunto,conteúdo)>
  <!ELEMENT para (#PCDATA)>
  <!ELEMENT de (#PCDATA)>
  <!ELEMENT assunto (#PCDATA)>
  <!ELEMENT conteúdo (#PCDATA)>
]>
```

Outra forma de representar a definição seria referencia-la em uma url externa. Desta forma, vários Parsers podem usar um DTD em comum para validar seus documentos.

```
<?xml version="1.0"?>
<!DOCTYPE email SYSTEM "http://www.w3.org/TR/html4/strict.dtd" >
```

Vimos um subconjunto externo utilizando um URL, que indica que a DTD é amplamente utilizada. No exemplo, está sendo apontada a DTD para documentos HTML. Assim, os analisadores que não têm cópia local da DTD podem utilizar

a URL fornecida para fazer o *download* da DTD, para fins de validação do documento.

A declaração de tipo de documento pode ainda conter arquivos externos de validação comuns. Acrescentando ainda elementos específicos para o documento. Desta forma, podemos especificar todo o documento mesmo se todos os elementos não estejam detalhados em arquivos externos de especificações.

```
<?xml version="1.0"?>
<!DOCTYPE email SYSTEM "definicaoXML.dtd" >
<!ELEMENT tipo (#PCDATA)>
<email>
  <para>Maria</para>
  <de>João</de>
  <tipo>html</tipo>
  <assunto>Jantar</assunto>
  <conteúdo>O que há para o jantar?</conteúdo>
</email>
```

Lembre-se que essa declaração interna é visível somente dentro do documento onde está definido. Para que outros documentos sejam validados por este elemento, torna-se imprescindível a sua definição no arquivo externo definicaoXML.dtd, no caso do exemplo. As DTDs que são usadas por mais de um documento devem ser colocadas no subconjunto externo.

## Declarações de Tipos de Elementos

Declarações de tipos de elementos identificam os nomes dos elementos e a natureza do seu conteúdo. Essa declaração vem na forma de ELEMENT.

```
<!ELEMENT email (para+,de| assunto*,conteúdo?)>
```

A declaração representa o elemento ‘email’. Contém um modelo de conteúdo que representa os elementos que constarão

dentro do elemento. A separação entre vírgulas representa uma obrigatoriedade. O sinal de adição representa que o elemento ‘para’ pode se repetir, mas pelo menos deve aparecer uma vez. Já o ponto de interrogação (após o conteúdo), indica que o elemento é opcional, mas não deve aparecer mais de uma vez. O símbolo ‘|’ representa uma ligação de ‘ou’. Essa ligação indica que o elemento ‘de’ ou o elemento ‘assunto’ serão representados. O símbolo “\*” indica que o elemento ‘assunto’ é opcional e pode ocorrer ‘n’ vezes no documento.

Após a declaração do modelo do documento, devemos declarar o tipo de informações apresentadas no XML. Cada elemento deve ser declarado separadamente da seguinte forma:

```
<!ELEMENT para (#PCDATA)>
```

O elemento ‘para’ é chamado **identificador genérico** e vem depois da declaração ELEMENT. O conjunto entre parênteses que segue o identificador genérico (#PCDATA) indica o conteúdo permitido para o elemento e é chamado de especificação de conteúdo. Esses dados serão analisados pelo analisador XML, portanto, qualquer texto de marcação (<, >, &, ‘ e ”) será tratado como marcação. As declarações para todos os elementos usados em qualquer modelo de conteúdo devem estar presente para que um processador XML verifique a validade do documento.

## Declarações de Atributos

Como não podia ser diferente, a declaração de atributos é uma parte fundamental do DTD. A declaração de atributo especifica uma lista de atributos para um determinado elemento usando a declaração pela palavra reservada *ATTLIST*. O elemento pode ter qualquer número de atributos.

```
<?xml version="1.0"?>
<!DOCTYPE email [
  <!ELEMENT email (para,de,assunto,conteúdo)>
  <!ATTLIST email tipo CDATA #REQUIRED>
  <!ELEMENT para (#PCDATA)>
  <!ELEMENT de (#PCDATA)>
  <!ELEMENT assunto (#PCDATA)>
  <!ELEMENT conteúdo (#PCDATA)>
]>
<email tipo="html">
  <para>Maria</para>
  <de>João</de>
  <assunto>Jantar</assunto>
  <conteúdo>O que há para o jantar?</conteúdo>
</email>
```

A declaração anterior especifica que o elemento ‘email’ possui um atributo ‘tipo’. A palavra reservada CDATA indica que ‘tipo’ pode conter qualquer texto de caracteres, exceto os caracteres <, >, & , ‘ e “ . Perceba que a palavra reservada CDATA na declaração de tipo possui significado diferente da sessão CDATA em um documento XML, como já explicado. Outra característica desse exemplo é a palavra reservada *REQUIRED*. Indicação do atributo ser obrigatório, caso o atributo seja opcional, devemos substituir *REQUIRED* por *IMPLIED*.

```
<!ATTLIST email tipo CDATA #IMPLIED>
```

Podemos ainda especificar o tipo de valores que os atributos podem receber. Ou seja, nos atributos é possível representar uma lista de valores nos quais o documento deve constar.

```
<!ATTLIST email tipo (html | text) 'html'>
```



Lembre-se que o uso do DTD requer que o interpretador esteja preparado para interpretá-lo. Caso contrário, o DTD será ignorado.

A declaração de atributos possui três partes: nome, tipo, valor padrão. Quanto ao nome, pode ser adotado o que melhor condiz com a regra do sistema. Obviamente o nome do atributo não pode se repetir para o mesmo elemento tampouco receber os marcadores especiais (<, >, &, ‘ ou ”). Quanto aos tipos, existem atualmente seis tipos de atributos:

<b>CDATA</b>	Qualquer texto é permitido. Não confunda atributos CDATA com sessões CDATA, eles não têm relação.
<b>ID</b>	Identifica de forma única um elemento. Ou seja, não pode haver outro elemento com atributo ID com o mesmo valor. Os IDs identificam unicamente elementos individuais em um documento. Os elementos podem ter um único atributo ID.
<b>IDREF ou IDREFS</b>	Apontam para elementos com um atributo ID. O valor de um atributo IDREFS pode conter valores IDREF múltiplos separados por espaços em branco.
<b>ENTITY ou ENTITIES</b>	Entidades são utilizadas para associar nomes com outros fragmentos do documento. Podem ser utilizadas entidades internas ou externas. O valor de um atributo ENTITIES pode conter valores de entidades múltiplos separados por espaços em branco.
<b>NMTOKEN ou NMTOKENS</b>	Um tipo de atributo mais restritivo, cujo valor consiste em caracteres letras, dígitos, pontos, sublinhados, hífens e dois-pontos. O valor de um atributo NMTOKENS pode conter valores NMTOKEN múltiplos separados por espaços em branco.
<b>Atributos Enumerados =</b>	Você pode especificar que o valor de um atributo deve ser pego de uma lista específica de nomes. Isso é frequentemente chamado de tipo enumerado, porque cada um dos valores possíveis está explicitamente enumerado na declaração.

Além dos tipos de atributos, é possível definir o valor padrão para cada atributo:

<b>#REQUIRED</b>	Especifica que o atributo deve aparecer no elemento. O documento XML não é válido se o atributo estiver faltando.
<b>#IMPLIED</b>	Especifica que, se o atributo não aparece no elemento, então a aplicação que está usando o documento XML pode usar qualquer valor que por ventura escolher. Um valor para este atributo não é requerido.
<b>#FIXED</b>	Especifica que o valor do atributo é constante e não pode ser diferente no documento XML. Um valor para o atributo é requerido.
<b>"valor"</b>	Especifica um valor padrão para o atributo, se não for definido um.

O analisador XML procura sempre normalizar os valores dos atributos. As referências de caractere são substituídas por caracteres referenciados, referências a entidades são resolvidas (recursivamente) e espaços em branco são normalizados.

## Declarações de Entidades

Declarações de entidades permitem associar um nome ou uma definição a uma sigla ou outro código de identificação. Essa identificação é livre e pode conter quaisquer caracteres que não infrinjam os marcadores do XML. Os principais tipos utilizados são: entidades internas e entidades externas.

### Entidades Internas

Entidades internas são as declarações realizadas dentro do próprio documento. Essas declarações têm o escopo limitado para o documento não sendo compartilhadas por outros.

```

<?xml version="1.0"?>
<!DOCTYPE email [
  <!ELEMENT email (para,de,assunto,conteúdo)>
  <!ATTLIST email tipo (html | text) "html">
  <!ELEMENT para (#PCDATA)>
  <!ELEMENT de (#PCDATA)>
  <!ELEMENT assunto (#PCDATA)>
  <!ELEMENT conteúdo (#PCDATA)>
  <!ENTITY UNISUL "Universidade do Sul de Santa Catarina">
]>
<email tipo="text">
  <para>Maria</para>
  <de>João</de>
  <assunto>Jantar</assunto>
  <conteúdo>Vou enviar um e-mail para a &UNISUL;
  </conteúdo>
</email>

```

O resultado desse documento será uma conversão da cadeia &UNISUL; para Universidade do Sul de Santa Catarina. Essa conversão será realizada em qualquer local do documento. Em outras palavras, permite definir atalhos para textos frequentemente digitados ou textos que se espera que sejam alterados, como o estado de revisão de um documento.

Entidades internas podem incluir referências para outras entidades internas, mas é errado elas serem recursivas. A especificação XML predefine cinco entidades internas:

```

? &lt; produz o sinal de menor, <
? &gt; produz o sinal de maior, >
? &amp; produz o E comercial, &
? &apos; produz um apóstrofo, '
? &quot; produz aspas, "

```

## Entidades externas

Entidades externas são as associações de definições externas ao documento atual. Permitem o documento XML referenciar o conteúdo de um outro arquivo. Se elas contêm texto, o conteúdo do arquivo externo é inserido no ponto de referência e analisado como parte do documento referente. Dados binários não são analisados e podem somente ser referenciados em um atributo; eles são usados para referenciar figuras e outro conteúdo não XML no documento.

Esse tipo de entidade é possível inserir seu conteúdo ao documento atual. Assim, podemos anexar vários documentos para que o seu conteúdo possa relacionar com o restante do documento.

```
<?xml version="1.0"?>
<!DOCTYPE email [
  <!ELEMENT email (para,de,assunto,conteúdo)>
  <!ATTLIST email tipo (html | text) "html">
  <!ELEMENT para (#PCDATA)>
  <!ELEMENT de (#PCDATA)>
  <!ELEMENT assunto (#PCDATA)>
  <!ELEMENT conteúdo (#PCDATA)>
  <!ENTITY UNISUL "Universidade do Sul de Santa Catarina">
  <!ENTITY inserirTexto SYSTEM "testelnserir.xml">
]>
<email tipo="text">
  <para>Maria</para>
  <de>João</de>
  <assunto>Jantar</assunto>
  <conteúdo>Vou enviar um e-mail para a > &UNISUL;
  &inserirTexto;
  </conteúdo>
</email>
```

O processador XML analisará o conteúdo desse arquivo como se ele ocorresse literalmente no local. Representando assim o resultado:

```
<?xml version="1.0" ?>
<!DOCTYPE email (View Source for full doctype...)>
- <email tipo="text">
  <para>Maria</para>
  <de>João</de>
  <assunto>Jantar</assunto>
- <conteúdo>
  Vou enviar um e-mail para a >
  Universidade do Sul de Santa Catarina
  <conteudoCompleto>Texto padrão</conteudoCompleto>
</conteúdo>
</email>
```

## Seção 3 – XML-Schema

O XML-Schema, também conhecido como XSD (*XML Schema Definition*), é uma maneira mais eficiente de descrever a estrutura e o conteúdo de um documento XML. O propósito de um XML-Schema é definir os blocos de construção permitidos em um documento XML, como um DTD.

O XML-Schema é semelhante a uma DTD, ambas definem a estrutura do XML. Originalmente o XML utilizou das DTDs como forma de definição de sua estrutura. Com os avanços em sua utilização, percebeu-se que as DTDs eram imensamente limitadas, não permitindo diversos pontos, como:

- as DTDs não são expansíveis, não permitem que novos elementos sejam introduzidos;
- DTD não possuem suporte a tipos de dados;
- os DTDs não possuem suporte a *namespaces*, este recurso é limitado ao documento XML.

Os desenvolvedores viram que as DTDs não podiam acompanhar a flexibilidade do XML. A saída para esse dilema foi o desenvolvimento de um novo padrão de definição da estrutura do XML, que foi denominada de XSD ou *XML Schema Definition*, também chamada **XML-Schema**.

Tanto a DTD quanto o XSD, ou XML-Schema, definem:

- elementos que podem aparecer em um documento;

- atributos que podem aparecer em um documento;
- que elementos são elementos filhos;
- a ordem dos elementos filhos;
- o número de elementos filhos;
- se um elemento é vazio ou pode incluir texto;
- tipos de dados para elementos e atributos;
- valores padrão e fixos para elementos e atributos.

As vantagens do XSD, ou XML-Schema, sobre a DTD são:

- XML-Schemas são extensíveis para adições futuras, com isso você pode:
  - reutilizar seu Schema em outros Schemas;
  - criar seus próprios tipos de dados derivados dos tipos padrões;
  - referenciar múltiplos esquemas em um mesmo documento.
- XML-Schemas são mais ricos e úteis que DTDs.
- XML-Schemas são escritos em XML e, com isso, você:
  - não tem de aprender outra linguagem;
  - pode usar seu editor XML para editar seus arquivos XML-Schemas;
  - pode usar seu parser XML para verificar seus arquivos XML-Schemas;
  - pode manipular seu XML Schema com XML DOM;
  - pode usar seu XML-Schema com XSLT.

- XML-Schemas suportam tipos de dados, com isso é mais fácil:
  - descrever conteúdo de documentos permissíveis;
  - validar os dados;
  - trabalhar com dados de um banco de dados;
  - definir restrições aos dados;
  - definir padrões/formatos de dados;
  - converter dados entre diferentes tipos.
- XML-Schemas suportam *namespaces*.

## Princípios do XSD

Já vimos que o *namespace* é usado para identificar unicamente o nome dos elementos. Basicamente consiste em nomear todos os elementos de um XML com um mesmo prefixo. Para identificar onde termina o prefixo e onde começa o nome propriamente dito, usa-se o caractere ":" (dois pontos).

Por definição, os elementos XML-Schema começam com o prefixo "xs". Desta forma, não há problemas em identificar elementos do XML-Schema e os elementos do XML caso eles tenham a mesma *tag*.

Visando a um melhor entendimento, vamos utilizar um exemplo de *e-mail*. Vamos comparar a definição com a DTD e o XSD.

```
<?xml version="1.0"?>
<!DOCTYPE email [
  <!ELEMENT email (para,de,assunto,conteúdo)>
  <!ELEMENT para (#PCDATA)>
  <!ELEMENT de (#PCDATA)>
```

```
<!ELEMENT assunto (#PCDATA)>
<!ELEMENT conteúdo (#PCDATA)>
]>
<email>
<para>João</para>
<de>Joaquim</de>
<assunto>Reunião</assunto>
<conteúdo>A reunião está agendada para as 10:00 hs</conteúdo>
</email>
```

Um XML-Schema para o mesmo XML pode ser escrito da seguinte maneira:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com" xmlns="http://www.
w3schools.com" elementFormDefault="qualified">
<xs:element name="email">
<xs:complexType>
<xs:sequence>
<xs:element name="para" type="xs:string"/>
<xs:element name="de" type="xs:string"/>
<xs:element name="assunto" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Os seguintes elementos “para”, “de” e “assunto” são todos elementos que possuem um tipo definido: “string”. O tipo é definido pelo atributo “type” do elemento xs:element.

O elemento nota não possui um atributo definindo o seu tipo. Podemos considerar que seu tipo é complexo ou complexType,

pois seu conteúdo é composto pela sequência de elementos de, para, assunto e texto.




---

Elementos complexos são aqueles que contêm outros elementos e elementos simples são aqueles que não contêm outros elementos.

---

## Elementos simples

Elementos simples não podem conter outros elementos ou atributos. O seu conteúdo é somente no formato de caractere, porém é possível alterar o tipo de caractere contido nesse elemento. O XML-Schema possui a flexibilidade do XML onde você pode construir seus próprios tipos. Ele já possui alguns predefinidos (integer, float, date, etc) que nos facilitam na construção.

```
xs:string
xs:decimal
xs:integer
xs:boolean
xs:date
xs:time
```

Veja como definir elementos simples com esses tipos.

Definição	Elemento
<xs:element name="nome" type="xs:string"/>	< nome >Silva</ nome >
<xs:element name="paginas" type="xs:integer"/>	< paginas >31</ paginas >
<xs:element name="datavencimento" type="xs:date"/>	< datavencimento >2010-05-01</ datavencimento >

Além dos tipos, podemos definir valores padrões para os elementos. Esses valores podem ser *fixed* ou *default*.

```
<xs:element name="color" type="xs:string" fixed="red"/>  
<xs:element name="color" type="xs:string" default="red"/>
```

Quando você define um atributo *fixed* a um elemento, significa que esse elemento não será capaz de receber outro valor a não ser aquele definido. Diferentemente, o atributo *default* indica que por padrão o elemento receberá um valor, mas poderá ser tranquilamente alterado.



Somente elementos complexos podem possuir atributos.

## Atributos

O XSD define que os atributos são elementos simples e somente elementos complexos podem ter atributos. Mesmo assim, a definição do atributo é diferente da definição de um elemento. Para definir um elemento em XSD, utilizamos *xs:element*, mas para definir um atributo usamos o elemento *xs:attribute*.

Ao definir um atributo, podemos declarar como ele sendo obrigatório ou opcional. Ainda, como vimos nos elementos simples, podemos definir um valor *default* ou *fixed*, como vemos a seguir.

```
<xs:attribute name="linguagem" type="xs:string" use="optional"/>  
<xs:attribute name="linguagem" type="xs:string" use="required"/>
```

Quando definimos o tipo, tanto para elemento simples quanto para o atributo, tem por objetivo realizar uma restrição quanto ao seu conteúdo. Desta forma, pretende-se sempre validar o elemento quanto à informação esteja em um formato válido ou não.

## Facetas de restrição

A restrição do conteúdo não se restringe aos tipos predefinidos. O XSD nos permite construir tipos de dados por meio de restrições. Para isso, é necessário utilizar um subconjunto do tipo de dados original, ou seja, é preciso limitar o domínio de valores possíveis. Isso é realizado por meio das “facetas de restrição”.

As facetas são muito úteis principalmente para os atributos. Frequentemente eles só podem assumir alguns valores predefinidos. Um exemplo clássico é o atributo sexo, onde criamos um conjunto finito e pequeno de valores que o atributo sexo pode suportar.

O tipo de dados “sexo” é uma *string* de um caractere que só aceita os valores ‘M’ e ‘F’ respectivamente ‘masculino’ e ‘feminino’. Esse tipo de dados poderá ser construído por meio do elemento *enumeration*:

```
<?xml version="1.0" encoding="utf-8"?>
<xss:simpleType name="sexo">

    <xss:restriction base="xsd:string">

        <xss:enumeration value="M"/>
        <xss:enumeration value="F"/>

    </xss:restriction>
</xss:simpleType>
```

Além do elemento *enumeration*, existem outros elementos que nos facilitam na definição das facetas. São eles:

- *length*: define o comprimento de um tipo de dados.
- *minLength*: define o comprimento mínimo de um tipo de dados.
- *maxLength*: define o comprimento máximo de um tipo de dados.
- *pattern*: define a expressão regular (*Regular Expressions*) a ser utilizada na validação do tipo.

- ***enumeration***: limita os valores possíveis de um tipo de dados a um conjunto previamente especificado.
- ***whiteSpace***: define como devem ser tratados os espaços em branco.
- ***minInclusive***: limite mínimo considerando o valor especificado.
- ***minExclusive***: limite mínimo desconsiderando o valor especificado.
- ***maxInclusive***: limite máximo considerando o valor especificado.
- ***maxExclusive***: limite máximo desconsiderando o valor especificado.
- ***totalDigits***: número máximo de dígitos considerando a parte inteira e a parte decimal.
- ***fractionDigits***: número máximo de casas decimais.



---

Lembre-se que é necessário também especificar o tipo de dados base, ou seja, a partir de qual tipo de dados o novo tipo está sendo construído. Isso é feito a partir de um elemento *restriction* e do atributo 'base'.

---

Vejamos outro exemplo onde criaremos um elemento ‘estado’. Como no exemplo anterior, utilizaremos o elemento *enumeration* para criar uma lista de valores.

```
<?xml version="1.0" encoding="utf-8"?>
<xss:element name="estado" type="TipoEstado"/>
<xss:simpleType name="TipoEstado">
    <xss:restriction base="xss:string">
        <xss:enumeration value="SP"/>
        <xss:enumeration value="PR"/>
        <xss:enumeration value="SC"/>
        <xss:enumeration value="RS"/>
    </xss:restriction>
</xss:simpleType>
```

Poderíamos definir o mesmo elemento estado de uma forma mais direta:

```
<?xml version="1.0" encoding="utf-8"?>
<xss:element name="estado">
    <xss:simpleType>
        <xss:restriction base="xss:string">
            <xss:enumeration value="SP"/>
            <xss:enumeration value="PR"/>
            <xss:enumeration value="SC"/>
            <xss:enumeration value="RS"/>
        </xss:restriction>
    </xss:simpleType>
</xss:element>
```

Perceba que a primeira forma possui um uso mais prático. Caso necessitássemos aplicar a mesma restrição a algum outro elemento do meu XML, atribui-se o tipo “TipoUF” sem precisar reescrever todos os valores novamente. O tipo “TipoUF” não possui uma restrição quanto ao seu uso em outros elementos.

Vamos construir uma faceta para restringir um tipo de dado chamado ‘livro’. Para descrever o livro, ele será um texto com comprimento mínimo de três e máximo de 100 caracteres.

Vamos ainda substituir os espaços em branco duplicados por um único espaço em branco. O XSD ficaria assim:

```
<?xml version="1.0" encoding="utf-8"?>  
<xss:simpleType name="livro">  
  <xss:restriction base="xsd:string">  
    <xss:minLength value="3"/>  
    <xss:maxLength value="100"/>  
    <xss:whiteSpace value="collapse"/>  
  </xss:restriction>  
</xss:simpleType>
```

## Elementos complexos

O elemento complexo é composto por outros elementos, porém esses elementos podem conter apenas texto simples. Para isso, o elemento deve possuir um ou mais atributos. Por essa definição, até mesmo elementos que não contenham informação textual (elementos vazios) podem ser considerados complexos. Desde que esses elementos vazios possuam pelo menos um atributo.



---

Elemento complexo é todo o elemento que contenha outros elementos e/ou atributos.

---

Como já explicado, existem duas formas de definição:

- criar um tipo de forma independente ao elemento e depois atribuir esse tipo ao elemento;
- definir diretamente um elemento nomeando-o e declarando seu conteúdo dentro da própria declaração do elemento.

Vamos adotar a primeira, apesar de a segunda parecer mais simples à primeira vista. Definir um tipo independente acarreta algumas vantagens, como:

- outros elementos podem usar aquele tipo;
- pode-se criar outros tipos que usem aquele tipo.

Definiremos um elemento complexo chamado “contato”. Esse elemento conterá três sub-elementos: “nome”, “endereço” e “cep”. Teríamos assim o seguinte XML:

```
<?xml version="1.0"?>
<contato>
  <nome>João da Silva</nome>
  <endereco>Av: Getúlio Vargas</endereco>
  <cep>12345678911</cep>
</contato >
```

Já o XSD ficaria:

```
<?xml version="1.0"?>
<xss:element name="contato" type="tContato"/>
<xss:complexType name="tContato">
  <xss:sequence>
    <xss:element name="nome" type="xss:string"/>
    <xss:element name="endereco" type="xss:string"/>
    <xss:element name="cep" type="xss:positiveInteger"/>
  </xss:sequence>
</xss:complexType>
```

Veja que o elemento xs:sequence tem a função de descrever a sequência em que cada sub-elemento deverão aparecer. Portanto, se no XML um elemento se um elemento não estiver na ordem definida, o XML não será validado pelo meu XML-Schema.

Utilizando desse formato de declaração é possível estender o tContato para tipos mais complexos. Podemos criar um tipo tPessoa que estenda o tipo tContato. Estender possui a mesma característica da Herança na Orientação Objeto. O novo tipo estendido possui as características do pai e mais algum

detalhamento. Veja como ficaria um tipo tPessoa que, além de nome, possui idade, sexo e endereço.

```
<?xml version="1.0"?>
<xs:complexType name="tPessoa">
    <xs:complexContent>
        <xs:extension base="tContato">
            <xs:sequence>
                <xs:element name="idade" type="xs:integer"/>
                <xs:element name="sexo" type="xs:string"/>
                <xs:element name="endereço" type="xs:string"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Os elementos complexos podem ser vazios, como já foi mencionado. Para isso, é necessário que esse elemento possua pelo menos um atributo.

```
<aluno matricula="45980"/>
```

O elemento aluno não possui conteúdo e, para definir esse tipo de elemento complexo, devemos definir um tipo que permita apenas elementos em seu conteúdo, mas nós não declaramos nenhum elemento realmente.

```
<?xml version="1.0"?>
<xs:element name="aluno" type="tMatricula"/>
<xs:complexType name="tMatricula">
    <xs:attribute name="matricula" type="xs:positiveInteger"/>
</xs:complexType>
```

Vamos supor que o elemento não esteja mais vazio, ele contém um texto e um atributo. Vamos usar o mesmo elemento aluno

do exemplo anterior, mas agora que ele possui um texto como conteúdo.

```
<aluno prodid="45980">Joaquim Carvalho</aluno>
```

A declaração inteligente seria assim:

```
<?xml version="1.0"?>
<xs:element name="aluno" type="tAluno"/>
<xs:complexType name="tAluno">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="matricula" type="xs:positiveInteger"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Sempre que o elemento contiver apenas conteúdo simples (texto e atributos), adiciona-se um *xs:simpleContent* em torno do conteúdo. Dentro do *xs:simpleContent* você pode optar por usar o *xs:extension* para expandir o tipo simples base do elemento ou o *xs:restriction* para limitá-lo.

Vamos definir um XSD onde se tenha elementos que contenham textos e outros elementos. Para isso, vamos tomar como exemplo um elemento chamado “contrato”.

```
<?xml version="1.0"?>
<contrato>
  Número <nrContrato>1032</nrContrato>
  Declaro que o Sr. <nome>João Silva</nome>
  Firma acordo de prestação de serviço com a Empresa
  <empresa>Qualquer LTDA.</empresa>
  Data: <data>2010-08-10</data>
</contrato>
```

O XSD seria definido da seguinte forma:

```
<?xml version="1.0"?>
<xss:element name="contrato" type="tContrato"/>
<xss:complexType name="tContrato" mixed="true">
    <xss:sequence>
        <xss:element name="nrContrato" type="xs:positiveInteger"/>
        <xss:element name="nome" type="xs:string"/>
        <xss:element name="empresa" type="xs:string"/>
        <xss:element name="data" type="xs:date"/>
    </xss:sequence>
</xss:complexType>
```

Para permitir que caracteres apareçam entre os elementos filhos de “contrato”, o atributo *mixed* deve ser definido como “true”. Em vez de utilizar o elemento *xss:sequence*, poderíamos usar o *xs 1:all* para indicar que os elementos filhos podem aparecer em qualquer ordem. Existe também o *xs:choice*, que é usado quando queremos que apenas um dos filhos apareça.

## Utilizando o PHP para validar o XML com o XSD

Vimos que o XSD é na realidade um XML que valida o próprio XML. Como não poderia ser diferente, vários *softwares* realizam a validação do XML por meio de um XSD, assim como diversas linguagens possuem bibliotecas que efetuam essa tarefa para nós.

Vamos demonstrar um exemplo de uma validação do XML utilizando o XSD por meio do PHP desenvolvida por Pedro Padron. Este exemplo se baseia em um catálogo de livros. Teríamos assim o XML:

```
<?xml version="1.0"?>
<catalogo>
    <livro>
        <isbn>8504006115</isbn>
        <autor>George Orwell</autor>
        <titulo>1984</titulo>
        <paginas>302</paginas>
    </livro>
    <livro>
        <isbn>8588639173</isbn>
        <autor>Ramez Elmasri</autor>
        <autor>Shankant Navathe</autor>
        <titulo>Sistemas de Bancos de Dados</titulo>
        <paginas>744</paginas>
        <editora>744</editora>
    </livro>
</catalogo>
```

Perceba que nesse documento, apesar de existir dois registros, eles possuem o mesmo número de elementos. Porém, a informação é plausível e deve ser validada. As definições para o XSD são estabelecidas a partir das seguintes regras:

- todo livro deve ter um único ISBN;
- todo livro deve ter pelo menos um autor e no máximo N autores;
- todo livro deve ter um único título;
- todo livro deve ter uma especificação do número de páginas;
- o nome da editora é opcional e assume-se que apenas uma editora tenha os direitos de publicação da obra;

- a ordem de aparição dos dados deve ser: ISBN, autor, título e número de páginas.

Convertendo essas regras para as definições do XSD fica assim:

```
<?xml version="1.0"?>
<xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- Este é o nosso nó principal -->
    <xss:element name="catalogo">
        <!-- Define um tipo formado por outros elementos -->
        <xss:complexType>
            <!-- Determina que os elementos a seguir devem aparecer nesta mesma ordem -->
            <xss:sequence>
                <!-- Define que o tipo "livro" pode ocorrer N vezes (unbounded = sem limite)-->
                <xss:element name="livro" maxOccurs="unbounded">
                    <!-- Define que "livro" é formado por outros elementos -->
                    <xss:complexType>
                        <!-- Os componentes de "livro" devem aparecer nessa ordem -->
                        <xss:sequence>
                            <!-- Define o ISBN -->
                            <xss:element name="isbn">
                                <!-- Define como tipo simples por não ser formado por outros elementos -->
                                <xss:simpleType>
                                    <!-- Define o tipo "string" para ISBN e informa que existem restrições -->
                                    <xss:restriction base="xs:string">
                                        <!-- Determina como 10 o tamanho máximo -->
                                        <xss:length value="10"/>
                                    </xss:restriction>
                                </xss:simpleType>
                            </xss:element>
                        </xss:sequence>
                    </xss:complexType>
                </xss:element>
            </xss:sequence>
        </xss:complexType>
    </xss:element>
</xsschema>
```

```

</xs:element>

    <!-- Define que o elemento "autor" é obrigatório, mas pode
aparecer N vezes -->

        <xs:element name="autor" type="xs:string" minOccurs="1"
maxOccurs="unbounded"/>

            <!-- O valor-padrão para minOccurs e maxOccurs é 1 quando
não forem definidos -->

                <xs:element name="titulo" type="xs:string"/>

                <xs:element name="paginas" type="xs:integer"/>

                <!-- Elemento editora não é obrigatório -->

                    <xs:element name="editora" type="xs:string"
minOccurs="0"/>

            </xs:sequence>

        </xs:complexType>

    </xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

```

Veja que o cenário normalmente se trata de que no servidor existe um arquivo de validação XSD e um *script* para validar o XML, no nosso caso o PHP. O mais comum para essa utilização é quando o *software* recebe um XML externo e requer que seja validado antes do processamento. Para isso, vejamos a construção do PHP:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />

<title>Valida&ccedil;&atilde;o do XML utilizando XSD</title>

```

```
</head>

<body>

<?php

/**

 * Ao tentar validar um arquivo XML, se algum erro
 * for encontrado a libxml irá gerar Warnings, o que
 * não creio que seja o mais interessante para nós.
 *
 * Para evitar que isso aconteça, você pode determinar
 * que irá obter os erros por sua própria conta. Lembre-se
 * que esta função abaixo deve ser chamada antes de
 * instanciar qualquer objeto da classe DomDocument!
 */

libxml_use_internal_errors(true);

/* Cria um novo objeto da classe DomDocument */
$objDom = new DomDocument();

/* Carrega o arquivo XML */
$objDom->load("catalogo.xml");

/* Tenta validar os dados utilizando o arquivo XSD */
if (!$objDom->schemaValidate("catalogo.xsd")) {

    /**

     * Se não foi possível validar, você pode capturar
     * todos os erros em um array
     */

    $arrayAllErrors = libxml_get_errors();

    /**

     * Cada elemento do array $arrayAllErrors
     * será um objeto do tipo LibXmlError
     */

    print_r($arrayAllErrors);
} else {
```

```

/* XML validado! */

echo "XML obedece &agrave;s regras definidas no arquivo XSD!";

}

?>

</body>

</html>

```



Procure alterar o arquivo catalogo.xml. Veja o que ocorre se você alterar as *tags* ou o tipo de informação.

## Seção 4 – RSS

Hoje em dia é comum buscar informações em tempo real. O envio dessas informações por *e-mail* acabou se tornando muito ruim, uma vez que, além das informações que necessitamos, recebemos uma avalanche de conteúdos sem qualquer interesse. Para resolver esse dilema, desenvolve-se um “dialeto” para XML chamado de *Rich Site Summary*, o RSS.



RSS é um recurso desenvolvido em XML que permite aos responsáveis por sites e blogs divulgarem notícias ou novidades destes. Para isso, o link e o resumo daquela notícia, ou a notícia na íntegra, são armazenados em um arquivo XML. Esse arquivo é conhecido como *feed*, *feed RSS*.

Graças a esse recurso, *sites* e *blogs* podem divulgar conteúdo novo de maneira rápida e precisa. Quanto ao usuário, este pode ser informado de cada novidade de seus *sites* preferidos.

Para se fazer uso desse recurso, o usuário deve incluir o *feed* (que nada mais é que o RSS) do *site* que deseja acompanhar em um programa leitor de RSS, chamado de **agregador**. O agregador

lê o conteúdo do *feed* e lista em uma interface. Cada *software* agregador possui sua própria interface. Os *softwares* agregadores permitem a inclusão de diversos *feeds*. Essa ação acarreta em um ganho em agregação das informações.

O sistema RSS facilita muito a leitura das informações. Você não necessita entrar em vários *sites* para se manter informado. Basta ler as chamadas das notícias ou atualizações no leitor-RSS e acessar as notícias que for de seu interesse.



O padrão RSS foi originalmente desenvolvido pela Netscape no início de 1999. O projeto foi descontinuado, a empresa entendia que o RSS não era viável para os usuários. A Netscape desenvolveu o padrão até a versão 0.90. A UserLand, uma pequena empresa, acreditou no padrão e simplificou o código para aplicar nas suas ferramentas de *blogs*. O RSS 0.91 foi lançado.

Um grupo de desenvolvedores continuou o RSS 0.90 baseando-o no padrão RDF, paralelamente a UserLand. Esses desenvolvedores lançaram a versão 1.0. A UserLand, no entanto, continuou seu trabalho, lançando versões como 0.92, 0.93 até que finalmente chegou à versão 2.0. Certamente, a UserLand daria a numeração 1.0 a esta última, mas essa sequência já tinha sido adotada pelo outro grupo. Diversos grupos desenvolvem, hoje, o padrão RSS. Cada uma acaba nomeando o projeto de diversas formas diferentes.

## RDF

Para saber mais sobre RDF  
pesquise em [www.w3.org/RDF](http://www.w3.org/RDF)

**RDF** é a sigla para *Resource Description Framework* e trata-se de uma especificação normalmente baseada em XML, que tem como objetivo automatizar e ampliar recursos para a internet por meio de representação de informações. O RDF tem como base o trabalho de uma série de grupos que desenvolvem tecnologias de informação. Isso significa que o RDF não foi criado exclusivamente para ferramentas como o RSS, como muitos pensam. Na verdade, se estudarmos o RDF a fundo, veremos que ele tem utilidade em várias aplicações, como em sistemas de busca e mecanismos de compartilhamento de informação.

## O que é Atom

Ao contrário de RSS, **Atom** não é uma sigla, é um formato para divulgação de notícias. Há quem diga que esse projeto é, inicialmente, uma proposta de unificação do RSS 1.0 e do RSS 2.0. O Atom também é baseado em XML, mas seu desenvolvimento é tido como mais sofisticado. O grupo que nele trabalha tem até o apoio de grandes corporações, como o Google.

**Para saber mais sobre Atom, pesquise em <www.atomenabled.org>.**

A maioria dos agregadores de *feed* disponíveis ao usuário suporta tanto as versões do RSS como o Atom.

## O padrão

O RSS é baseado no XML, nada melhor do que explicar o funcionamento do RSS por meio de seu código-fonte. Para servir de exemplo, usaremos o código a seguir. Note que ele contém campos básicos, o que significa que *feeds* RSS disponíveis na internet podem contar com mais detalhes:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="0.91">
<channel>
<title>Folha.com - Em cima da hora - Principal</title>
<link>http://redir.folha.com.br/redir/online/emcimadahora/rss091/*http://www1.folha.uol.com.br/emcimadahora/</link>
<description>Primeiro jornal em tempo real em língua portuguesa</description>
<language>pt-br</language>
<copyright>Copyright Folha Online. Todos os direitos reservados.</copyright>
<docs>http://redir.folha.com.br/redir/online/emcimadahora/rss091/*http://www1.folha.uol.com.br/folha/conheca/arquivo_e_copyright.shtml</docs>
<webMaster>webmaster@folha.com.br (Webmaster Folha Online)</webMaster>
<image>
```

```
<title>Folha.com - Em cima da hora - Principal</title>
<url>http://www1.folha.uol.com.br/folha/images/logo-folha_online-
88x31.gif</url>
<link>http://redir.folha.com.br/redir/online/emcimadahora/rss091/*http://
www1.folha.uol.com.br/emcimadahora/</link>
<width>88</width>
<height>31</height>
<description>Primeiro jornal em tempo real em língua portuguesa</
description>
</image>
<item>
<title>Novela: Nara dá um tapa na cara de Beto em "Uma Rosa com
Amor"</title>
<link>http://redir.folha.com.br/redir/online/emcimadahora/rss091/*http://
www1.folha.uol.com.br/ilustrada/763300-novela-nara-da-um-tapa-na-
cara-de-beto-em-uma-rosa-com-amor.shtml</link>
<description>
&lt;i&gt;A programação está sujeita a alteração devido à edição dos
capítulos e é de responsabilidade da emissora&lt;/i&gt;
&lt;b&gt;Uma Rosa com Amor (SBT, 20h15)&lt;/b&gt;
Egídio pede para Elisa levar um suco a Serafina. Serafina pega o copo de
suco, mas não toma. Serafina diz para Raquel que se aproximar de Milton,
vai arrebentar a cara dela. Serafina afirma para Nara que está ajudando
Claude a construir casas para o povo. Serafina afirma que não tomará
o suco, porque ele pode estar envenenado. Beto desafia Egídio e pega
o suco para tomar. Egídio impede. Egídio explica que cuspiu e colocou
laxante no suco de Serafina. Egídio joga o copo na parede. Serafina sai da
casa. Mrs. Smith pergunta se Claude esteve no restaurante. Claude nega.
&lt;a href="http://redir.folha.com.br/redir/online/emcimadahora/
rss091/*http://www1.folha.uol.com.br/ilustrada/763300-novela-nara-da-
um-tapa-na-cara-de-beto-em-uma-rosa-com-amor.shtml"&gt;Leia
mais&lt;/a&gt; (12/07/2010 - 00h12)</description>
</item>
<item>
<title>Novela: Alfredo descobre que Joca matou o senador em
"Ribeirão do Tempo"</title>
```

```

<link>http://redir.folha.com.br/redir/online/emcimadahora/rss091/*http://
www1.folha.uol.com.br/ilustrada/763294-novela-alfredo-descobre-que-
joca-matou-o-senador-em-ribeirao-do-tempo.shtml</link>

<description>
&lt;i&gt;A programação está sujeita a alteração devido à edição dos
capítulos e é de responsabilidade da emissora&lt;/i&gt;
&lt;b&gt;Ribeirão do Tempo (Record, 22h)&lt;/b&gt;
Eleonora pede para Arminda representá-la no velório do senador.
Lincon pede para Ellen encontrá-lo na Folha da Corredeira. Sílvio diz
a Guilherme que Zuleide ligou para dizer que ele estava na praça com
Tião. Guilherme fala para o pai que Tião/Diana não tem nada demais.
Três mulheres choram muito diante do caixão do senador. Beatriz fica
irritada e pergunta a Nicolau quem são as mulheres. Lincon conta a Ellen
que uma mensagem foi enviada da Secretaria de Segurança do Estado,
ameaçando o senador de morte.
&lt;a href="http://redir.folha.com.br/redir/online/emcimadahora/
rss091/*http://www1.folha.uol.com.br/ilustrada/763294-novela-
alfredo-descobre-que-joca-matou-o-senador-em-ribeirao-do-tempo.
shtml"&gt;Leia mais&lt;/a&gt; (12/07/2010 - 00h08)</description>
</item>
</channel>
</rss>
```

A primeira linha informa que se trata de um documento XML. A segunda se refere à versão do RSS usado. Neste caso, é a 0.91. O elemento *channel* indica o canal do *feed*. O elemento *link* é a referência em que o RSS está associada. O elemento *title* é preenchido com o nome do *site* (Folha.com – Em cima da hora – Principal). Já o elemento *description* fornece uma breve descrição do *site*. No entanto, muitos *webmasters* preferem informar o *slogan* do *site* nesse campo. O elemento *language* informa o idioma usado no *site* (português do Brasil). O elemento *copyright* fala sobre o direito autoral das informações. Já o *docs* representa o *link* para os documentos sobre o direito autoral. O elemento *webmaster* informa o responsável por ele.

Nesse exemplo está sendo utilizado um bloco de imagem. Esse bloco tem a finalidade de mostrar a logo do jornal no RSS e não é obrigatório.

Até o momento vimos apenas as informações de cabeçalho. Agora analisaremos os elementos do corpo do RSS. O bloco *item* está a notícia. Para cada uma, é necessário ter uma área *item* específica. Nesse exemplo, foram retiradas apenas duas notícias, portanto possui dois blocos *item*. O elemento *title* dentro do bloco *item* possui a informação do título da notícia. O elemento *description* permite a visualização de um resumo do texto (ou do texto completo). O elemento *link* indica o endereço da notícia.



Existem diversos softwares que você pode utilizar para a geração do RSS. Pesquise sobre eles.



Figura 5.1: logotipo do RSS.

Fonte: favor incluir fonte completa.

## Logotipo

A partir de sua versão 1.5, o navegador de internet Mozilla Firefox passou a adotar um símbolo diferente para o RSS. Depois disso, o símbolo se tornou padrão para representar *feeds* RSS, isto é, se tornou universal.

No caso do Firefox ou de outros programas compatíveis com RSS, basta clicar no símbolo em questão – que geralmente aparece no canto direito da barra de endereços ou em um ponto específico da página visitada – para que o *feed* do *site* acessado seja adicionado.

O exemplo a seguir foi retirado da página <broculos.net> e é muito simples a forma de representar um escritor de *feeds* utilizando o PHP.

```
<?php
/**
 * For demonstration purposes, the data is defined here.
 * In a real scenario it should be loaded from a database.
 */
$channel = array("title" => "Exemplo de RSS feed",
```

```

"description" => "Exemplo de construção de RSS utilizando o
PHP.",

"link"      => "http://www.broculos.net",

"copyright" => "Copyright (C) 2008 Broculos.net");

$items = array(
    array("title"      => "Exemplo 1",
          "description" => "Descrição do primeiro Exemplo.",
          "link"        => "http://www.example.com/example1.html",
          "pubDate"     => date("D, d M Y H:i:s O", mktime(22, 10, 0, 12, 29,
2008))),
    array("title"      => "Exemplo 2",
          "description" => "Esta é a descrição do Exemplo 2.",
          "link"        => "http://www.example.com/example2.html",
          "pubDate"     => date("D, d M Y H:i:s O", mktime(14, 27, 15, 1, 3,
2008)))
);

$output = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?\" . ">";
$output .= "<rss version=\"2.0\">";
$output .= "<channel>";
$output .= "<title>" . $channel["title"] . "</title>";
$output .= "<description>" . $channel["description"] . "</description>";
$output .= "<link>" . $channel["link"] . "</link>";
$output .= "<copyright>" . $channel["copyright"] . "</copyright>";
foreach ($items as $item) {
    $output .= "<item>";
    $output .= "<title>" . $item["title"] . "</title>";
    $output .= "<description>" . $item["description"] . "</description>";
    $output .= "<link>" . $item["link"] . "</link>";
    $output .= "<pubDate>" . $item["pubDate"] . "</pubDate>";
    $output .= "</item>";
}

```

```
$output .= "</channel>";  
$output .= "</rss>";  
header("Content-Type: application/rss+xml; charset=ISO-8859-1");  
echo $output;  
?>
```



## Síntese

Você viu nesta unidade o quanto o XML é flexível e se adéqua a diversas funcionalidades no qual necessitamos. Ainda verificou com ocorre uma definição de tipo de documento assim como seus elementos e suas características. Verificamos que, apesar de essas definições serem úteis, acabam por ser um tanto limitadas. Para acompanhar o desenvolvimento do próprio XML, necessitamos de uma definição mais apurada, o XSD.

Por fim, você viu uma grande utilidade do XML que se refere à comunicação com diversos *browser* e aplicativos para sintetizar informações.



## Atividades de autoavaliação

Ao final de cada unidade, você será provocado a realizar atividades de autoavaliação. O gabarito está disponível no final do livro-didático, mas se esforce para resolver as atividades sem ajuda do gabarito, pois assim você estará estimulando a sua aprendizagem.

- 1) Pesquise ao menos três tecnologias onde o XML está sendo utilizado. Sintetize as principais características da funcionalidade e publique no Fórum da ferramenta de aprendizado virtual.



## Saiba mais

Se você desejar, aprofunde os conteúdos estudados nesta unidade ao consultar as seguintes referências:

**W3Schools. W3Schools Online Web Tutorials**, 2010.

Disponível em: <[www.w3schools.com/](http://www.w3schools.com/)>. Acesso em: 01 mar. 2010.

**W3C. World Wide Web Consortium**, 2010. Disponível em: <[www.w3.org/](http://www.w3.org/)>. Acesso em: 2 mar. 2010.

**Dicas-1**, 2010. Disponível em <[www.dicas-1.com.br//](http://www.dicas-1.com.br/)>. Acesso em: 16 maio 2010.

**iMasters**, 2010. Disponível em <[http://imasters.uol.com.br//](http://imasters.uol.com.br/)>. Acesso em: 16 maio 2010.

**Plugmasters**, 2010. Disponível em: <[www.plugmasters.com.br](http://www.plugmasters.com.br)>. Acesso em: 15 maio 2010.



# Asynchronous Javascript and XML (AJAX)



## Objetivos de aprendizagem

Ao final do estudo desta unidade, é importante que você:

- Entenda a tecnologia que envolve o AJAX.
- Saiba o que são requisições assíncronas.
- Saiba como interagir o AJAX com o PHP.
- Conheça os princípios para construir uma aplicação utilizando o AJAX.



## Seções de estudo

**Seção 1** O que é o AJAX

**Seção 2** Método GET no AJAX

**Seção 3** Método POST no AJAX

**Seção 4** Framework



## Para início de estudo

Todas as aplicações Web sempre foram vistas como um enorme potencial. A portabilidade que possui associada com a arquitetura da internet coloca as aplicações Web em outro patamar de abordagem. Essa mesma vantagem acarreta em um problema crucial: o tempo de resposta das requisições.

O desenvolvimento Web se baseia em requisições feitas pelos usuários a um servidor. Consequentemente esses usuários aguardam a resposta dessas requisições. Torna-se extremamente lento quando ocorre alguma inconsistência na regra de negócio imposta no servidor. O formulário retorna com um aviso onde o usuário deve ajustar as informações.

Várias alternativas foram criadas para minimizar esse problema. Alguns adotaram que o formulário já retornaria com os campos preenchidos. Mesmo assim, o tempo de resposta e a frustração do usuário ainda permaneciam.

Outra solução, um pouco mais engenhosa e gerenciada pelo JavaScript, utilizava páginas auxiliares abertas por meio de *pop-ups*, onde eram realizadas as validações e retornava as informações. Essa solução tinha um problema fundamental, pois, quando se utilizava *pop-up*, os *browsers* começaram a bloquear por causa das propagandas excessivas nos *sites*. A aplicação parava de funcionar e era necessário um recurso para abrir uma segunda janela a cada momento que uma validação no servidor fosse necessária.

Apesar das soluções encontradas, o problema se prolongava. Como poderiam os desenvolvedores criar uma aplicação Web utilizando recursos de uma aplicação *desktop*? A base para essa resposta nasceu com o AJAX.

## Seção 1 – O que é o AJAX

AJAX (acrônimo em língua inglesa de *Asynchronous JavaScript and XML*) é o uso sistemático do JavaScript e do XML para tornar o navegador mais interativo com o usuário. O AJAX utiliza de solicitações assíncronas de informações e é uma iniciativa na construção de aplicações Web mais dinâmicas e criativas.

O AJAX não é uma tecnologia. São realmente várias tecnologias trabalhando juntas, cada uma fazendo sua parte, oferecendo novas funcionalidades.

O AJAX foi criado por Jesse James Garrett, da Adaptive Path, em meados de fevereiro de 2005. Todas as tecnologias presentes no AJAX já existem há algum tempo e são de conhecimento amplamente difundido entre os desenvolvedores Web. São elas:

- XHTML;
- JavaScript;
- CSS;
- DOM;
- XML.

As atualizações das páginas existem desde os primórdios da internet. O LiveScript da Netscape possibilitava a inclusão de *scripts* de páginas Web que poderiam ser executados pelo cliente, isso na década de 1990. O JavaScript é a evolução do LiveScript. A Microsoft foi a responsável pela introdução do objeto *XMLHttpRequest* para criar e gerenciar solicitações e respostas assíncronas.

O AJAX promoveu um *marketing* para as tecnologias que o compõe, tornando o próprio AJAX uma das tecnologias de desenvolvimento Web mais em evidência, permitindo que aplicações Web concorram com as aplicações de *desktop*.

A fim de entender como o mecanismo do AJAX funciona, vamos analisar o processo de requisição de uma aplicação Web sem a utilização do AJAX:

1. Usuário preenche os campos do formulário e submete;
2. O navegador gera uma solicitação para o servidor;
3. O servidor processa a solicitação;
4. O servidor gera uma resposta que contém a página exata que o navegador interpretará;
5. O navegador limpa a janela e carrega a página enviada pelo servidor;
6. O usuário interage com a nova página.

Você pode observar que em todo o transcorrer desse simples processo o usuário ficou totalmente obsoleto. Não se podia interagir com as informações enquanto a requisição estava sendo processada. Esse tipo de requisição é chamado de **requisição síncrona**. Esse modelo tornou-se obsoleto, pois foi concebido para os documentos hipertextos originais da internet.

O AJAX acrescenta uma camada nas aplicações Web que a utiliza. Essa camada fica entre o cliente e o servidor e gerencia a comunicação entre ambos. Essa camada é criada quando o usuário interage com a página: o cliente cria um objeto de controle chamado *XMLHttpRequest* que gerencia a solicitação. A camada do AJAX recebe todas as solicitações e enfileira as requisições para o servidor. À medida que o servidor responde, o AJAX manipula a página do cliente conforme os dados de retorno.

O gerenciamento do AJAX permite que o formulário seja submetido parcialmente. Essa característica torna fundamental a maior interação com o usuário. Quando trabalhamos com pequenos formulários, o envio de informações parciais acaba não se tornando parte fundamental do processo. Porém, em formulários maiores ou que necessite de validações constantes, torna-se indispensável.

## Aplicabilidade do AJAX

Existem diversos tipos de situações em que o uso do AJAX é amplamente utilizado. Algumas situações são mais perceptíveis, como: validação de formulários, atualização de enquetes e de carrinhos de compras (*e-commerce*), conversação *on-line* (*chats*), entre outras que priorizam a atualização de informações em tempo real.

Outro exemplo bastante comum são os **sites de relacionamento**.



Muitos utilizam plenamente o AJAX para melhorar a interação com os usuários. Um dos mais populares sites conhecidos – a rede social Orkut – pertencente ao Google e utiliza bastante esse recurso quanto à classificação de amigos. O Twitter, outra rede social, utiliza o AJAX para verificar se existem novos tweets para o usuário.

Podemos encontrar o AJAX em *sites* para melhorar a apresentação de imagens. Muitos utilizam esse recurso para que não seja necessário carregar todas as fotos no início da página, tornando assim o tempo de resposta muito menor. Um exemplo da facilidade do AJAX para a imagem é o **Google Maps**. Conforme o usuário arrasta o mapa, as imagens vão carregando por setores.

As compras realizadas em *sites* de venda vêm gradualmente utilizando mais e mais o AJAX. Muitos já utilizam quando necessitam alterar o valor no carrinho de compras, tanto no cálculo do frete como quando trocamos a quantidade de mercadorias.

## Seção 2 – Método GET no AJAX

Já vimos que o AJAX é uma tecnologia responsável pela maior interação entre as aplicações Web e os usuários. Agora vamos aprofundar esse conjunto de tecnologias, para isso, vamos utilizar o PHP com linguagem de servidor para nos ajudar.

O método de formulário GET envia as informações para o servidor na url da página. Esse método é mais simples, porém menos seguro de utilizar. A aplicabilidade desse método está em um maior gerenciamento do processo de envio das informações ao servidor. Não é necessário registrar um elemento para que os dados possam ser descritos na url.

Vamos considerar uma página simples para iniciar o entendimento da tecnologia. Nesta página é necessário escolher o estado e deve carregar algumas cidades.

```
<!DOCTYPE html PUBLIC "-// W3C // DTD XHTML 1.0 Transitional // EN"
"http :// www.w3. org /TR/ xhtml1 / DTD / xhtml1 - transitional . dtd ">

<html xmlns=" http://www.w3.org/1999/xhtml " xml:lang="pt-br"
lang="pt-br">

<head>

<title>Exemplo 3 &minus; Escolha a Cidade</title>

<script type="text/ecmascript" src="ajaxInit.js"></script>

<script type="text/javascript">

function selecionaCidade (estado) {

    // testa o conteúdo da combo

    if (estado!="") {

        // chamada da função

        ajax = ajaxInit();

        if (ajax) {

            // realiza a leitura do retorno do AJAX

            ajax.onreadystatechange = escreveCidades ;

            // url onde se encontra o código do servidor
```

```
url = 'cidade.php?estado=' + estado;
// envia a url para o AJAX pelo método GET
ajax.open ('GET' , url , true );
// executa o AJAX
ajax.send (null);
}
} else {
// limpa a combo de cidades
limparCombo("cidade");
}
}

function escreveCidades () {
// verifica a preparação do AJAX, 4 está pronto
if ( ajax.readyState == 4) {
// status do AJAX
if ( ajax.status == 200) {
// o AJAX retorna em forma de texto.
// a função interpreta o retorno para montar a combo
acrescentaOption('cidade',ajax.responseText);
}
}
}

function acrescentaOption(combo, texto) {
// quebra o texto em um vetor
var option=texto.split("^");
// identifica o elemento no formulário neste exemplo é uma caixa
de seleção
var elemento=document.getElementById(combo);
var valores = new Array();
```

```
// limpa todo o conteúdo do lemento
elemento.length=0;
for (var i=0;i< option.length; i++) {
// acrescenta um conteúdo
elemento.length++;
// divide o valor retornado do AJAX em um vetor
valores = option[i].split("|");
// acrescenta um conteúdo
elemento.options[i] = new Option(valores[1], valores[0], 0, 0);
}
}

function limparCombo(combo) {
// recupera o elemento
var elemento=document.getElementById(combo);
// limpa todo o conteúdo do lemento
elemento.length=0;
// acrescenta um conteúdo
elemento.options[0] = new Option("-----", "", 0, 0);
}
</script>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/></head>
<body>

<form name="seleccioneCidade">
<label for="estado" accesskey="e">
<u>E</u>stado:
</label>
<select name="estado" id="estado" onChange="selecionaCidade(this.
options[this.selectedIndex].value)">
<option value="" selected="selected">-----</option>
```

```
<option value="pr">PR</option>
<option value="sp">SP</option>
</select>
<br/>
<label for="cidade" accesskey="c"><u>C</u>idade :</label>
<select name="cidade" id="cidade">
</select>
</form>
</body>
</html>
```

Nesta página, estamos criando o HTML, realizando a chamada do AJAX e tratando a resposta do AJAX.

O *script* de JS (ajaxInit.js) onde se realiza o instanciamento do objeto é o *XMLHttpRequest*. A este instanciamento podemos chamar de **AJAX puro**.

```
// Documento JavaScript

function ajaxInit () {
    // define variável
    var xmlhttp ;
    // tratamento de erros
    try {
        // chamada do objeto
        xmlhttp = new XMLHttpRequest () ;
    } catch (ee) {
        try {
            // criando o objeto sobre os navegadores diferentes da Microsoft
            xmlhttp = new ActiveXObject ("Msxml2.XMLHTTP");
        } catch (e) {
            try {
```

```
// criando objeto sobre o navegador da Microsoft
xmlhttp = new ActiveXObject ("Microsoft.XMLHTTP");

} catch (E) {

    // caso não de suporte
    xmlhttp = false;

}

}

return xmlhttp;

}
```

Podemos perceber que a utilização do objeto *XMLHttpRequest* é simples, mas requer cuidados. Veja que o código é pequeno, onde basicamente se restringe ao *ActiveXObject*. O restante das instruções são desvios para cada *browser* que devemos ter o cuidado de estar presente. Caso contrário, o AJAX pode não funcionar adequadamente. Vejamos como deve estar o PHP para trabalhar com este *script*.

```
<?

// recupera a informação passada pelo método GET
$estado = $_GET['estado'];
$cidades = "";
// Verifica qual estado está sendo trabalhado
if ( $estado=="pr" ) {
    //acrescenta informações para a variável $cidade
    $cidades .= "1|Tupãssi^";
    $cidades .= "2|Toledo^";
    $cidades .= "3|Cascavel^";
    $cidades .= "4|Pato Branco";
}
/// Verifica qual estado está sendo trabalhado
```

```

if ( $estado=="sp" ) {
    //acrescenta informações para a variável $cidade
    $cidades .= "5|Mogi^";
    $cidades .= "6|Palmeiras^";
    $cidades .= "7|Santos";
}
// escreve a variável
echo $cidades;
?>

```



Mesmo com essa simplicidade de código deve sempre existir alguns cuidados. A acentuação sempre é um problema. Procure utilizar codificação mais apropriada para o navegador.

O *readyState* pode ser utilizado para montar uma interação com o usuário mais agradável como uma animação informando que a requisição se encontra em processamento. Ele pode ocorrer de quatro tipos:

1. Sendo enviado;
2. Processando;
3. Armazenando;
4. Pronto.



A resposta do AJAX não ocorre uma única vez, mas é recursivo. Portanto, o processo de enviar a requisição e retornar ocorre várias vezes durante o processo.

Neste exemplo, verificamos a extensão do AJAX. Como ele se comporta em uma requisição de informações? Apesar de estarmos utilizando um formulário, o seu enfoque não está em registrar informações, mas no resgate. Perceba que você pode estar mesclando as metodologias conforme a regra de aplicação, utilizando o AJAX para compor as informações do formulário e

enviar as informações para o servidor no método tradicional para garantir a sua naveabilidade.

Um exemplo disso seria a inclusão de um produto a um carrinho de compras. Ele remete ao conteúdo do seu carrinho (sem uso do AJAX). Já no carrinho, caso queira aumentar a quantidade de produtos, retirar algum ou calcular o frete, utiliza-se o AJAX para uma maior interação.

Apesar de uma grande parte das aplicações Web trabalharem essencialmente com recuperação de dados, necessitamos em outros momentos incluir dados na aplicação. Para aumentar um pouco a complexidade, vamos montar um cadastro de pessoa com quatro campos: código, nome, UF Residência, Cidade de Residência. Vamos precisar de duas tabelas no banco de dados. Primeiro, a tabela de cidades:

```
DROP TABLE IF EXISTS prg3.cidade;

CREATE TABLE prg3.cidade (
    id_cidade int(10) unsigned NOT NULL AUTO_INCREMENT,
    nm_cidade varchar(45) COLLATE latin1_general_ci NOT NULL,
    id_uf varchar(2) COLLATE latin1_general_ci NOT NULL,
    PRIMARY KEY (id_cidade),
    KEY Index_2 (id_uf)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1
COLLATE=latin1_general_ci;
```

A outra tabela é de pessoas. Esta tabela será relacionada com a de cidade.

```

DROP TABLE IF EXISTS prg3.pessoa;

CREATE TABLE prg3.pessoa (
    id_pessoa int(10) unsigned NOT NULL AUTO_INCREMENT,
    nm_pessoa varchar(45) DEFAULT NULL,
    id_cidade int(10) unsigned NOT NULL,
    PRIMARY KEY (id_pessoa),
    KEY FK_pessoa_1 (id_cidade),
    CONSTRAINT FK_pessoa_1 FOREIGN KEY (id_cidade) REFERENCES
    cidade (id_cidade)
) ENGINE=InnoDB AUTO_INCREMENT=22 DEFAULT CHARSET=latin1;

```

Essas duas tabelas serão criadas no banco de dados prg3, já utilizado em outros exemplos.

Vamos montar o formulário HTML e nele acrescentaremos cada função JavaScript para manipular os dados e utilizar o AJAX. Podemos chamar esse arquivo de inicial.html.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Cadastro de Pessoas</title>

<script type="text/ecmascript" src="ajaxInit.js"></script>
<script type="text/javascript">

    // variável global
    var tipoDados="";

    // o restante dos scripts será acrescentado a partir deste ponto.

</script>

<style type="text/css">
<!--
.normal {

```

```
background-color: #FFF;  
color: #000;  
font-weight: normal;  
}  
-->  
</style>  
<link href="teste.css" rel="stylesheet" type="text/css" />  
</head>  
<body>  
<h1>Cadastro de Pessoas</h1>  
<hr />  
<form name="frmpessoa" method="post" enctype="multipart/  
form-data" onsubmit="validarcampos(this); return false;"  
onreset="limparstyles();">  
<table width="40%" border="1">  
<tr>  
<td scope="col"><label for="codigo">Código</label></td>  
<td scope="col"><input name="codigo" type="text" class="normal"  
id="codigo" value="" readonly="readonly" /></td>  
</tr>  
<tr>  
<td><label for="nome">Nome</label></td>  
<td>  
<input type="text" name="nome" id="nome" value=""  
obrigatorio="sim" descr="Nome" class="normal"/>  
</td>  
</tr>  
<tr>  
<td><label for="uf">UF Residencia</label></td>  
<td>
```

```
<select name="uf" class="normal" id="uf" onChange="carregarDados('estado='+this.options[this.selectedIndex].value, 'cidade.php', 'cidade')"  
obrigatorio="sim" descr="UF Residencia">  
    <option value="">-----</option>  
    <option value="SP">SP</option>  
    <option value="PR">PR</option>  
    <option value="SC">SC</option>  
    <option value="RS">RS</option>  
</select>  
</td>  
</tr>  
<tr>  
    <td><label for="cidade">Cidade Residencia</label></td>  
    <td>  
        <select name="cidade" class="normal" id="cidade" obrigatorio="sim"  
descr="Cidade Residencia" "><br />  
        <option value="">-----</option>  
</select>  
</td>  
</tr>  
<tr>  
    <td colspan="2" align="center">  
        <input type="submit" name="salvar" value="Salvar" />  
        &nbsp;  
        <input type="reset" name="limpar" value="Limpar" />  
        <br />  
        <input type="button" value="Listar" name="listar"  
        onclick="carregarDados ('', 'salvarDados.php', 'salvar')"/></td>  
</tr>  
</table>  
</form>  
<hr />
```

```
<p>  
<div id="pagina">  
</div>  
</p>  
</body>  
</html>
```

Existe um script em JavaScript chamada *ajaxInit.js* que está sendo carregado na página. Esse script já foi explicado em exemplos anteriores que se refere ao objeto *XMLHttpRequest*. Nesse formulário já está sendo feito as chamadas das funções. Uma já é conhecida em exemplos anteriores, na *combobox* cidade carregamos as cidades. É o primeiro elemento a ser acessado, já que para cadastrarmos qualquer informação é necessário informar o estado e a cidade em que reside a pessoa.

```
<select name="uf" class="normal" id="uf" onChange="carregarDados('estado='+this.options[this.selectedIndex].value,'cidade.php','cidade')"  
obrigatorio="sim" descr="UF Residencia">
```

Nesse trecho, está sendo realizada uma chamada de função genérica, ou seja, essa função chamada em outras partes do programa para realizar tarefas distintas. Assim, temos a função:

```
// função genérica para carregar informações  
function carregarDados (codigo, url, tipo) {  
    // variável auxiliar  
    var aux=codigo.split("=".  
    // testa o conteúdo  
    if (aux[1]!="") {  
        // chamada da função  
        ajax = ajaxInit();  
        if (ajax) {
```

```

        tipoDados=tipo;

        ajax.onreadystatechange = escreveDados ;
        url = url+'?'+codigo ;
        ajax.open ('GET' , url , true ) ;
        ajax.send (null) ;

    }

} else {

    // limpa a combo de cidades

    if (tipo=="cidade")

        limparCombo("cidade");

    }

}

```

Essa função monta a camada do AJAX para o envio e recebimento de informações e passa por três parâmetros:

código	são os parâmetros do método get enviados pela URL.
url	é a página de destino do AJAX.
tipo	é um controlador para realizar os desvios necessários do programa.

O que podemos ressaltar da função é a utilização da variável global *tipoDados* usada para transmitir o controlador. Ainda existe a chamada da função *limparCombo*, utilizada para excluir os elementos da *combobox* cidade caso não tenha sido escolhido nenhum estado. Outra chamada é a função *escreveDados*, que verifica o retorno do AJAX e manipula a informação recebida.

```

function escreveDados() {
    if ( ajax.readyState == 4) {
        if ( ajax.status == 200) {
            if (tipoDados=="cidade")
                acrescentaOption('cidade',ajax.responseText);
        }
    }
}

```

A função *escreveDados* verifica quando o AJAX retorna o código de sucesso para, então, poder enviar os dados a serem acrescentados na *combobox*. Perceba que está sendo testado o tipo para realizar um desvio. Mais adiante, acrescentaremos outros desvios a essa função.

Podemos então acrescentar as funções de manipulação de *combobox*: acrescentar dados, retirar e selecionar dados. Essas funções não utilizam o AJAX, porém utilizam conceitos de vetores e delimitadores de caracteres. Esse tipo de programação é um artifício amplamente utilizado para agrupar as informações e posteriormente poder separá-las de forma correta.

```

function selecionaCombo(objSel, valor) {
    for (i=0; i < objSel.length; i++){
        // verifica se o item possui o valor procurado.
        if (objSel.options[i].value==valor) {
            // seleciona o item
            objSel.selectedIndex = i;
            break;
        }
    }
    // acrescenta os dados na combobox
}

```

```
function acrescentaOption(combo, texto) {  
    //transforma o conjunto de caractere em um vetor  
    var option=texto.split("^");  
    var elemento=document.getElementById(combo);  
    var valores = new Array();  
    // limpa a combobox  
    elemento.length=0;  
    for (var i=0;i< option.length; i++) {  
        // adiciona um novo indice a combobox  
        elemento.length++;  
        // quebra em um novo vetor para montar os valores da combobox  
        valores = option[i].split("|");  
        // acrescenta o novo element na combobox  
        elemento.options[i] = new Option(valores[1], valores[0], 0, 0);  
    }  
}  
  
// limpa a combobox  
  
function limparCombo(combo) {  
    var elemento=document.getElementById(combo);  
    elemento.length=0;  
    elemento.options[0] = new Option("-----","", 0, 0);  
}
```

A montagem da *combobox* cidade se dá por meio de uma consulta ao banco de dados. Desta forma, foi utilizado um programa para realizar essa consulta e preparar os dados de retorno. O programa foi nomeado de *cidade.php*:

```
<?php  
  
$estado = $_GET['estado'];  
  
$cidades = "";  
  
// inclui os parâmetros para conectar no banco de dados  
include("variaveis.bd");  
  
// conecta do banco  
  
$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o  
banco de dados");  
  
// vamos selecionar o banco de dados errado o que resultará em um erro  
mysql_select_db("prg3");  
  
// a consulta é montada e encaminhada ao servidor  
  
$result= mysql_query("SELECT c.id_cidade, c.nm_cidade FROM cidade c  
WHERE c.id_uf='".$estado."");  
  
//lista o resultado do banco de dados  
  
$cidades="|-----";  
  
while ($lista_cidades=mysql_fetch_array($result)) {  
  
    $cidades.="\n".$lista_cidades['id_cidade']."|".mb_convert_  
encoding($lista_cidades['nm_cidade'],"utf-8","iso-8859-1");  
  
}  
  
// encerra a conexão  
  
mysql_close();  
  
echo $cidades ;  
  
?>
```

Esse programa realiza a consulta no banco de dados e retorna a informação. A preparação dessa informação é feita por meio de delimitadores, nos quais serão posteriormente quebrados em vetores (função *acrescentaOption*). Uma função do PHP que facilita a escrita do AJAX em HTML 4 é a função *mb\_convert\_encoding*. Essa função converte os dados em uma nova decodificação. Ela garante que a acentuação sairá de forma correta dentro do AJAX.



Procure verificar qual os tipos de codificação possíveis para a função *MB\_convert\_encoding*

Neste ponto já temos o necessário para carregar as informações para efetivar um novo cadastro. Vamos agora montar a função em JavaScript para validar os dados do formulário. Esta função deve ser acrescentada ao programa *inicial.html*.

```
function validarcampos(form) {
    var elementos=form.elements;
    var resp="";
    var enviarcampos="";
    for (var i=0;i<elementos.length; i++) {
        // verifica o tipo de campo para ser testado corretamente
        if (((elementos[i].type=="text") || (elementos[i].type=="hidden")
        || (elementos[i].type=="select-one")) && (elementos[i].getAttribute("obrigatorio")=="sim") && (document.getElementById(elementos[i].getAttribute("id")).value==""))
        {
            // altera o estilo do campo com problema
            elementos[i].style.backgroundColor= "#FF9";
            elementos[i].style.color= "#F00";
            elementos[i].style.fontWeight= "bold";
            resp+="O campo "+elementos[i].getAttribute("descr") +
            é obrigatorio.\n";
        }
        else {
            // garante que os botões não serão alterados
            if ((elementos[i].getAttribute("type")!="submit") &&
            (elementos[i].getAttribute("type")!="reset") && (elementos[i].getAttribute("type")!="button"))
            {
                elementos[i].style.backgroundColor= "#FFF";
                elementos[i].style.color= "#000";
                elementos[i].style.fontWeight= "normal";
            }
        }
    }
}
```

```
// não acrescenta os botões no envio das informações

if ((elementos[i].getAttribute("type")!="submit") && (elementos[i].getAttribute("type")!="reset") && (elementos[i].getAttribute("type")!="button")) {

    // monta a url de envio

    if (enviarcampos=="") enviarcampos=elementos[i].getAttribute("name")+"="+document.getElementById(elementos[i].getAttribute("id")).value;

    else enviarcampos+="&"+elementos[i].getAttribute("name")+"="+document.getElementById(elementos[i].getAttribute("id")).value;

    enviarcampos

}

}

if (resp!="") {

    alert(resp);

    return false;

} else {

    // envia as informações para o AJAX

    carregarDados (enviarcampos, 'salvarDados.php', 'salvar')

}

form.reset();

return false;

}
```

Essa função vasculha em todos os elementos do formulário a fim de identificar quais são os campos obrigatórios e validá-los. A função utiliza um artifício do XML em que foi empregado no HTML. Os campos obrigatórios receberam dois atributos customizados *obrigatorio* e *descr*. Um para informar a obrigatoriedade e o outro a identificação na mensagem de erro. No formulário, fica desta forma:

```
<input type="text" name="nome" id="nome" value="" obrigatorio="sim"
descr="Nome" class="normal"/>
```

Desta forma, a função se torna mais genérica e o controle passa a ser do formulário não mais do código de validação. Outra informação importante dessa função é que são testados os tipos de campos. Dependendo do tipo, pode ser feita uma validação mais apropriada. Nos campos utilizados, foram manipulados os estilos em CSS para ressaltar as informações com problemas de validação. Uma vez validado os campos, os dados são enviados para o AJAX submeter ao programa *salvarDados.php*.

```
<?php

// prepara uma variável de retorno com uma tabela HTML

$retorno = "<table width=\"70%\" align=\"center\""
border="1"><tr><th>Código</th><th>Nome</th><th>Cidade</th><th>UF</th><th>Ação</th></tr>\n";
// inclui os parâmetros para conectar no banco de dados
include("variaveis.bd");

// conecta do banco
$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o
banco de dados");
// vamos selecionar o banco de dados errado o que resultará em um erro
mysql_select_db("prg3");
// realiza uma validação nos dados
if ((@$_GET['nome']!="") && (@$_GET['cidade']!='')) {
    // é feita uma validação ternária, o qual verifica um dado e acrescenta
    um valor a uma variável
    $codigo = @$_GET['codigo']!=""? $_GET['codigo'] : 0;
    // converte para o encode correto e a função addslashes prepara as
    informações para cadastrar em um banco de dados
    $nome = addslashes(mb_convert_encoding(@$_GET['nome'], "iso-
8859-1", "utf-8"));
    $cidade = @$_GET['cidade'];
    // o mysql permite a função replace (deleta e insere a mesma
    informação) quando não existe insere, se existe altera
}
```

```
    mysql_query("replace into pessoa (id_pessoa, nm_pessoa, id_cidade)
value ($codigo, '$nome',$cidade)");

}

// monta uma consulta de retorno na tela

$result= mysql_query("SELECT p.id_pessoa, p.nm_pessoa, c.nm_cidade,
c.id_uf FROM pessoa p, cidade c WHERE p.id_cidade=c.id_cidade");

//lista o resultado do banco de dados

while ($pessoas=mysql_fetch_array($result)) {

    // prepara as informações para mostrar na tela

    $retorno.= "<tr>\n";
    $retorno.= "<td>".$pessoas["id_pessoa"]."</td>\n";
    $retorno.= "<td>".mb_convert_encoding($pessoas["nm_pessoa"],"utf-8", "iso-8859-1")."</td>\n";
    $retorno.= "<td>".mb_convert_encoding($pessoas["nm_cidade"],"utf-8", "iso-8859-1")."</td>\n";
    $retorno.= "<td>".$pessoas["id_uf"]."</td>\n";
    $retorno.= "<td><a href=\"#" onclick=\"carregarDados('codigo='".$pessoas["id_pessoa"]."','carregaDados.php', 'pessoa')\">Alterar</a></td>\n";
    $retorno.= "</tr>\n";
}

// encerra a conexão

mysql_close();

$retorno.= "</table>";

echo $retorno;

?>
```

A informação montada no programa é formatada em uma tabela HTML. Assim, não mostra somente a informação cadastrada, mas todas as informações contidas no banco de dados. Para

retornar essa lista, é necessário acrescentar um desvio na função *escreveDados*.

```
function escreveDados() {
    if ( ajax.readyState == 4) {
        if ( ajax.status == 200) {
            if (tipoDados=="cidade")
                acrescentaOption('cidade',ajax.responseText);
            if (tipoDados=="salvar")
                document.getElementById('pagina').innerHTML =
ajax.responseText;
        }
    }
}
```

Esse desvio é feito para escrever na *tag div* a tabela de retorno. Essas informações abrem a opção para outra funcionalidade: a alteração de dados. Essa funcionalidade utiliza o mesmo mecanismo da funcionalidade de inserir. A diferença é que sabemos qual o código está sendo enviado. A dificuldade aqui é em recuperar as informações. Para diminuir o custo do AJAX em processar as informações, vamos utilizar um único programa para carregar as informações em todos os campos, inclusive a *combobox* de cidade. Para isso, já foi formatado na tabela de resposta um *link* para alterar os dados. Este *link* envia o código da pessoa para o AJAX.

```
<td><a onclick="carregarDados('codigo=1', 'carregaDados.php',
'pessoa')" href="#">Alterar</a></td>
```

O programa *carregaDados.php* realiza a consulta das informações e formata o retorno.

```
<?php

$codigo = $_GET['codigo'];

$pessoa = "";

// inclui os parâmetros para conectar no banco de dados

include("variaveis.bd");

// conecta do banco

$link = mysql_connect($serv, $user, $pass) or die("Erro na conexão com o
banco de dados");

// vamos selecionar o banco de dados errado o que resultará em um erro

mysql_select_db("prg3");

// a consulta é montada e encaminhada ao servidor

$result= mysql_query("SELECT p.id_pessoa, p.nm_pessoa, c.id_cidade,
c.id_uf FROM pessoa p, cidade c WHERE p.id_cidade=c.id_cidade AND
p.id_pessoa='".$codigo);

//lista o resultado do banco de dados

$listPessoa=mysql_fetch_array($result);

// formata os dados de retorno por meio de delimitadores
$pessoa=$listPessoa['id_pessoa'].'.'.$mb_convert_
encoding($listPessoa['nm_pessoa'],"utf-8", "iso-8859-1").'$'.$listPessoa['id_'
cidade'].'.'.$$listPessoa['id_uf'].~';

// realiza a consulta de todas as cidades do estado cadastrado

$result= mysql_query("SELECT c.id_cidade, c.nm_cidade, c.id_uf FROM
cidade c where id_uf='".$listPessoa['id_uf']."'");

$cidades="|-----";

while ($lista_cidades=mysql_fetch_array($result)) {

    // formata o retorno por meio de delimitadores.

    $cidades.="^".$lista_cidades['id_cidade'].|^.$mb_convert_
encoding($lista_cidades['nm_cidade'],"utf-8", "iso-8859-1");

}

$pessoa.=$cidades;

// encerra a conexão

mysql_close();

echo $pessoa ;

?>
```

Veja que neste programa estão sendo utilizadas duas consultas: uma para pessoas e outra para montar a *combobox* cidade. Essa prática reduz o tempo de resposta e garante que as informações estão sendo enviadas de forma correta.

O AJAX recupera a informação, porém para poder atribuir as informações de maneira correta, é necessário realizar mais um desvio na função *escreveDados*.

```
function escreveDados() {
    if ( ajax.readyState == 4) {
        if ( ajax.status == 200) {
            if (tipoDados=="cidade")
                acrescentaOption('cidade',ajax.responseText);
            if (tipoDados=="pessoa")
                carregarPessoa(ajax.responseText);
            if (tipoDados=="salvar")
                document.getElementById('pagina').innerHTML =
ajax.responseText;
        }
    }
}
```

Quando o retorno é destinado à alteração de dados, o tipo deverá ser *pessoa*. Porém, necessitamos de outra função onde é atribuída todas as informações aos seus respectivos campos.

```
function carregarPessoa(retorno) {
    // monta um vetor para o tipo de dados de combobox e texto
    var tipos=retorno.split("~");
    // separa em vetor os dados de texto
    var vl_campos=tipos[0].split("$");
    // atribui os dados para cada campo de formulário
    document.frmpessoa.codigo.value=vl_campos[0];
```

```
document.frmpessoa.nome.value=vl_campos[1];
    // seleciona o estado cadastrado para a pessoa
selecionaCombo(document.frmpessoa.uf, vl_campos[3]);
    // acrescenta as cidades do estado selecionado
acrescentaOption('cidade',tipos[1]);
    //seleciona a cidade cadastrada para a pessoa
selecionaCombo(document.frmpessoa.cidade, vl_campos[2]);
}
```

A função *carregarPessoa* separa em vetores a cadeia de caracteres retornada pelo AJAX. Assim, acrescenta os dados nos campos, além de realizar a chamada para montar a combobox de cidades.

Até aqui, já foram realizadas a inclusão, a alteração dos dados e um retorno formatado dos dados incluídos. Porém, para se tornar mais amigável, é necessário mostrar essa listagem das informações cadastradas mesmo antes de se cadastrar um item novo. Como o programa *salvarDados.php* foi montado para realizar a manipulação dos dados somente quando os dados do formulário forem enviados, podemos utilizar a mesma chamada para listar todas as informações de pessoas. Assim, acrescentamos um botão ao formulário para que seja chamado o programa sem parâmetros.

```
<input type="button" value="Listar" name="listar"
onclick="carregarDados ('', 'salvarDados.php', 'salvar')"/>
```

O programa apenas executará a consulta. Não será realizado nenhum tipo de manipulação de dados, já que os campos do formulário estão em branco.

Você pode perceber que o AJAX em si é extremamente simples. Porém, a programação empregada a ele para manipular os dados se torna extremamente complexa. Qualquer informação que transite por essa camada deve ser observada, monitorada, gerenciada e testada de forma que possa ser controlada e suprimir o máximo de inconsistências possíveis.

## Seção 3 – Método POST no AJAX

O método GET nos garante um grande volume de soluções, porém possui um limitador. O número de caracteres. Esse número é uma limitação da própria URL que não pode ultrapassar 255 caracteres. Portanto, tratando-se de formulários com um volume de dados maior, temos de utilizar outra solução. O método POST do formulário resolve essa situação.

O método POST requer que os dados sejam encapsulados no corpo do formulário. O próprio HTML naturalmente faz isso no seu código. Mas, utilizando o AJAX, o HTML não gerencia o envio, e sim o JavaScript que realiza essa tarefa. Portanto, para nos auxiliar no encapsulamento das informações, utilizamos a função *encodeURI*.

Essa função força a identificação do recurso sobre o HTML. Desta forma, sua aplicabilidade sobre o AJAX a torna quase perfeita na sua utilização.

Vamos ver um exemplo do envio de um formulário simples por meio do método POST. Primeiramente montamos o HTML com o nome de *form.php*

```
<html>
<head>
<title>Enviando formulário POST com PHP e AJAX</title>
<!-- Carrega o arquivo 'script.js' ao iniciar a página! //-->
<script language="javascript" src="script.js" type="text/javascript"></script>
</head>
<body>
<table cellpadding="2" cellspacing="0" width="50%">
<!--
//No evento 'onsubmit' (ao enviar), ele seta o valor dos campos na
função setarCampos() e depois executa a função enviarForm() da página
script.js. Caso o usuário não tenha Javascript instalado/habilitado no
navegador, ele envia o formulário via ACTION.
```

```
-->

<form action="" method="post" onSubmit="setarCampos(this);
enviarForm('processar.php', campos, 'divResultado'); return false;">

<tr>
<td>Nome</td>
<td>
<input name="txtNome" id="txtNome" type="text">
</td>
</tr>

<tr>
<td>Email</td>
<td>
<input name="txtEmail" id="txtEmail" type="text">
</td>
</tr>

<tr>
<td>&ampnbsp</td>
<td>
<input type="submit" value="Enviar">&ampnbsp<input type="reset"
value="Limpar">
</td>
</tr>
</form>
</table>
<div id="divResultado"/>
<script>
//Cria a função com os campos para envio via parâmetro
function setarCampos() {
    // recusros sendo atribuidos
    campos = "txtNome="+encodeURI(document.
getElementById('txtNome').value).toUpperCase()+"&txtEmail="+encodeU
RI(document.getElementById('txtEmail').value);
```

```
}

</script>

</body>

</htm>
```

Veja que as informações estão sendo preparadas para o envio ao AJAX. A função *encodeURI* monta o recurso em uma variável. Essa variável utilizaremos dentro do JavaScript contendo o AJAX. Para este exemplo, vamos chamar de *script.js*.

```
//Cria e atribui à variável global 'navegador' (em caracteres minúsculos) o
// nome e a versão do navegador

var navegador = navigator.userAgent.toLowerCase();
var elemento;

//Cria uma variável global chamada 'xmlhttp'

var xmlhttp;
//Função que inicia o objeto XMLHttpRequest

function objetoXML() {
    //Desvio do Internet Explorer

    if (navegador.indexOf('msie') != -1) {
        var controle = (navegador.indexOf('msie 5') != -1) ? 'Microsoft.
        XMLHTTP' : 'Msxml2.XMLHTTP';

        //Operador ternário que adiciona o objeto padrão do seu
        navegador (caso for o IE) à variável 'controle'

        try {
            //Inicia o objeto no IE
            xmlhttp = new ActiveXObject(controle);
        } catch (e) {}

    } else {
        //Firefox, Safari, Mozilla
        //Inicia o objeto no Firefox, Safari, Mozilla
```

```
        xmlhttp = new XMLHttpRequest();

    }

}

function handleHttpResponse()
{
    if (xmlhttp.readyState == 4)
    {
        if(xmlhttp.status==200)
        {

            //Insere no 'elemento' a página postada
            elemento.innerHTML = xmlhttp.responseText;
        } else {

            //Insere no 'elemento' o texto atribuído
            elemento.innerHTML = 'Página não encontrada!';

        }
    }
}

//Função que envia o formulário

function enviarForm(url, campos, destino) {
    //Atribui à variável 'elemento' o elemento que irá receber a página
    //postada
    elemento = document.getElementById(destino);

    //Executa a função objetoXML()
    objetoXML();

    //Se o objeto de 'xmlhttp' não estiver true
    if (!xmlhttp) {

        //Insere no 'elemento' o texto atribuído
        elemento.innerHTML = 'Impossível iniciar o objeto
        XMLHttpRequest.';
```

```

        return;

    } else {
        //Insere no 'elemento' o texto atribuído
        elemento.innerHTML = 'Carregando...';

    }

    //Abre a página que receberá os campos do formulário
    xmlhttp.open('POST', url, true);
    xmlhttp.onreadystatechange = handleHttpResponse;
    //altera os cabeçalhos da página de resposta
    xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xmlhttp.setRequestHeader("Content-length", 2);
    //Envia o formulário com dados da variável 'campos' (passado por
    parâmetro)
    xmlhttp.send(campos);
}

```

Como você pode perceber, houve poucas modificações entre o método GET e o método POST. Porém, essas pequenas alterações nos auxiliam no desenvolvimento. Para finalizar o exemplo, vamos montar o arquivo *processar.php*.

```

<?php
//Determina o tipo da codificação da página
header("content-type: text/html; charset=iso-8859-1");

//Extrai os dados do formulário
extract($_POST);

//Verifica se algum nome foi digitado
$nome = ($txtNome != "") ? $txtNome : "desconhecido";
//Verifica se algum email foi digitado
$email = ($txtEmail != "") ? $txtEmail : "desconhecido";

```

```
//Retorna com a resposta  
echo "Olá <b>".$nome."</b>, seu email é: <a href='mailto:".$email."><b>".$email."</b></a>";  
?>
```

Realizando esse teste você poderá verificar que, mesmo alterando o cabeçalho da página, possivelmente ocorrerá um erro de acentuação onde não serão identificadas as acentuações. Quando acontece isso, troca-se as palavras por caracteres estranhos e indecifráveis. Mesmo nessa situação, podemos recorrer a alguns artifícios que nos auxiliam em ajustar a acentuação.

Poderíamos estar utilizando duas funções para ajustar esse problema. A primeira é a *mb\_convert\_encoding*, que converte uma codificação para outra. A segunda seria a função *addslashes*: ela interpreta caracteres especiais como aspas simples e duplas, barra invertida, entre outras. Desta forma, trocaríamos a linha.

```
$nome = ($txtNome != "") ? $txtNome : "desconhecido";
```

Por esta linha:

```
$nome = ($txtNome != "") ? strtoupper(addslashes(mb_convert_encoding($txtNome,"iso-8859-1","utf-8")) : "desconhecido";
```

Ainda foi acrescentada a função *strtoupper* para que fosse alterado o texto para caixa-alta.

Outro exemplo que é muito utilizando é a **busca dinâmica**, que consiste em realizar uma busca a um banco de dados enquanto o usuário digita letra a letra. A sua aplicação é muito utilizada para selecionar rapidamente as informações desejadas. Neste exemplo, vamos nos ater em buscar as informações.

Primeiramente vamos construir um banco de dados com as informações já cadastradas. Vamos utilizar o banco *prg3* e criar uma tabela chamada *pessoa*.

```

CREATE TABLE `pessoa` (
    `id_pessoa` int(10) unsigned NOT NULL auto_increment,
    `nm_pessoa` varchar(45) default NULL,
    PRIMARY KEY (`id_pessoa`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `pessoa` (`id_pessoa`, `nome`) VALUES (1,'ALEXANDRE VIEIRA DE OLIVEIRA');

INSERT INTO `pessoa` (`id_pessoa`, `nome`) VALUES (2,'ALEXANDRE TESTANDO');

INSERT INTO `pessoa` (`id_pessoa`, `nome`) VALUES (3,'WAGNER DOS SANTOS');

INSERT INTO `pessoa` (`id_pessoa`, `nome`) VALUES (4,'VIRGINIA');

INSERT INTO `pessoa` (`id_pessoa`, `nome`) VALUES (5,'PEDALA ROBINHO');

INSERT INTO `pessoa` (`id_pessoa`, `nome`) VALUES (6,'MASSA DEMAIS');

```

Após carregar as informações, passamos ao desenvolvimento do código. Vamos criar o arquivo *inicio.php*:

```

<html>

<head>
    <autor>Alexandre Vieira</autor>
    <site>http://www.codigofonte.net/dicas/php/170_busca-dinamica-utilizando-ajax</site>
    <title>BUSCA DIN&Acirc;MICA</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <script src="script.js"></script>
<script>
function pesquisa(valor)
{
    //FUNÇÃO QUE MONTA A URL E CHAMA A FUNÇÃO AJAX
    campos = "valor="+encodeURIComponent(valor);

```

```
url="busca_nome.php";
ajax(url,campos);
}
</script>
</head>

<body>
<table width="657" border="0" bgcolor="#f7f7f7">
<tr>
<td align="center" bgcolor="#CCCCCC">
<strong>
<font color="#FF0000" size="1" face="Verdana, Arial, Helvetica, sans-serif">
<-BUSCA DINÂMICAMICA ->
</font>
</strong>
</td>
</tr>
<tr>
<td>
<font size="1" face="Verdana, Arial, Helvetica, sans-serif"><strong>DIGITE AQUI O NOME DA PESSOA</strong></font>;
<!-- NOTE QUE A CADA TECLA PRESSIONADA É CHAMADA A
FUNÇÃO PESQUISA PASSANDO O QUE O USUÁRIO TEM DIGITADO ATÉ O
MOMENTO -->
<input type="text" name="nome" onKeyPress="pesquisa(this.value)">
</td>
</tr>
<tr>
<td>
<!-- AQUI SERÁ APRESENTADO O RESULTADO DA BUSCA DINÂMICA,
OU SEJA, OS NOMES -->
<div id="pagina">
```

```

</div>
</td>
</tr>
</table>

</body>
</html>
```

Fonte: <[www.codigofonte.net/dicas/php/170\\_busca-dinamica-utilizando-ajax](http://www.codigofonte.net/dicas/php/170_busca-dinamica-utilizando-ajax)>.

Veja que não foi necessário montar a *tag form*. Essa omissão foi proposital para que você possa perceber que o AJAX gerencia o tipo de método a ser enviado para o servidor. Agora, basta montar o arquivo *script.js*:

```

// JavaScript Document

// FUNÇÃO RESPONSÁVEL DE CONECTAR A UMA PÁGINA EXTERNA, NO
// NOSSO CASO A BUSCA_NOME.PHP

// E RETORNAR OS RESULTADOS

function ajax(url,campos) {
    req = null;

    // Procura por um objeto nativo (Mozilla/Safari)
    if (window.XMLHttpRequest) {

        req = new XMLHttpRequest();
        req.open("POST",url,true);
        req.onreadystatechange = processReqChange;

        req.setRequestHeader("Content-type", "application/x-www-form-
        urlencoded");

        req.setRequestHeader("Content-length", 2);
        req.send(campos);

        // Procura por uma versão ActiveX (IE)
    } else if (window.ActiveXObject) {

        req = new ActiveXObject("Microsoft.XMLHTTP");
```

```
if (req) {  
    req.open("POST",url,true);  
    req.onreadystatechange = processReqChange;  
    req.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
    req.setRequestHeader("Content-length", 2);  
    //Envia o formulário com dados da variável 'campos' (passado por parâmetro)  
    req.send(campos);  
}  
}  
}  
  
function processReqChange() {  
    // apenas quando o estado for "completado"  
    if (req.readyState == 4) {  
        // apenas se o servidor retornar "OK"  
        if (req.status ==200) {  
            // procura pela div id="pagina" e insere o conteudo  
            // retornado nela, como texto HTML  
            document.getElementById('pagina').innerHTML = req.responseText;  
        } else {  
            alert("Houve um problema ao obter os dados:" + req.statusText);  
        }  
    }  
}
```

Fonte: <[www.codigofonte.net/dicas/php/170\\_busca-dinamica-utilizando-ajax](http://www.codigofonte.net/dicas/php/170_busca-dinamica-utilizando-ajax)>.

Permanecemos com o AJAX de forma limpa, sem nenhuma interferência sobre a codificação. Montamos então o arquivo *busca\_nome.php*:

```
<?php

if(!empty($_POST["valor"])) {

// O CAMPO VALOR CONTERÁ O QUE O USUÁRIO DIGITOU ATÉ O
// MOMENTO..

// CONECTA AO BANCO COLOCA PARAMENTROS IP,USUÁRIO,SENHA
$conexao=mysql_connect("localhost","web","123");

//SELECIONA O BANCO DE DADOS QUE VAI USAR
mysql_select_db("prg3");

// EXECUTA A INSTRUÇÃO SELECT PASSANDO O QUE O USUÁRIO DIGITOU
$sql="select * from pessoa where nm_pessoa like '".$_POST[valor]%"';

$resultado=mysql_query($sql) or die (mysql_error());

//VERIFICA A QUANTIDADE DE REGISTROS RETORNADOS
$linhas=mysql_num_rows($resultado);

if($linhas>0){

//EXECUTA UM LOOP PARA MOSTRAR OS NOMES DAS PESSOAS
// VALE LEMBRAR QUE TODOS ESSES RESULTADOS SERÃO MOSTRADOS
// DENTRO DA PÁGINA INDEX.PHP

// DENTRO DO DIV 'PAGINA'
echo "<ul>\n";
while($pegar=mysql_fetch_array($resultado))
echo "<li>$pegar[nm_pessoa]</li>\n";
}
echo "</ul>\n";
}

?>
```

Fonte: <[www.codigofonte.net/dicas/php/170\\_busca-dinamica-utilizando-ajax](http://www.codigofonte.net/dicas/php/170_busca-dinamica-utilizando-ajax)>.

Apesar de exigir uma certa complexidade, a utilização do AJAX possui uma codificação simples, necessitando apenas de um pequeno número de decodificação.



---

Procure testar se está sendo enviado realmente pelo método POST utilizando a função `echo var_dump($_POST);` no arquivo `busca_nome.php`.

---

## ***Upload de arquivos***

Apesar das vantagens, o AJAX sofre com a sua própria tecnologia. Você deve se lembrar que o AJAX não encaminha informações ao servidor por meio da postagem de formulário. Em vez disso, ele cria uma camada de comunicação com o servidor.

Essa característica possui um custo e, dependendo da aplicação, pode ser muito alto. A funcionalidade de enviar arquivos para o servidor é feita por meio da postagem do HTML. Isso significa que o *browser* decodifica o arquivo para o HTML poder encaminhar a postagem. O HTML processa o arquivo para o servidor.

O AJAX não possui uma postagem. Então como proceder nessa situação? Atualmente não possui uma solução para o AJAX realizar o *upload* de arquivos. Em contrapartida, existem diversas soluções que simulam essa atividade pelo AJAX, uma delas é a utilização de *iFrames*. Essa utilização pode acarretar riscos já que não são todos os *browsers* que tratam o *iFrame* de forma concisa. Veja o seguinte exemplo de sua utilização.

```
1. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//  
EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.  
dtd">  
2. <html xmlns="http://www.w3.org/1999/xhtml">  
3. <head>  
4. <!--Script desenvolvido pelo http://www.ajaxf1.com/-->  
5. <meta http-equiv="Content-Type" content="text/html;  
charset=utf-8" />  
6. <title>Max's AJAX File Uploader</title>  
7. <link href="style/style.css" rel="stylesheet" type="text/css" />  
8.  
9. <script language="javascript" type="text/javascript">  
10.<!--  
11.function startUpload(){  
12. document.getElementById('f1_upload_process').style.  
    visibility = 'visible';  
13. document.getElementById('f1_upload_form').style.visibility  
    = 'hidden';  
14. return true;  
15.}  
  
16.function stopUpload(success){  
17. var result = "  
18. if (success == 1){  
19.     result = '<span class="msg">The file was uploaded  
        successfully!</span><br/><br/>';  
20. }  
21. else {  
22.     result = '<span class="emsg">There was an error during file  
        upload!</span><br/><br/>';  
23. }  
24. document.getElementById('f1_upload_process').style.  
    visibility = 'hidden';  
25. document.getElementById('f1_upload_form').innerHTML =  
    result + '<label>File: <input name="myfile" type="file" size="30"  
    /></label><label><input type="submit" name="submitBtn"  
    class="sbtn" value="Upload" /></label>';  
26. document.getElementById('f1_upload_form').style.visibility  
    = 'visible';  
27. return true;  
28.}  
29.//-->  
30.</script>  
31.</head>
```

```
32.<body>
33.    <div id="container">
34.        <div id="header"><div id="header_left"></div>
35.        <div id="header_main">Max's AJAX File Uploader</
36.            <div id="header_right"></div></div>
37.        <div id="content">
38.            <form action="upload.php" method="post"
39.                enctype="multipart/form-data" target="upload_target"
40.                onsubmit="startUpload();">
41.                <p id="f1_upload_process">Loading...<br/><br/></p>
43.                <p id="f1_upload_form" align="center"><br/>
44.                    <label>File:
45.                        <input name="myfile" type="file" size="30" />
46.                    </label>
47.                    <label>
48.                        <input type="submit" name="submitBtn"
49.                            class="sbtn" value="Upload" />
50.                    </label>
51.                </p>
52.            <iframe id="upload_target" name="upload_target"
53.                src="#" style="width:0;height:0; border:0px solid #fff;"></iframe>
54.        </form>
55.    </div>
56.    <div id="footer"><a href="http://www.ajaxf1.com"
57.        target="_blank">Powered by AJAX F1</a></div>
58. </div>
59.</body>
60.</html>
```

Fonte: <[www.ajaxf1.com](http://www.ajaxf1.com)>.

Veja que o diferencial está na linha 39 que a ponta o “alvo” do formulário sendo o *iFrame*. Esta tag permite a utilização de um bloco da página como sendo uma página independente da original.

O arquivo que processa o arquivo pode ser escrito desta forma:

```

<?php

// Edit upload location here

$destination_path = getcwd().DIRECTORY_SEPARATOR;

$result = 0;

$target_path = $destination_path . basename( $_FILES['myfile']
['name']);

if(@move_uploaded_file($_FILES['myfile']['tmp_name'], $target_path)) {

    $result = 1;

}

sleep(1);

?>

<script language="javascript" type="text/javascript">window.top.
window.stopUpload(<?php echo $result; ?>);</script>

```

Fonte: <[www.ajaxf1.com](http://www.ajaxf1.com)>.

Todo o processo percorre de forma que não utiliza o AJAX, mas para o usuário permanece essa ilusão. O *script* completo se encontra postado na midiateca.

## Seção 4 – Bibliotecas

O desenvolvimento de *softwares* é traçado por diversos tipos de ferramentas que facilitam e enriquecem o trabalho. Os *frameworks* tanto podem ser *softwares* que criam programas com uma maior facilidade como *scripts* contendo diversos recursos.

Quando um *framework* não cria um programa, mas possibilita a utilização de soluções com o intuito de agilizar o desenvolvimento, recebem o nome de **bibliotecas**.

As bibliotecas possuem a característica de reduzir a repetição de códigos, assim como agrupar funcionalidades em comuns. Desta forma, é possível compactar diversas funcionalidades e reutiliza em diversos projetos. Assim, procura-se reduzir a quantidade de codificação desenvolvida em cada projeto. Sem falar na confiabilidade de um código testado constantemente em diversas aplicações.

## Dojo

A biblioteca Dojo possui uma distribuição gratuita e foi desenvolvida em JavaScript. O Dojo possui a proposta de reduzir o tratamento de solicitações assíncronas a uma única chamada de função, além de oferecer diversas funcionalidades DOM para diversos navegadores.

DOM é uma sigla de Modelo de Objetos de Documentos. Esse modelo é uma especificação do consórcio W3C que independe da plataforma e linguagem. A especificação define os objetos que podem alterar dinamicamente sua estrutura, o conteúdo e os estilos dos documentos eletrônicos. O DOM oferece um padrão para acessar os elementos do documento trabalhando com eles separadamente.

Por se tratar de uma biblioteca, a sua utilização é extremamente simples. Não é necessária uma instalação. Necessita apenas de uma chamada por meio de uma *tag script* e em seguida invocar as funções.



---

O Dojo possui uma documentação própria em que você pode estar estudando sobre como utilizar as funcionalidades. A página oficial dessa ferramenta é <[www.dojotoolkit.org](http://www.dojotoolkit.org)>.

---

## Sarissa

Sarissa é uma biblioteca direcionada para a manipulação do XML em JavaScript. Possui suporte a componentes ActiveX MSXML do Internet Explorer, Mozilla, Opera, Konqueror e Safari.

O Sarissa utiliza o padrão Adapter para criar um objeto *XMLHttpRequest* baseado em JavaScript para os navegadores que não possui esse objeto nativo. Desta forma, ele padroniza os chamados realizando as devidas adaptações. Portanto, o código a seguir deverá funcionar para todos os *browsers*.

```
var xhr = new XMLHttpRequest();
xhr.open("GET","dados.xml");
xhr.onreadystatechange = function {
    if (xhr.readyState == 4) {
        alert(xhr.responseXML);
    }
}
```

## Prototype

Esta biblioteca também é escrita em JavaScript, porém possui seu foco nas funcionalidades desta linguagem com o intuito de suportar uma programação orientada a objetos. Apesar de sua leitura ser muito difícil, a sua utilização é simples e direta.

O Prototype por muitas vezes é considerado como uma biblioteca geral para desenvolver bibliotecas. Os desenvolvedores AJAX têm uma probabilidade maior de utilizarem alguma biblioteca com base na Prototype. Essa característica descende de ela ser construída com base na programação orientação objeto. Assim, todos os recursos dessa metodologia são herdados pela biblioteca. Essa característica permite uma customização mais natural e dinâmica.



---

O Prototype é uma biblioteca poderosa e você pode aprender mais sobre ela consultando a página oficial <[www.prototypejs.org](http://www.prototypejs.org)>.

---

Existem diversas bibliotecas que podem auxiliar no desenvolvimento. Procure sempre escolher as que possuem uma boa documentação. O grupo que mantém biblioteca deve conter um controle de versão, assim você saberá que os *bugs* estão sendo corrigidos. De forma geral, procure sempre testar a biblioteca escolhida. Caso tenha necessidade de alterar alguma funcionalidade da biblioteca, certifique-se se essa alteração não se trata de uma casualidade do projeto, assim deve permanecer no projeto. Bibliotecas são genéricas.



---

Bibliotecas, em geral, têm a obrigação de reduzir o nosso esforço em construir códigos repetitivos e maçantes. As casualidades devem ser tratadas dentro do próprio projeto de desenvolvimento.

---



## Síntese

Nesta unidade você viu uma tecnologia muito útil no desempenho de aplicações Web. O AJAX é um conjunto de aplicações destinado em promover uma agilidade às páginas Web semelhante a uma aplicação *desktop*. Vimos como o AJAX é complexo e, em contrapartida, possui um código enxuto com pouca codificação, além de sua limitação no *upload* de arquivos e o quanto os *frameworks* facilitam o nosso trabalho como desenvolvedores.



## Atividades de autoavaliação

Ao final de cada unidade, você será provocado a realizar atividades de autoavaliação. O gabarito está disponível no final do livro-didático, mas se esforce para resolver as atividades sem ajuda do gabarito, pois assim você estará estimulando a sua aprendizagem.

- 1) Desenvolva uma aplicação utilizando o AJAX e um banco de dados.  
Poderá construir um cadastro de pessoas, onde, ao digitar parcialmente o nome ou usuário, tenha a opção de selecionar o nome e carregar as informações para a tela.



## Saiba mais

Se você desejar, aprofunde os conteúdos estudados nesta unidade ao consultar as seguintes referências:

**W3Schools. W3Schools Online Web Tutorials**, 2010.  
Disponível em: <[www.w3schools.com](http://www.w3schools.com)>. Acesso em: 01 mar. 2010.

**W3C. World Wide Web Consortium**, 2010. Disponível em: <[www.w3.org](http://www.w3.org)>. Acesso em: 2 mar. 2010.

**AJAX F1**, 2010. Disponível em: <[www.ajaxf1.com](http://www.ajaxf1.com)>. Acesso em: 20 maio 2010.

**iMasters**, 2010. Disponível em: <<http://imasters.uol.com.br>>. Acesso em: 16 maio 2010.

**CodigoFonte.net**, 2010. Disponível em: <[www.codigofonte.net](http://www.codigofonte.net)>. Acesso em: 15 jul. 2010.





## Para concluir o estudo

Ao se dedicar ao estudo das unidades apresentadas neste livro didático, você teve a oportunidade de interagir e consolidar alguns conceitos sobre o desenvolvimento de aplicações para Web assim como harmonizar as diferentes linguagens que compõe um projeto deste tipo.

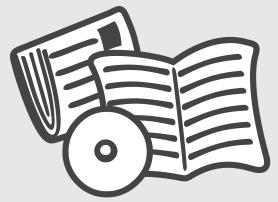
Conhecer a linguagem e saber utilizá-la não significa um sucesso no projeto. Detalhe o melhor possíveis todos os itens da sua aplicação. Destaque sempre as regras de negócio o qual você está buscando uma solução ou melhoria. Um planejamento bem estruturado e detalhado representa 60% do projeto.

Mantenha sempre em foco onde cada linguagem é melhor empregada. Não desperdice recursos tentando construir uma alternativa onde já existam diversas soluções. Perder alguns minutos com uma pesquisa pela rede, normalmente economizam muitas horas em desenvolvimentos desnecessários.

Linguagens como Javascript e PHP possuem uma vasta bibliografia e comunidades onde são disponibilizados diversos scripts que possivelmente irão auxiliá-lo em seu problema. Porém, você perceberá que o entendimento destes scripts é indispensável ao se adotar como solução.

Boa sorte e sucesso!!!





# Referências

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML.** 2. ed. Rio de Janeiro: Campus, 2007.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML, guia do usuário.** Rio de Janeiro : Elsevier, 2000.

GUEDES, Gilleanes T. A. **UML: Uma Abordagem Prática.** São Paulo: Novatec, 2004.

FOWLER, Martin. **UML essencial** : um breve guia para a linguagem-padrão de modelagem de objetos. 3.ed.. Porto Alegre, RS : Bookman, 2005.

DICIONÁRIO ELETRÔNICO MICHAELIS. **Moderno dicionário da língua portuguesa.** São Paulo: Melhoramentos, 2009. Disponível em: <<http://michaelis.uol.com.br/moderno/portugues/index.php?lingua=portugues-portugues&palavra=intera%E7%E3o>>. Acesso em: 05/10/2010.

OMG. **Introduction to Unified Modeling Language.** Disponível em: <[http://www.omg.org/gettingstarted/what\\_is.uml.htm](http://www.omg.org/gettingstarted/what_is.uml.htm)>. Acessado em 08-mar-2010.

OMG-About. **About the Object Management Group.** Disponível em: <<http://www.omg.org/gettingstarted/gettingstartedindex.htm>>. Acessado em 10-out-2010.

SPARXSYSTEMS. **UML 2 Tutorial.** Disponível em: <[http://www.sparxsystems.com/resources/uml2\\_tutorial/](http://www.sparxsystems.com/resources/uml2_tutorial/)>. Acessado em 10-out-2010.

WAZLAWICK, Raul Sidnei. **Análise e projeto de sistemas de informação orientados a objetos.** 1. ed. Rio de Janeiro: Elsevier, 2004.

**Manual PHP**, 2010. Disponível em: [http://br2.php.net/manual/pt\\_BR/index.php](http://br2.php.net/manual/pt_BR/index.php)> Acessado em 05 abril 2010.

PHP Brasil, 2010. Disponível em: <<http://www.phpbrasil.com>>. Acessado em 05 abril 2010.

**Truques e Dicas - Tutoriais de Javascript**, 2010. Disponível em: <<http://www.truquesedicas.com/tutoriais/php/index.htm>>. Acessado em 02 mar 2010.

**Manual do MySQL**, 2010. Disponível em: <<http://dev.mysql.com/doc/refman/5.1/en/index.html>> Acessado em 05 abril 2010.

**Código Fonte**, 2010. Disponível em <<http://codigofonte.uol.com.br/artigos/javascript-dhtml>> Acessado em 01 mar 2010.

**jsfromhell.com JavaScript Repository**, 2010. Disponível em <<http://jsfromhell.com/>>. Acessado em 15/03/2010.

W3Schools. **W3Schools Online Web Tutorials**, 2010. Disponível em <<http://www.w3schools.com>> Acessado em 01 mar 2010.

W3C. **World Wide Web Consortium**, 2010. Disponível em <<http://www.w3.org>>. Acessado em 02 mar 2010.

**Dicas-I**, 2010. Disponível em <<http://www.dicas-i.com.br>>. Acessado em 16 mai 2010.

**iMasters**, 2010. Disponível em <<http://imasters.uol.com.br>>. Acessado em 16 mai 2010.

**Plugmasters**, 2010. Disponível em <<http://www.plugmasters.com.b>>. Acessado em 15 mai 2010.

**AJAX F1**, 2010. Disponível em <<http://www.ajaxf1.com>>. Acessado em 20 mai 2010.

**CodigoFonte.net**, 2010. Disponível em <<http://www.codigofonte.net>>. Acessado em 15 julho 2010.



## Sobre o professor conteudista

**Edson Orivaldo Lessa Junior** é especialista em Engenharia de projetos de software pela Universidade do Sul de Santa Catarina – UNISUL. Professor da Unisul desde 2008 em disciplinas oferecidas pelos cursos de Ciências da Computação, Gestão da Tecnologia da Informação e Web Design e Programação. Participou de projetos de desenvolvimentos em Gestão Financeira, Framework, Sistemas distribuídos.





# Respostas e comentários das atividades de autoavaliação

## Unidade 1

1. Uma das principais vantagens da aplicação Webtop é a sua portabilidade. Praticamente, o usuário possui acesso ao seu *desktop* em qualquer lugar que tenha acesso à internet. Outra vantagem é a centralização para manutenção, qualquer novo recurso implementado para a aplicação Webtop se propagará para todos os usuários. Sem falar de uma vantagem primordial, como toda a aplicação está em um servidor, o *backup* das informações é muito mais fácil.

Uma desvantagem é que tudo depende da rede. Caso uma conexão de dados não possua uma velocidade adequada, reflete diretamente no desempenho da Webtop. Outra desvantagem é a compatibilidade com as aplicações. Muitas aplicações não são compatíveis com este modelo o que torna muitas vezes inviável a sua utilização.

2.

1 – Linguagem <i>Server-side</i>	( 1 )	PostgreSQL
2 – Linguagem <i>Client-side</i>	( 2 )	HTML
	( 2 )	Oracle
	( 2 )	SQLite
	( 1 )	Javascript
	( 1 )	Microsoft Silverlight
	( 2 )	PHP
	( 2 )	MySQL
	( 2 )	ASP
	( 2 )	DB2
	( 2 )	.NET
	( 2 )	CGI
	( 2 )	Perl
	( 2 )	Java
	( 1 )	CSS
	( 2 )	Python
	( 1 )	Microsoft SQL Server
	( 1 )	Ruby
	( 1 )	Firebird
	( 1 )	ColdFusion
	( 1 )	Websphere
	( 2 )	Flash

3. Não há limite, logicamente eles não podem ser redundantes na mesma tag. Mas é possível utilizar quantas forem necessárias e criar atributos novos para uma personalização de aplicação.

## Unidade 2

1.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exercicio Um</title>
</head>

<script language="javascript" type="text/javascript">

function validanumero(numero) {
    // função para validar se um número é valido
    if (numero.length == 0) {
        return false;
    } else {
        // verifica se todos os caracteres são números
        for (i = 0; i < numero.length; i++)
            if (!(numero.charAt(i) >= "0") && (numero.charAt(i) <=
"9")))
                return false;
    }
    return true;
}

function validar(form) {
    //Variáveis de mensagem de erro padrão e controle de erro
    var erro="ATENÇÃO\n\n";
    var msgerro=false;
```

```
if (form.txt_nome.value=="") {  
    erro+="O Campo Nome é obrigatório\n";  
    msgerro=true;  
}  
  
if (form.txt_nomemae.value==""){  
    erro+="O Campo Nome da Mãe é obrigatório\n";  
    msgerro=true;  
}  
  
var idade=form.txt_idade.value;  
  
if (validanumero(idade)){  
    // testa para verificar se a idade está contida no intervalo correto  
    if ((parseInt(idade)<10) || (parseInt(idade)>120)) {  
        erro+="O Campo Idade deve conter uma idade entre 10 e  
120 anos\n";  
        msgerro=true;  
    }  
}  
}  
  
} else {  
    erro+="O Campo Idade deve conter somente números  
inteiros\n";  
    msgerro=true;  
}  
  
var email=form.txt_email.value;  
  
if (email=="") {  
    erro+="O Campo E-mail é obrigatório\n";  
    msgerro=true;  
}  
}  
  
} else {  
    // verifica se existe um caractere @ e não se encontra na primeira  
    posição  
    // verifica se existe um caractere entre o @ e o .  
    // verifica se o . não é o último caractere  
    if ((email.indexOf("@")<0) || (email.indexOf(".")<=(email.  
indexOf("@")+1)) || ((email.indexOf(".")+1)>=email.length)) {
```

```
        erro+="O Campo E-mail deve conter um e-mail válido\n";
        msgerro=true;
    }
}
if (msgerro) {
    alert(erro);
    return false;
}
return true;
}

</script>

<body>
<h1>Cadastro</h1>
<p>
<form name="cadpessoa" id="cadpessoa" method="post" action="" onsubmit="return validar(this);">
    <table width="338">
        <tr>
            <td width="163"><label for="txt_nome">Nome:</label></td>
            <td width="163"><input type="text" name="txt_nome" id="txt_nome" value="" /></td>
        </tr>
        <tr>
            <td>Endereço</td>
            <td><input type="text" name="txt_endereco" id="txt_endereco" value="" /></td>
        </tr>
        <tr>
            <td><label for="txt_nomemae">Nome da M  e</label></td>
            <td><input type="text" name="txt_nomemae" id="txt_nomemae" value="" /></td>
        </tr>
    </table>
</form>

```

```
<tr>
    <td><label for="txt_idade">Idade</label></td>
    <td><input type="text" name="txt_idade" id="txt_idade" value="" /></td>
</tr>
<tr>
    <td><label for="txt_email">E-mail</label></td>
    <td><input type="text" name="txt_email" id="txt_email" value="" /></td>
</tr>
<tr>
    <td align="center"><input type="submit" name="btn_validar" value="Validar" /></td>
    <td align="center"><input type="reset" name="btn_limpar" value="Limpar" /></td>
</tr>
</table>
</form>
</p>
</body>
</html>
```

2.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exercício dois</title>
</head>
<script language="javascript" type="text/javascript">
```

```
function inserir(texto) {  
    //converte as quebras de linhas para a tag <br> do HTML  
    while (texto.indexOf("\n")>=0)  
        texto=texto.replace("\n","<br />");  
    //escreve do elemento que contém o Id div_result  
    document.getElementById("div_result").innerHTML=texto;  
}  
</script>  
<body>  
<form name="forminsert" id="forminsert">  
    <label for="txa_insert">Insira o texto abaixo:</label><br />  
    <textarea name="txa_insert" id="txa_insert" cols="50" rows="10"  
        onkeyup="inserir(this.value);"></textarea>  
    <br />  
    O Resultado:<br />  
    <div id="div_result" style="color:#009">  
    </div>  
</form>  
</body>  
</html>
```

### Unidade 3

1.  
<<<arquivo exercicio1.php >>>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml">  
  
<head>  
  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
  
<title>Exercício 1 envio Form</title>
```

```
</head>

<body>

<form name="formulario" target="_self" action="recebeform.php"
method="post">

<table align="left" cellpadding="0" cellspacing="0" border="1">
<tr>
<td>Nome</td>
<td><input type="text" name="txt_nome" id="txt_nome" value="" /></td>
</tr>
<tr>
<td>Endereço</td>
<td><input type="text" name="txt_endereco" id="txt_endereco" value="" /></td>
</tr>
<tr>
<td>Nome da mãe</td>
<td><input type="text" name="txt_nomemae" id="txt_nomemae" value="" /></td>
</tr>
<tr>
<td>Idade</td>
<td><input type="text" name="txt_idade" id="txt_idade" value="" /></td>
</tr>
<tr>
<td>E-mail</td>
<td><input type="text" name="txt_email" id="txt_email" value="" /></td>
</tr>
<tr>
```

```
<td><input type="submit" name="btn_enviar" id="btn_enviar"
value="Enviar" /></td>

<td><input type="reset" name="btn_limpar" id="btn_limpar"
value="Limpar" /></td>

</tr>

</table>

</form>

</body>

</html>
```

<<< arquivo recebeform.php >>>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Recebe Formulário</title>

</head>

<body>

<?php

function erro($tipo, $msg="Sem informação"){

    switch ($tipo) {

        case 1:

            echo "<h1>Formulário não enviado!</h1>";

            break;

        case 2:

            echo "<h1>Erro nos valores do formulário!</h1>";

            echo $msg;

            break;

    }

}
```

```
echo "<a href=\"exercicio1.php\">Retornar ao formulário";  
  
}  
  
// Verifica se o formulário foi enviado  
  
if (@$_POST["btn_enviar"]=="") erro(1);  
  
else {  
  
    // variável de controle  
  
    $controle=false;  
  
    if ($_POST["txt_nome"]=="") {  
  
        $validacampos="O campo nome deve ser preenchido<br />\n";  
  
        $controle=true;  
    }  
  
    if ($_POST["txt_endereco"]=="") {  
  
        $validacampos.="O campo endereço deve ser preenchido<br  
        />\n";  
  
        $controle=true;  
    }  
  
    if ($_POST["txt_nomemae"]=="") {  
  
        $validacampos.="O campo nome da mãe deve ser preenchido<br  
        />\n";  
  
        $controle=true;  
    }  
  
    if ($_POST["txt_idade"]=="") {  
  
        $validacampos.="O campo idade deve ser preenchido<br />\n";  
  
        $controle=true;  
    } else {  
  
        if (intval($_POST["txt_idade"])<3) {  
  
            $validacampos.="A idade deve ser maior que dois  
anos<br />\n";  
  
            $controle=true;  
        }  
    }  
}
```

```
}

if ($_POST["txt_email"]=="") {

    $validacampos.="O campo e-mail deve ser preenchido<br />\n";
    $controle=true;

} else {

    /*

        strpos verifica a posição que um caractere aparece em uma string
        strrpos verifica a última posição que um caractere aparece
    */

    if (((strpos($_POST["txt_email"], "@")==false) &&
(strpos($_POST["txt_email"], ".")==false)) || (strrpos($_POST["txt_email"], ".")<strpos($_POST["txt_email"], "@")))) {

        $validacampos.="O campo e-mail deve conter um e-mail
válido<br />";

        $controle=true;
    }
}

if ($controle==true)

    erro(2,$validacampos);

else {

    ?>

<table align="left" cellpadding="2" cellspacing="0" border="1">

<tr>

<td>Nome</td>

<td><?=$_POST['txt_nome']?></td>

</tr>

<tr>

<td>Endereço</td>

<td><?=$_POST['txt_endereco']?></td>

</tr>

<tr>
```

```
<td>Nome da mãe</td>
<td><?=$_POST["txt_nomemae"]?></td>
</tr>
<tr>
<td>Idade</td>
<td><?=$_POST["txt_idade"]?></td>
</tr>
<tr>
<td>E-mail</td>
<td><?=$_POST["txt_email"]?></td>
</tr>
</table>

<?php
}

}

?>
</body>
</html>
```

2.

<<< arquivo recebeform.php >>>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Recebe Formulário</title>
</head>
```

```
<body>

<?php

function erro($tipo, $msg="Sem informação"){

    switch ($tipo) {

        case 1:

            echo "<h1>Formulário não enviado!</h1>";

            break;

        case 2:

            echo "<h1>Erro nos valores do formulário!</h1>";

            echo $msg;

            break;

    }

    echo "<a href=\"exercicio1.php\">Retornar ao formulário";

}

// Verifica se o formulário foi enviado

if (@$_POST["btn_enviar"]=="") erro(1);

else {

    // variável de controle

    $controle=false;

    if ($_POST["txt_nome"]=="") {

        $validacampos="O campo nome deve ser preenchido<br />\n";

        $controle=true;

    }

    if ($_POST["txt_endereco"]=="") {

        $validacampos.="O campo endereço deve ser preenchido<br />\n";

        $controle=true;

    }

    if ($_POST["txt_nomemae"]=="") {
```

```
$validacampos.="O campo nome da mãe deve ser preenchido<br />\n";
$controle=true;
}

if ($_POST["txt_idade"]=="") {
    $validacampos.="O campo idade deve ser preenchido<br />\n";
    $controle=true;
} else {
    if (intval($_POST["txt_idade"])<3) {
        $validacampos.="A idade deve ser maior que dois
anos<br />\n";
        $controle=true;
    }
}

if ($_POST["txt_email"]=="") {
    $validacampos.="O campo e-mail deve ser preenchido<br />\n";
    $controle=true;
} else {
    /*
        strpos verifica a posição que um caractere aparece em uma string
        strrpos verifica a última posição que um caractere aparece
    */
    if (((strpos($_POST["txt_email"], "@")==false) &&
(strpos($_POST["txt_email"], ".")==false)) || (strrpos($_POST["txt_email"], ".")<strpos($_POST["txt_email"], "@")) ) {
        $validacampos.="O campo e-mail deve conter um e-mail
válido<br />";
        $controle=true;
    }
}

if ($controle==true)
    erro(2,$validacampos);
```

```
else {
    ?>
<table align="center" cellpadding="2" cellspacing="0" border="1">
<tr>
    <td>Nome</td>
    <td><?=$_POST["txt_nome"]?></td>
</tr>
<tr>
    <td>Endereço</td>
    <td><?=$_POST["txt_endereco"]?></td>
</tr>
<tr>
    <td>Nome da mãe</td>
    <td><?=$_POST["txt_nomemae"]?></td>
</tr>
<tr>
    <td>Idade</td>
    <td><?=$_POST["txt_idade"]?></td>
</tr>
<tr>
    <td>E-mail</td>
    <td><?=$_POST["txt_email"]?></td>
</tr>
</table>

<?php
$arq="arquivo.txt";
$fp=fopen($arq,"w+");
if ($fp) {
    fwrite($fp,"Nome:".$_POST["txt_nome"]."\n");
```

```

fwrite($fp,"Endereço:".$_POST["txt_endereco"]."\n");
fwrite($fp,"Nome da mãe:".$_POST["txt_nomemae"]."\n");
fwrite($fp,"Idade:".$_POST["txt_idade"]."\n");
fwrite($fp,"E-mail:".$_POST["txt_email"]."\n");

fclose($fp);

echo "<p><a href=\"learquivo.php?arq=".$arq."\">Ler arquivo  

Texto Gerado</a></p>";
} else echo "<h4>Erro ao gerar o arquivo</h4>";
}

}

?>

</body>
</html>

```

#### <<< arquivo learquivo.php >>>

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Lê arquivo</title>

</head>

<body>

<?php

if (@$_GET["arq"]!="") {

    // verifica se o arquivo existe
    if (file_exists($_GET["arq"])) {

        $fp=fopen($_GET["arq"],"r");
        if ($fp) {

```

```

echo "<h1>Resultado da leitura</h1>";
echo '<table align="left" cellpadding="2" cellspacing="0"
border="1">';
//feof testa o fim do arquivo, utiliza um ponteiro lendo
linha a linha
while (!feof($fp)) {
    $conteudo=fgets($fp);
    //converte uma string em array ao encontrar o
caractere :
    $dados=explode(":", $conteudo);
    if ($dados[0]== "") continue;
    echo "<tr><td>".$dados[0]."</td>";
    echo "<td>".$dados[1]."</td></tr>";
}
} else echo "<h4>Erro ao gerar o arquivo</h4>";
}
}
?>
</body>
</html>

```

## Unidade 4

1.

**<<< estrutura de arquivos >>>**



**<<< criar tabela no banco de dados >>>**

```

DROP TABLE IF EXISTS `livro`;

CREATE TABLE IF NOT EXISTS `livro` (
  `codigo` int(11) NOT NULL AUTO_INCREMENT,
  `nome` varchar(100) NOT NULL,
  `autor` varchar(100) DEFAULT NULL,
  `editora` varchar(100) DEFAULT NULL,
  `paginas` int(11) DEFAULT NULL,
  PRIMARY KEY (`codigo`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

```

**<<< arquivo config.php >>>**

```

<?php
// conexão com o banco de dados
mysql_connect('localhost', 'web', 'senha123');
mysql_select_db('prg3');
?>

```

**<<< arquivo estilo.css >>>**

```

/* CSS criado para melhorar o visual da aplicação*/
/* Tudo */
* { margin:0; padding:0; }

/* Tags */
html { background:#F4F4F4; text-align:center; }

body { font-size:10pt; font-family:"Trebuchet MS"; width:400px;
background:#FFFFFF; margin:0 auto; text-align:left; padding:10px; }

a { text-decoration:none; color:#336699; }

a:hover { text-decoration:underline; }

/* Mensagens */
.mensagem { margin-bottom:10px; padding:10px; }

```

```
.erro { background:#FFFFE0; border:1px solid #E6DB55; color:#333333; }

.sucesso { border:1px solid #568F0F; color:#333; background:#DFF1C0; }

/* Formulario */

form { text-align:left; display:table; width:100%; clear:both; }

label, input { display:block; padding:5px; }

input#nome, input#autor, input#editora { width:390px; }

input#consultar, input#cadastrar { float:right; }

/* Menu */

#menu { border-bottom:1px dashed #CCC; padding-bottom:5px; }

#menu li { display:inline; width:100px; padding:10px; }

/* Tabelas */

table { border-collapse:collapse; width:100%; border:1px solid #B4D1DC;

border-bottom:3px solid #B4D1DC; }

table tr { background:#FFF; }

table th { background:#B4D1DC; border-bottom:3px solid #777; }

table td, table th { padding:5px; text-align:center; border:1px solid #ccc; }
```

**<<< arquivo cadastro.php >>>**

```
<?php

//inclui o conexão com o banco de dados

include('./cfg/config.php');

?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.

w3.org/1999/xhtml">

<head>
```

```

<meta http-equiv="Content-Type" content="text/
html; charset=ISO-8859-1"/>

<title>Cadastro de Livros</title>

<link href=".css/estilo.css" type="text/css" rel="stylesheet" />

</head>

<body>

<?php

// inclui o menu de navegação
include('menu.php');

?>

<h1>Cadastro de Livros</h1>

<?php

// verifica se o formulário foi enviado
if(!empty($_POST)):

    // valida os campos

    if( (!empty($_POST['nome'])) && (!empty($_POST['autor']))
&& (!empty($_POST['editora'])) && (!empty($_POST['paginas'])) ):

        $nome = $_POST['nome'];
        $autor = $_POST['autor'];
        $editora = $_POST['editora'];
        $paginas = $_POST['paginas'];

        $livros = mysql_query('SELECT * FROM livro
WHERE lower(nome) = "'.$strtolower($nome).'"');

        if(mysql_num_rows($livros) >= 1):

            ?>

            <div class="mensagem erro">
                Jé; existe uma livro com
este nome cadastrado.
            </div>

            <?php

            elseif(mysql_query('INSERT INTO livro VALUES
(NULL, "'.$nome.'", "'.$autor.'", "'.$editora.'", "'.$paginas.'")')):
```

```
?>

<div class="mensagem sucesso">
    Livro cadastrado com sucesso.
</div>

<?php

else:
?>

<div class="mensagem erro">
    Erro ao cadastrar livro.
</div>

<?php

endif;

else:
?>

<div class="mensagem erro">
    &Eacute; necess&aacute;rio preencher
todos os campos para cadastrar o livro.
</div>

<?php

endif;

endif;
?>

<form name="cadastro" id="cadastro" method="post">
    <label for="nome">Nome:</label>
    <input type="text" name="nome" id="nome"
tabindex="1" />
    <label for="autor">Autor:</label>
    <input type="text" name="autor" id="autor"
tabindex="2" />
    <label for="editora">Editora:</label>
    <input type="text" name="editora" id="editora"
tabindex="3" />
```

```

        <label for="paginas">Páginas:</label>
        <input type="text" name="paginas" id="paginas"
tabindex="4" />
        <input type="submit" name="cadastrar" id="cadastrar"
value="Cadastrar" tabindex="5" />
    </form>
</body>
</html>

```

**<<< arquivo consulta.php >>>**

```

<?php
//inclui a conexão
include('cfg/config.php');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.
w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/
html; charset=ISO-8859-1"/>
        <title>Consulta de Livros</title>
        <link href=".css/estilo.css" type="text/css" rel="stylesheet" />
    </head>
    <body>
        <?php
        //inclui o menu de navegação
        include('menu.php');
        ?>
        <h1>Consulta de Livros</h1>
        <?php
        if(!empty($_POST)):
```

```
$nome = strtolower($_POST['nome']);

$queryConsulta = mysql_query('SELECT * FROM livro
WHERE lower(nome) LIKE "%'.$nome.'%"');

$registros = mysql_num_rows($queryConsulta);

if( ($registros >= 1) && $nome != ""):

?>

<div class="mensagem sucesso">
    <strong><?php echo $registros; ?></strong> livro(s) encontrado(s).

</div>

<table>
    <thead>
        <tr>
            <th>C&oacute;digo</th>
            <th>Nome</th>
            <th>Autor</th>
            <th>Editora</th>
            <th>P&aacute;ginas</th>
        </tr>
    </thead>
    <tbody>
        <?php
            while($livro = mysql_fetch_
assoc($queryConsulta)):

?>

        <tr>
            <td><a href="./
livro.php?id=<?php echo $livro['codigo']; ?>"><?php echo $livro['codigo'];
?></a></td>
            <td><a href="./
livro.php?id=<?php echo $livro['codigo']; ?>"><?php echo $livro['nome'];
?></a></td>
```

```
<td><?php echo  
$livro['autor']; ?></td>  
  
<td><?php echo  
$livro['editora']; ?></td>  
  
<td><?php echo  
$livro['paginas']; ?></td>  
  
</tr>  
  
<?php  
endwhile;  
  
?>  
  
</tbody>  
  
</table>  
  
<?php  
  
else:  
  
?>  
  
<div class="mensagem erro">  
  
    Nenhum livro encontrado.  
  
</div>  
  
<?php  
  
endif;  
  
endif;  
  
?>  
  
<form name="consulta" id="consulta" method="post">  
  
    <label for="nome">Nome:</label>  
  
    <input tabindex="1" type="text" id="nome"  
name="nome" />  
  
    <input tabindex="2" id="consultar" name="consultar"  
type="submit" value="Consultar" />  
  
</form>  
  
</body>  
  
</html>
```

**<<< arquivo index.php >>>**

```
<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.w3.org/1999/xhtml">

    <head>
        <title>Livros</title>
        <link href=".css/estilo.css" type="text/css" rel="stylesheet" />
    </head>
    <body>
        <?php
            // inclui o menu de navegação
            include('menu.php');
        ?>
    </body>
</html>
```

**<<< arquivo lista.php >>>**

```
<?php
include('./cfg/config.php');

?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.w3.org/1999/xhtml">

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
        <title>Lista de Livros</title>
        <link href=".css/estilo.css" type="text/css" rel="stylesheet" />
    </head>
    <body>
```

```
<?php
include('menu.php');

?>
<h1>Lista de Livros</h1>

<table>

    <thead>

        <tr>
            <th>C&oacute;digo</th>
            <th>Nome</th>
            <th>Autor</th>
            <th>Editora</th>
            <th>P&aacute;ginas</th>
        </tr>
    </thead>
    <tbody>

        <?php
        $livros = mysql_query('SELECT * FROM livro');
        while($livro = mysql_fetch_assoc($livros)):

            ?>
            <tr>
                <td><a href=".//livro.
php?id=<?php echo $livro['codigo']; ?>"><?php echo $livro['codigo']; ?></a></td>
                <td><a href=".//livro.
php?id=<?php echo $livro['codigo']; ?>"><?php echo $livro['nome']; ?></a></td>
                <td><?php echo $livro['autor'];
?></td>
                <td><?php echo $livro['editora'];
?></td>
                <td><?php echo $livro['paginas'];
?></td>
            </tr>
        <?php
        }
    </tbody>
</table>
```

```
<?php  
endwhile;  
?>  
</tbody>  
</table>  
</body>  
</html>
```

**<<< arquivo livro.php >>>**

```
<?php  
include('./cfg/config.php');  
?>  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
      "http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.  
w3.org/1999/xhtml">  
  <head>  
    <meta http-equiv="Content-Type" content="text/  
html; charset=ISO-8859-1"/>  
    <title>Detalhes do Livro</title>  
    <link href=".css/estilo.css" type="text/css" rel="stylesheet" />  
  </head>  
  <body>  
    <?php  
    include('menu.php');  
    $id = $_GET['id'];  
    $queryLivro = mysql_query('SELECT * FROM livro WHERE codigo  
= '.$id);  
    if(mysql_num_rows($queryLivro) == 1):  
      $livro = mysql_fetch_assoc($queryLivro);  
    ?>  
    <h1>Detalhes do Livro</h1>
```

```
<table>
    <tbody>
        <tr>
            <th>Código</th>
            <td><?php echo $livro['codigo'];
?></td>
        </tr>
        <tr>
            <th>Nome</th>
            <td><?php echo $livro['nome'];
?></td>
        </tr>
        <tr>
            <th>Autor</th>
            <td><?php echo $livro['autor'];
?></td>
        </tr>
        <tr>
            <th>Editora</th>
            <td><?php echo $livro['editora'];
?></td>
        </tr>
        <tr>
            <th>Páginas</th>
            <td><?php echo $livro['paginas'];
?></td>
        </tr>
    </tbody>
</table>
<?php
else:
?>
```

```
<div class="mensagem erro">  
    Livro n&atilde;o encontrado.  
</div>  
<?php  
endif;  
?>  
</body>  
</html>
```

**<<< arquivo menu.php >>>**

```
<ul id="menu">  
    <li><a tabindex="51" href="cadastro.php" title="Cadastro">Cadastro</a></li>  
    <li><a tabindex="52" href="consulta.php" title="Consulta">Consulta</a></li>  
    <li><a tabindex="53" href="lista.php" title="Lista">Lista</a></li>  
</ul>
```

2.

**<<<estrutura de arquivos>>>**



**<<< criar tabela no banco de dados >>>**

```
CREATE TABLE IF NOT EXISTS `usuario` (  
    `codigo` int(11) NOT NULL AUTO_INCREMENT,  
    `nome` varchar(100) DEFAULT NULL,  
    `login` varchar(20) DEFAULT NULL,  
    `senha` char(32) DEFAULT NULL,
```

```
PRIMARY KEY (`codigo`)  
 ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

**<<< arquivo config.php >>>**

```
<?php  
//realiza a conexão com o banco de dados  
mysql_connect('localhost', 'web', 'senha123');  
mysql_select_db('prg3');  
?>
```

**<<< arquivo estilo.css >>>**

```
/* CSS criado para melhorar o visual da aplicação*/  
/* Tudo */  
* { margin:0; padding:0; }  
  
/* Tags */  
html { background:#F4F4F4; text-align:center; }  
body { font-size:10pt; font-family:"Trebuchet MS"; width:400px;  
background:#FFFFFF; margin:0 auto; text-align:left; padding:10px; }  
a { text-decoration:none; color:#336699; }  
a:hover { text-decoration:underline; }  
  
/* Mensagens */  
.mensagem { margin-bottom:10px; padding:10px; }  
.erro { background:#FFFFE0; border:1px solid #E6DB55; color:#333333; }  
.sucesso { border:1px solid #568F0F; color:#333; background:#DFF1C0;}  
  
/* Formulario */  
form { text-align:left; display:table; width:100%; clear:both; }  
label, input { display:block; padding:5px;}
```

```
input#nome, input#login, input#senha { width:390px; }

input#consultar, input#cadastrar { float:right; }

/* Menu */

#menu { border-bottom:1px dashed #CCC; padding-bottom:5px; }

#menu li { display:inline; width:100px; padding:10px; }

/* Tabelas */

table { border-collapse:collapse; width:100%; border:1px solid #B4D1DC;
border-bottom:3px solid #B4D1DC; }

table tr { background:#FFF; }

table th { background:#B4D1DC; border-bottom:3px solid #777; }

table td, table th { padding:5px; text-align:center; border:1px solid #ccc; }
```

<<< arquivo **cadastro.php** >>>

```
<?php

//verifica se o usuário está logado

include('verificacao.php');

//inclui a conexão com o banco de dados

include('cfg/config.php');

?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.
w3.org/1999/xhtml">

<head>

    <meta http-equiv="Content-Type" content="text/
html;charset=ISO-8859-1"/>

    <title>Cadastro de Usu&aacute;rios</title>

    <link href=".css/estilo.css" type="text/css" rel="stylesheet" />

</head>

<body>
```

```
<?php
include('menu.php');

?>

<h1>Cadastro de Usu&acute;rios</h1>

<?php
if(!empty($_POST)):

    if( (!empty($_POST['nome'])) && (!empty($_POST['login']))
&& (!empty($_POST['senha'])) ):

        $nome = $_POST['nome'];
        $login = $_POST['login'];
        $senha = $_POST['senha'];

        $usuarios = mysql_query("SELECT * FROM
usuario WHERE (lower(nome) = '".strtolower($nome)."') OR (lower(login) =
".strtolower($login)."')");

        if(mysql_num_rows($usuarios) >= 1):

            ?>
            <div class="mensagem erro">
                J&acute;e existe uma
                usu&acute;rio com este nome ou login cadastrado.
            </div>

            <?php

            elseif(mysql_query("INSERT INTO usuario VALUES
(NULL, '$nome', '$login', MD5('$senha'))"):

            ?>
            <div class="mensagem sucesso">
                Usu&acute;rio cadastrado com
                sucesso.
            </div>

            <?php

            else:
            ?>
            <div class="mensagem erro">
```

```
        Erro ao cadastrar usu&acute;rio.  
    </div>  
    <?php  
endif;  
else:  
?>  
    <div class="mensagem erro">  
        &Eacute; necess&acute;rio preencher  
        todos os campos para cadastrar o usu&acute;rio.  
    </div>  
    <?php  
endif;  
endif;  
?>  
    <form name="cadastro" id="cadastro" method="post">  
        <label for="nome">Nome:</label>  
        <input type="text" name="nome" id="nome"  
        tabindex="1" />  
        <label for="login">Login:</label>  
        <input type="text" name="login" id="login"  
        tabindex="2" />  
        <label for="senha">Senha:</label>  
        <input type="password" name="senha" id="senha"  
        tabindex="3" />  
        <input type="submit" name="cadastrar" id="cadastrar"  
        value="Cadastrar" tabindex="4" />  
    </form>  
    </body>  
</html>
```

## &lt;&lt;&lt; arquivo consulta.php &gt;&gt;&gt;

```
<?php
//verifica se o usuário está logado
include('./verificacao.php');
//inclui a conexão com o banco de dados
include('./cfg/config.php');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.
w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/
html;charset=ISO-8859-1"/>
<title>Consulta de Usu&aacute;rios</title>
<link href=".css/estilo.css" type="text/css" rel="stylesheet" />
</head>
<body>
<?php
include('menu.php');
?>
<h1>Consulta de Usu&aacute;rios</h1>
<?php
if(!empty($_POST)):
    $nome = strtolower($_POST['nome']);
    $queryConsulta = mysql_query('SELECT * FROM usuario
WHERE lower(nome) LIKE "%'.$nome.'%"');
    $registros = mysql_num_rows($queryConsulta);
    if( ($registros >= 1) && $nome != ""):
?>
<div class="mensagem sucesso">
```

```
<strong><?php echo $registros; ?></strong> usu&aacute;rio(s) encontrado(s).

</div>

<table>

<thead>

<tr>

<th>C&oacute;digo</th>

<th>Nome</th>

<th>Login</th>

</tr>

</thead>

<tbody>

<?php

while($usuario = mysql_fetch_assoc($queryConsulta)):

?>

<tr>

<td><a href="./" usuario.php?id=<?php echo $usuario['codigo']; ?>"><?php echo $usuario['codigo']; ?></a></td>

<td><a href="./" usuario.php?id=<?php echo $usuario['codigo']; ?>"><?php echo $usuario['nome']; ?></a></td>

<td><?php echo $usuario['login']; ?></td>

</tr>

<?php

endwhile;

?>

</tbody>

</table>

<?php

else:
```

```

?>

<div class="mensagem erro">
    Nenhum usuário encontrado.
</div>

<?php

endif;

endif;

?>

<form name="consulta" id="consulta" method="post">
    <label for="nome">Nome:</label>
    <input tabindex="1" type="text" id="nome"
name="nome" />
    <input tabindex="2" id="consultar" name="consultar"
type="submit" value="Consultar" />
</form>
</body>
</html>

```

**<<< arquivo index.php >>>**

```

<?php
// Verifica se está logado
include('verificacao.php');

?>

<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.
w3.org/1999/xhtml">

    <head>
        <title>Usuários</title>
        <link href=".css/estilo.css" type="text/css" rel="stylesheet" />
    </head>

    <body>

```

```
<?php  
    // Se estiver logado inclui o menu  
    include('menu.php');  
?  
</body>  
</html>
```

**<<< arquivo lista.php >>>**

```
<?php  
include('./verificacao.php');  
include('./cfg/config.php');  
?  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    "http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.  
w3.org/1999/xhtml">  
    <head>  
        <meta http-equiv="Content-Type" content="text/  
html; charset=ISO-8859-1"/>  
        <title>Lista de Usu&aacute;rios</title>  
        <link href=".//css/estilo.css" type="text/css" rel="stylesheet" />  
    </head>  
    <body>  
        <?php  
        include('menu.php');  
        ?>  
        <h1>Lista de Usu&aacute;rios</h1>  
        <table>  
            <thead>  
                <tr>
```

```
<th>C&oacute;digo</th>
<th>Nome</th>
<th>Login</th>
</tr>
</thead>
<tbody>
<?php
$usuarios = mysql_query("SELECT * FROM
usuario");
while($usuario = mysql_fetch_assoc($usuarios)):
?>
<tr>
<td><a href=". /usuario.
php?id=<?php echo $usuario['codigo']; ?>"><?php echo
$usuario['codigo']; ?></a></td>
<td><a href=". /usuario.
php?id=<?php echo $usuario['codigo']; ?>"><?php echo $usuario['nome'];
?></a></td>
<td><?php echo $usuario['login'];
?></td>
</tr>
<?php
endwhile;
?>
</tbody>
</table>
</body>
</html>
```

## &lt;&lt;&lt; arquivo login.php &gt;&gt;&gt;

```
<?php
//inicia a sessão
session_start();
include('./cfg/config.php');

?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.
w3.org/1999/xhtml">

<head>
    <meta http-equiv="Content-Type" content="text/
html;charset=ISO-8859-1"/>
    <title>&Aacute;rea Restrita</title>
    <link href=".css/estilo.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <h1>&Aacute;rea Restrita</h1>
    <?php
        if(!empty($_POST)):
            if( (!empty($_POST['login'])) && (!empty($_POST['senha'])))
            :
                $login = $_POST['login'];
                $senha = $_POST['senha'];
                $queryUsuario = mysql_query('SELECT * FROM
usuario WHERE login = "'.$login.'" AND senha = MD5("'.$senha.'")');
                if(mysql_num_rows($queryUsuario) == 1):
                    $usuario = mysql_fetch_
assoc($queryUsuario);
                    // registra uma sessão
                    $_SESSION['administracao']['usuario'] =
$usuario['nome'];
                endif;
            endif;
        endif;
    </body>
</html>
```

```
?>

<div class="mensagem sucesso">
    Autenticado com sucesso. <br />
    <a href=".index.php">Clique
aqui</a> para continuar.

</div>
<?php

else:
    ?>
    <div class="mensagem erro">
        Login/Senha incorretos.
    </div>
    <?php

endif;

else:
    ?>
    <div class="mensagem erro">
        &Eacute; necess&aacute;rio preencher
todos os campos.

    </div>
    <?php

endif;

endif;
?>

<form name="cadastro" id="cadastro" method="post">
    <label for="login">Login:</label>
    <input type="text" name="login" id="login" tabindex="1"
/>
    <label for="senha">Senha:</label>
    <input type="password" name="senha" id="senha"
tabindex="2" />
```

```
<input type="submit" name="entrar" id="entrar"  
value="Entrar" tabindex="3" />  
  
</form>  
  
</body>  
  
</html>
```

**<<< arquivo logout.php >>>**

```
<?php  
  
//inicia a sessão  
  
session_start();  
  
//destroi a variável  
  
unset($_SESSION['administracao']);  
  
//redireciona para o login  
  
header('Location: ./login.php');  
  
?>
```

**<<< arquivo menu.php >>>**

```
<div class="mensagem sucesso">  
  
    Bem vindo <strong><?php echo $_SESSION['administracao']['usuario'];  
?></strong>! [<a href=".logout.php">Sair</a>]  
  
</div>  
  
<ul id="menu">  
  
    <li><a tabindex="51" href="cadastro.php" title="Cadastro">Cadastro</a></li>  
  
    <li><a tabindex="52" href="consulta.php" title="Consulta">Consulta</a></li>  
  
    <li><a tabindex="53" href="lista.php" title="Lista">Lista</a></li>  
  
</ul>
```

## &lt;&lt;&lt; arquivo usuário.php &gt;&gt;&gt;

```
<?php
include('verificacao.php');
include('cfg/config.php');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.w3.org/1999/xhtml">

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
        <title>Detalhes do Usu&aacute;rio</title>
        <link href=".css/estilo.css" type="text/css" rel="stylesheet" />
    </head>

    <body>
        <?php
            include('menu.php');
            $id = $_GET['id'];
            $queryUsuario = mysql_query('SELECT * FROM usuario WHERE
codigo = '.$id);
            if(mysql_num_rows($queryUsuario) == 1):
                $usuario = mysql_fetch_assoc($queryUsuario);
            ?>
            <h1>Detalhes do Usu&aacute;rio</h1>
            <table>
                <tbody>
                    <tr>
                        <th>C&oacute;digo</th>
                        <td><?php echo
$usuario['codigo']; ?></td>
```

```
</tr>
<tr>
    <th>Nome</th>
    <td><?php echo
$usuario['nome']; ?></td>
</tr>
<tr>
    <th>Login</th>
    <td><?php echo $usuario['login'];
?></td>
</tr>
</tbody>
</table>
<?php
else:
?>
<div class="mensagem erro">
    Usu&aacute;rio n&atilde;o encontrado.
</div>
<?php
endif;
?>
</body>
</html>
```

**<<< arquivo verificacao.php >>>**

```
<?php  
//inicia a sessão  
session_start();  
//verifica se uma variável foi registrada  
if(!isset($_SESSION['administracao'])):  
    // caso não esteja redireciona para a página de login  
    header('Location: ./login.php');  
    // caso ocorra erro interrompe a construção da página  
    die();  
endif;  
?>
```

## Unidade 5

1. Questão aberta.

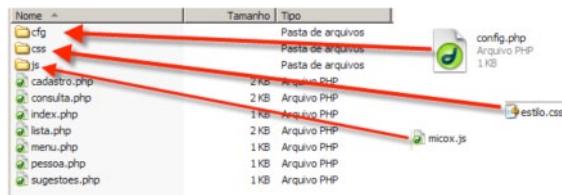
## Unidade 6

1.

### **Script para criar para criar o banco de dados**

```
CREATE TABLE IF NOT EXISTS pessoa (  
    codigo int(11) NOT NULL AUTO_INCREMENT,  
    nome varchar(100) DEFAULT NULL,  
    cadastro datetime DEFAULT NULL,  
    PRIMARY KEY (codigo)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1 ;
```

**A solução foi desenvolvida e organizada em três diretórios com um arquivo cada. Já o diretório padrão possui sete arquivos dispostos como na imagem?**



**A folha de estilo CSS (opcional apenas melhoramento visual), organizada dentro do diretório css com o nome de *estilo.css*.**

```
/* Todo */
* { margin:0; padding:0; }

/* Tags */
html { background:#F4F4F4; text-align:center; }
body { font-size:10pt; font-family:"Trebuchet MS"; width:400px;
background:#FFFFFF; margin:0 auto; text-align:left; padding:10px; }
a { text-decoration:none; color:#336699; }
a:hover { text-decoration:underline; }

/* Mensagens */
.mensagem { margin-bottom:10px; padding:10px; }
.erro { background:#FFFE0; border:1px solid #E6DB55; color:#333333; }
.sucesso { border:1px solid #568F0F; color:#333; background:#DFF1C0; }

/* Formulario */
form { text-align:left; display:table; width:100%; clear:both; }
label, input { display:block; padding:5px; }
input#nome { width:390px; }
input#consultar, input#cadastrar { float:right; }

/* Menu */
```

```
#menu { border-bottom:1px dashed #CCC; padding-bottom:5px; }

#menu li { display:inline; width:100px; padding:10px; }

/* Tabelas */

table { border-collapse:collapse; width:100%; border:1px solid #B4D1DC;
border-bottom:3px solid #B4D1DC; }

table tr { background:#FFF; }

table th { background:#B4D1DC; border-bottom:3px solid #777; }

table td, table th { padding:5px; text-align:center; border:1px solid #ccc; }

/* Sugestões */

#sugestoes { max-height:300px; overflow:scroll; text-align:left; margin-
top:-35px; z-index:1000; width:250px; background:#E7F3FF; border:1px
solid #DDD; display:none; }

#sugestoes ul li { list-style-type:none; line-height:25px; height:25px; }

#sugestoes ul li a { padding-left:10px; line-height:25px; height:25px;
width:250px; }

#sugestoes ul li:hover, #sugestoes ul li a:hover { background:#DDD;
color:#777; }
```

**Script em JavaScript com o nome de *micox.js* dentro do diretório *js*.**

```
function ajaxGet(url,elemento_retorno,exibe_carregando){

/*
 * ajaxGet - Coloca o retorno de uma url em um elemento qualquer
 * Use à vontade, mas coloque meu nome nos créditos. Dúvidas, me
 * mande um email.
 *
 * A função é grande, coloque-a em um arquivo .js separado.
 *
 * Versão: 1.2 - 20/04/2006
 *
 * Autor: Micox - Náiron J.C.G - micoxjcg@yahoo.com.br - elmicox.
 * blogspot.com
 *
 * Parâmetros:
 *
 * url: string; elemento_retorno: object|string; exibe_carregando:boolean
 */}
```

```
* - Se elemento_retorno for um elemento html (inclusive inputs e
selects),
*   exibe o retorno no innerHTML / value / options do elemento
* - Se elemento_retorno for o nome de uma variavel
*   (o nome da variável deve ser declarado por string, pois será feito um
eval)
*   a função irá atribuir o retorno à variável ao receber a url.

*****/
var ajax1 = pegaAjax();
if(ajax1){
    url = antiCacheRand(url)
    ajax1.onreadystatechange = ajaxOnReady
    ajax1.open("GET", url ,true);
    //ajax1.setRequestHeader("Content-Type", "text/html; charset=iso-
8859-1");//"application/x-www-form-urlencoded");
    ajax1.setRequestHeader("Cache-Control", "no-cache");
    ajax1.setRequestHeader("Pragma", "no-cache");
    if(exibe_carregando){ put("Carregando ...") }
    ajax1.send(null)
    return true;
}else{
    return false;
}
function ajaxOnReady(){
    if (ajax1.readyState==4){
        if(ajax1.status == 200){
            var texto=ajax1.responseText;
            if(texto.indexOf(" ")<0) texto=texto.replace(/\+/g," ");
            //texto=unescape(texto); //descomente esta linha se tiver usado o
urlencode no php ou asp
            put(texto);
            extraiScript(texto);
        }
    }
}
```

```
 }else{  
    if(exibe_carregando){put("Falha no carregamento. " +  
    httpStatus.ajax1.status));}  
}  
  
ajax1 = null  
  
}else if(exibe_carregando){//para mudar o status de cada carregando  
    put("Carregando ...")  
}  
}  
  
}  
  
function put(valor){ //coloca o valor na variavel/elemento de retorno  
  
    if(typeof(elemento_retorno).toLowerCase() == "string"){ //se for o  
    nome da string  
  
        if(valor != "Falha no carregamento"){  
            eval(elemento_retorno + '= unescape("' + escape(valor) + "')')  
        }  
  
    }else if(elemento_retorno.tagName.toLowerCase() == "input"){  
        valor = escape(valor).replace(/\%0D\%0A/g,"")  
        elemento_retorno.value = unescape(valor);  
    }else if(elemento_retorno.tagName.toLowerCase() == "select"){  
        select_innerHTML(elemento_retorno,valor)  
    }else if(elemento_retorno.tagName){  
        elemento_retorno.innerHTML = valor;  
        //alert(elemento_retorno.innerHTML)  
    }  
}  
  
}  
  
function pegaAjax(){ //instancia um novo xmlhttprequest  
  
    //baseado na getXMLHttpObj que possui muitas cópias na net e eu  
    nao sei quem é o autor original  
  
    if(typeof(XMLHttpRequest) != 'undefined'){return new  
    XMLHttpRequest();}  
  
    var axO = ['Microsoft.XMLHTTP','Msxml2.XMLHTTP','Msxml2.  
    XMLHTTP.6.0','Msxml2.XMLHTTP.4.0','Msxml2.XMLHTTP.3.0'];
```

```
for(var i=0;i<axO.length;i++){ try{ return new ActiveXObject(axO[i]);}
catch(e){} }

return null;

}

function httpStatus(stat){ //retorna o texto do erro http

switch(stat){

    case 0: return "Erro desconhecido de javascript";

    case 400: return "400: Solicitação incompreensível"; break;

    case 403: case 404: return "404: Não foi encontrada a URL
solicitada"; break;

    case 405: return "405: O servidor não suporta o método solicitado";
break;

    case 500: return "500: Erro desconhecido de natureza do servidor";
break;

    case 503: return "503: Capacidade máxima do servidor alcançada";
break;

    default: return "Erro " + stat + ". Mais informações em http://www.
w3.org/Protocols/rfc2616/rfc2616-sec10.html"; break;

}

}

function antiCacheRand(aurl){

    var dt = new Date();

    if(aurl.indexOf("?")>=0){// já tem parametros

        return aurl + "&" + encodeURI(Math.random() + "_" + dt.getTime());

    }else{ return aurl + "?" + encodeURI(Math.random() + "_" +
dt.getTime());}

}

}

function select_innerHTML(objeto,innerHTML){

*****  

* select_innerHTML - altera o innerHTML de um select independente se é
FF ou IE  

* Corrige o problema de não ser possível usar o innerHTML no IE
corretamente
```

```
* Veja o problema em: http://support.microsoft.com/default.aspx?scid=kb;en-us;276228

* Use a vontade mas coloque meu nome nos créditos. Dúvidas, me
mande um email.

* Versão: 1.0 - 06/04/2006

* Autor: Micox - Náiron J.C.G - micoxjcg@yahoo.com.br - elmicox.
blogspot.com

* Parâmetros:

* objeto(tipo object): o select a ser alterado

* innerHTML(tipo string): o novo valor do innerHTML

*****/

objeto.innerHTML = ""

var selTemp = document.createElement("micoxselect")

var opt;

selTemp.id="micoxselect1"

document.body.appendChild(selTemp)

selTemp = document.getElementById("micoxselect1")

selTemp.style.display="none"

if(innerHTML.toLowerCase().indexOf("<option")<0){//se não é option eu
    converto

    innerHTML = "<option>" + innerHTML + "</option>"

}

innerHTML = innerHTML.replace(/<option/g,"<span").replace(/<\/
option/g,"</span")

selTemp.innerHTML = innerHTML

for(var i=0;i<selTemp.childNodes.length;i++){

    if(selTemp.childNodes[i].tagName){

        opt = document.createElement("OPTION")

        for(var j=0;j<selTemp.childNodes[i].attributes.length;j++){

            opt.setAttributeNode(selTemp.childNodes[i].attributes[j].
cloneNode(true))

        }

    }

}
```

```
opt.value = selTemp.childNodes[i].getAttribute("value")
opt.text = selTemp.childNodes[i].innerHTML
if(document.all){ //IEca
    objeto.add(opt)
} else{
    objeto.appendChild(opt)
}
}

}

document.body.removeChild(selTemp)
selTemp = null
}

function extraiScript(texto){
//Maravilhosa função feita pelo Skywalker.TO do imasters/forum
//http://forum.imasters.com.br/index.php?showtopic=165277&
// inicializa o inicio ><
var ini = 0;
// loop enquanto achar um script
while (ini!=-1){
    // procura uma tag de script
    ini = texto.indexOf('<script', ini);
    // se encontrar
    if (ini >=0){
        // define o inicio para depois do fechamento dessa tag
        ini = texto.indexOf('>', ini) + 1;
        // procura o final do script
        var fim = texto.indexOf('</script>', ini);
        // extrai apenas o script
        codigo = texto.substring(ini,fim);
    }
}
```

```

// executa o script
//eval(codigo);
/*************
 * Alterado por Micox - micoxjcg@yahoo.com.br
 * Alterei pois com o eval não executava funções.
 *****/
novo = document.createElement("script")
novo.text = codigo;
document.body.appendChild(novo);
}
}

```

Fonte: <www.elmicox.blogspot.com>.

## Scripts PHP

**Arquivo de configuração *config.php* no diretório *cfg*.**

```

<?php
mysql_connect('localhost', 'root', '');
mysql_select_db('exe_lessa');
?>

```

**Arquivo *index.php* no diretório padrão.**

```

<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.
w3.org/1999/xhtml">

    <head>
        <title>Pessoas</title>
        <link href=".css/estilo.css" type="text/css" rel="stylesheet" />
    </head>
    <body>
        <?php
        include('menu.php');

```

```
?>  
</body>  
</html>
```

**Arquivo *menu.php* no diretório padrão.**

```
<ul id="menu">  
    <li><a tabindex="3" href="cadastro.php" title="Cadastro">Cadastro</a></li>  
    <li><a tabindex="4" href="consulta.php" title="Consulta">Consulta</a></li>  
    <li><a tabindex="5" href="lista.php" title="Lista">Lista</a></li>  
</ul>
```

**Arquivo *cadastro.php* no diretório padrão.**

```
<?php  
include('cfg/config.php');  
?>  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    "http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.  
w3.org/1999/xhtml">  
    <head>  
        <meta http-equiv="Content-Type" content="text/  
html; charset=ISO-8859-1"/>  
        <title>Cadastro de Pessoas</title>  
        <link href=".css/estilo.css" type="text/css" rel="stylesheet" />  
    </head>  
    <body>  
        <?php  
        include('menu.php');  
        ?>  
        <h1>Cadastro de Pessoas</h1>
```

```

<?php

if(!empty($_POST)):

    if(!empty($_POST['nome'])):

        $nome = $_POST['nome'];

        $pessoas = mysql_query("SELECT * FROM pessoa
WHERE UPPER(nome) = '".strtoupper($nome)."'");

        if(mysql_num_rows($pessoas) >= 1):

            ?>

            <div class="mensagem erro">

                J&acute; existe uma pessoa com
este nome cadastrada.

            </div>

            <?php

                elseif(mysql_query("INSERT INTO pessoa VALUES
(NULL, '".$nome."', SYSDATE())")):

                    ?>

                    <div class="mensagem sucesso">

                        Pessoa Cadastrada com sucesso.

                    </div>

                    <?php

                else:

                    ?>

                    <div class="mensagem erro">

                        Erro ao cadastrar pessoa.

                    </div>

                    <?php

                endif;

            else:

                ?>

                <div class="mensagem erro">

                    Voc&ecirc; precisa digitar um nome para
cadastrar a pessoa.

                </div>

```

```
</div>

<?php

endif;

endif;

?>

<form name="cadastro" id="cadastro" method="post">

    <label for="nome">Nome:</label>

    <input type="text" name="nome" id="nome"
tabindex="1" />

    <input type="submit" name="cadastrar" id="cadastrar"
value="Cadastrar" tabindex="2" />

</form>

</body>

</html>
```

**Arquivo *consulta.php* no diretório padrão.**

```
<?php

session_start();

echo $_SESSION['teste'];

unset($_SESSION['teste']);

include('cfg/config.php');

?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.
w3.org/1999/xhtml">

    <head>

        <meta http-equiv="Content-Type" content="text/
html;charset=UTF-8"/>

        <title>Consulta de Pessoas</title>

        <link href=".css/estilo.css" type="text/css" rel="stylesheet" />
```

```
<script src=".js/micox.js" type="text/javascript"></script>
<script type="text/javascript">

    function exibeSugestoes() {
        document.getElementById('pessoa').style.
display='none';

        nome = document.getElementById('nome').value;
        ajaxGet('sugestoes.php?nome='+nome,
document.getElementById('sugestoes'), true);

        document.getElementById('sugestoes').style.
display='block';
    }

    function preenche(nome) {
        document.getElementById('sugestoes').style.
display='none';

        document.getElementById('pessoa').style.
display='none';

        document.getElementById('nome').value = nome;
    }

    function exibePessoa(nome) {
        document.getElementById('sugestoes').style.
display='none';

        nome = document.getElementById('nome').value;
        ajaxGet('pessoa.php?nome='+nome, document.
getElementById('pessoa'), true);

        document.getElementById('pessoa').style.
display='block';
        return false;
    }
</script>
</head>
<body>
<?php
    include('menu.php');
```

```
?>

<h1>Consulta de Pessoas</h1>

<form name="consulta" id="consulta"
onSubmit="javascript:return exibePessoa();">

    <label for="nome">Nome:</label>

    <input tabindex="1" type="text" onkeyup="javascript:ex
ibeSugestoes();" name="nome<?php echo rand(1,10000); ?>" id="nome"
/>

    <input tabindex="2" id="consultar" name="consultar"
type="submit" value="Consultar" />

</form>

<div id="sugestoes">

</div>

<div id="pessoa">

</div>

</body>

</html>
```

**Arquivo *lista.php* no diretório padrão.**

```
<?php

include('./cfg/config.php');

?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://w3c.globo.com/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html dir="ltr" xml:lang="pt-br" lang="pt-br" xmlns="http://www.
w3.org/1999/xhtml">

    <head>

        <meta http-equiv="Content-Type" content="text/
html;charset=ISO-8859-1"/>

        <title>Lista de Pessoas</title>

        <link href=".css/estilo.css" type="text/css" rel="stylesheet" />
```

```
</head>

<body>

<?php
include('menu.php');

?>

<h1>Lista de Pessoas</h1>

<table>

<thead>

<tr>
<th>C&oacute;digo</th>
<th>Nome</th>
<th>Data de Cadastro</th>
</tr>

</thead>

<tbody>

<?php
$pessoas = mysql_query('SELECT * FROM
pessoa');

while($pessoa = mysql_fetch_assoc($pessoas)):

?>

<tr>
<td><?php echo
$pessoa['codigo']; ?></td>
<td><?php echo $pessoa['nome'];
?></td>
<td><?php echo
$pessoa['cadastro']; ?></td>
</tr>

<?php
endwhile;

?>

</tbody>
```

```
</table>  
</body>  
</html>
```

**Arquivo *pessoa.php* no diretório padrão.**

```
<?php  
header("Content-Type: text/html; charset=ISO-8859-1",true);  
include('./cfg/config.php');  
$nome = $_GET['nome'];  
$pessoas = mysql_query('SELECT * FROM pessoa WHERE nome LIKE  
"%'.$nome.'%"');  
if(mysql_num_rows($pessoas) == 0):  
?>  
<div class="mensagem erro">  
    Nenhuma pessoa encontrada.  
</div>  
<?php  
elseif(mysql_num_rows($pessoas) == 1):  
    $pessoa = mysql_fetch_assoc($pessoas);  
?>  
<h1><?php echo $pessoa['nome']; ?></h1>  
<table>  
    <thead>  
        <tr>  
            <th>C&oacute;digo</th>  
            <th>Nome</th>  
            <th>Data de Cadastro</th>  
        </tr>  
    </thead>  
    <tbody>  
        <tr>
```

```

        <td><?php echo $pessoa['codigo']; ?></td>
        <td><?php echo $pessoa['nome']; ?></td>
        <td><?php echo $pessoa['cadastro']; ?></td>

    </tr>
</tbody>
</table>
<?php
else:
?>
<div class="mensagem erro">
    Foram encontradas v&aacute;rias pessoas. Por favor, refine sua
busca.
</div>
<?php
endif;
?>
```

### **Arquivo *sugestoes.php* no diretório padrão.**

```

<?php
header("Content-Type: text/html; charset=ISO-8859-1",true);
include('./cfg/config.php');
$nome = $_GET['nome'];
$nomes = mysql_query('SELECT * FROM pessoa WHERE nome LIKE
"%'.$nome.'%"');
if(mysql_num_rows($nomes) >= 1):
?>
<ul>
<?php
while($nome = mysql_fetch_assoc($nomes)):
?>
```

```
<li><a href="#pessoa-<?php echo $nome['codigo'];?>?"  
onclick="javascript:preenche('<?php echo $nome['nome'];?>');"><?php  
echo $nome['nome'];?>?></a></li>  
  
<?php  
endwhile;  
  
?>  
  
</ul>  
  
<?php  
else:  
  
?>  
  
Nenhuma pessoa encontrada.  
  
<?php  
endif;  
  
?>
```

# Biblioteca Virtual



Veja a seguir os serviços oferecidos pela Biblioteca Virtual aos alunos a distância:

- Pesquisa a publicações online  
[www.unisul.br/textocompleto](http://www.unisul.br/textocompleto)
- Acesso a bases de dados assinadas  
[www.unisul.br/bdassinadas](http://www.unisul.br/bdassinadas)
- Acesso a bases de dados gratuitas selecionadas  
[www.unisul.br/bdgratis](http://www.unisul.br/bdgratis)
- Acesso a jornais e revistas on-line  
[www.unisul.br/periodicos](http://www.unisul.br/periodicos)
- Empréstimo de livros  
[www.unisul.br/emprestimos](http://www.unisul.br/emprestimos)
- Escaneamento de parte de obra\*

Acesse a página da Biblioteca Virtual da Unisul, disponível no EVA e explore seus recursos digitais.

Qualquer dúvida escreva para [bv@unisul.br](mailto:bv@unisul.br)

---

\* Se você optar por escaneamento de parte do livro, será lhe enviado o sumário da obra para que você possa escolher quais capítulos deseja solicitar a reprodução. Lembrando que para não ferir a Lei dos direitos autorais (Lei 9610/98) pode-se reproduzir até 10% do total de páginas do livro.



# UnisulVirtual

A sua universidade a distância

