



Guia de Estudo

**DESENVOLVIMENTO DE APLICAÇÕES
PARA INTERNET**



SABE – Sistema Aberto de Educação

Av. Cel. José Alves, 256 - Vila Pinto
Varginha - MG - 37010-540
Tele: (35) 3219-5204 - Fax - (35) 3219-5223

Instituição Credenciada pelo MEC – Portaria 4.385/05

Centro Universitário do Sul de Minas - UNIS
Unidade de Gestão da Educação a Distância – GEaD

Mantida pela
Fundação de Ensino e Pesquisa do Sul de Minas - FEPESMIG

Varginha/MG





004.699 PRADO, Alan Souza.
P896g Guia de Estudo – Desenvolvimento de
Aplicações para Internet. Alan Souza Prado.
Varginha: GEaD-UNIS, 2009.
88p.

1. Internet 1. 2. PHP 1. 3. Dinamismo 1. I.
Título.

Todos os direitos desta edição reservados ao Sistema Aberto de Educação – SABE.
É proibida a duplicação ou reprodução deste volume, ou parte do mesmo, sob qualquer meio, sem
autorização expressa do SABE.





REITOR

Prof. Ms. Stefano Barra Gazzola

GESTOR

Prof. Ms. Wanderson Gomes de Souza

Supervisora Técnica

Prof^a. Ms. Simone de Paula Teodoro Moreira

Design Instrucional

Prof. Celso Augusto dos Santos Gomes

Jacqueline Aparecida Silva

Coord. do Núcleo de Comunicação

Renato de Brito

Coord. do Núcleo de Recursos Tecnológicos

Lúcio Henrique de Oliveira

Equipe de Tecnologia Educacional

Prof^a. Débora Cristina Francisco Barbosa

Danúbia Pinheiro Teixeira

Revisão ortográfica / gramatical

Gisele Silva Ferreira

Autor

ALAN SOUZA PRADO

Bacharel em Ciência da Computação (2001) – UNIS-MG, Varginha - MG. Especialista em Redes de Computadores (2004) – UNIS-MG, Varginha. Especializando em MBA em Gestão de TI (2008) – UNIS-MG, Varginha. Mestrando em Modelagem Matemática e Computacional – CEFET, Belo Horizonte. Atualmente ocupa o cargo de Assessor de Tecnologia da Informação do Centro Universitário do Sul de Minas - UNIS, Varginha - MG. Atua também como coordenador do Curso Superior de Tecnologia em Informática e do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas e é professor em cursos de Graduação, presencial e a distância, neste mesmo Centro Universitário.





Ícones



REALIZE. Determina a existência de atividade a ser realizada.
Este ícone indica que há um exercício, uma tarefa ou uma prática para ser realizada. Fique atento a ele.



PESQUISE. Indica a exigência de pesquisa a ser realizada na busca por mais informação.



PENSE. Indica que você deve refletir sobre o assunto abordado para responder a um questionamento.

Conclusão

CONCLUSÃO. Todas as conclusões, sejam de idéias, partes ou unidades do curso virão precedidas desse ícone.



IMPORTANTE. Aponta uma observação significativa. Pode ser encarado como um sinal de alerta que o orienta para prestar atenção à informação indicada.



HIPERLINK. Indica um link (ligação), seja ele para outra página do módulo impresso ou endereço de Internet.



EXEMPLO. Esse ícone será usado sempre que houver necessidade de exemplificar um caso, uma situação ou conceito que está sendo descrito ou estudado.



SUGESTÃO DE LEITURA. Indica textos de referência utilizados no curso e também faz sugestões para leitura complementar.



APLICAÇÃO PROFISSIONAL. Indica uma aplicação prática de uso profissional ligada ao que está sendo estudado.



CHECKLIST ou PROCEDIMENTO. Indica um conjunto de ações para fins de verificação de uma rotina ou um procedimento (passo a passo) para a realização de uma tarefa.



SAIBA MAIS. Apresenta informações adicionais sobre o tema abordado de forma a possibilitar a obtenção de novas informações ao que já foi referenciado.



REVENDO. Indica a necessidade de rever conceitos estudados anteriormente.





Sumário

APRESENTAÇÃO	8
INTRODUÇÃO.....	10
O QUE É O PHP?	10
BREVE HISTÓRICO DO PHP	11
O QUE PODEMOS FAZER COM O PHP?	11
ALGUMAS VANTAGENS DO PHP.....	12
PRÉ-REQUISITOS	12
SCRIPTS <i>WEB</i> DO LADO DO SERVIDOR.....	13
HTML ESTATICA	13
TECNOLOGIAS DO LADO CLIENTE	16
SCRIPTS DO LADO SERVIDOR	18
PORQUE SCRIPT DO LADO SERVIDOR?	21
INSTALANDO O PHP	23
INSTALAÇÃO DO PHP NO LINUX	23
INSTALAÇÃO DO PHP NO WINDOWS	24
ADICIONANDO O PHP À HTML	26
SUA HTML JÁ É COMPATÍVEL COM PHP!	26
ESCAPANDO DA HTML	26
FORMA CANÔNICA.....	27
FORMA ABREVIADA.....	27
ESTILO ASP.....	27
ESTILO SCRIPT HTML	27
SINTAXE BÁSICA	29
PHP E LINGUAGEM C	29
PHP NÃO CONSIDERA MAIS DE UM ESPAÇO EM BRANCO.....	29
DISTINÇÃO ENTRE LETRAS MAIÚSCULAS E MINÚSCULAS	30
INSTRUÇÕES SÃO TERMINADAS POR PONTO-E-VÍRGULA	31
EXPRESSÕES SÃO AVALIADAS	32
PRECEDÊNCIA.....	32
COMENTÁRIOS.....	33
VARIÁVEIS	34
ESCOPO DE VARIÁVEIS.....	36
ESCOPO DE VARIÁVEIS EM FUNÇÕES.....	37
ESCOPO DE VARIÁVEIS ENTRE TAGS	37
CONSTANTES.....	38
FUNÇÕES DE SAÍDA	38
ASPAS SIMPLES E ASPAS DUPLAS.....	39
QUEBRA DE LINHAS	40
TIPOS DE VARIÁVEIS NO PHP	43
ATRIBUIÇÃO POR CONTEXTO	44
RESUMO DOS TIPOS	44
TESTANDO OS TIPOS	56
OPERADORES LÓGICOS.....	62
PRECEDÊNCIA DOS OPERADORES LÓGICOS	62





OPERADORES DE COMPARAÇÃO	63
COMPARAÇÃO DE STRINGS	63
ESTRUTURAS DE CONTROLE	64
DESVIANDO	64
LOOPS	66
UTILIZANDO FUNÇÕES	70
SINTAXE BÁSICA PARA UTILIZAR UMA FUNÇÃO	70
DEFININDO AS SUAS PRÓPRIAS FUNÇÕES	72
INCLUDE E REQUIRE	73
PASSANDO INFORMAÇÕES ENTRE PÁGINAS	74
ARGUMENTOS GET	75
ARGUMENTOS POST	77
SESSÃO EM PHP	79
COMO É O FUNCIONAMENTO DAS SESSÕES EM PHP	79
COOKIE EM PHP	81
COMO É O FUNCIONAMENTO DE UM COOKIE EM PHP	81
PHP COM MYSQL	83
FUNÇÃO DE CONEXÃO COM O SERVIDOR MYSQL	83
FUNÇÃO DE SELEÇÃO DE BANCO DE DADOS COM O SERVIDOR MYSQL	84
EXECUTANDO “QUERIES” EM PHP	84





APRESENTAÇÃO

"Falta de tempo é desculpa daqueles que perdem tempo por falta de métodos."

ALBERT EINSTEIN

Prezado(a) aluno(a),

Este será seu Guia de Estudos da disciplina Desenvolvimento de Aplicações para Internet ministrada para o curso de Bacharelado em Sistemas de Informação à distância pelo Centro Universitário do Sul de Minas – UNIS-MG.

Esta disciplina tratará de uma importante área da Tecnologia de Informação que é a utilização de tecnologias para a disponibilização de informações, conteúdos, aplicações e sistemas que serão veiculados através da utilização do *Quarto Canal*¹.

No intuito de mensurar esta importante relação entre as Tecnologias da Informação disponíveis e suas aplicações no Quanto Canal o limite será o do ambiente em que, inicialmente, se está inserido, no caso o Brasil, sendo assim de acordo com dados do *Ibope/NetRatings*, apresentados em março de 2008, existem 22,7 milhões de internautas com acesso residencial a internet.

Este crescimento representa 56,7% em relação ao mesmo período do ano anterior, ainda de acordo com a pesquisa o Brasil vive “o maior boom” de crescimento em número de acessos residenciais à Internet desde 2000.

“O Ibope atualizou o número de pessoas com acesso à Internet em qualquer local -- residência, telecentros ou *lan houses* -- para 40 milhões no último trimestre de 2007, o que equivale a um crescimento de 21,27 por cento sobre o mesmo período de 2006, quando era de 32,9 milhões de pessoas.

O tempo médio de navegação do brasileiro em casa subiu 3 horas e 17 minutos em fevereiro deste ano, comparado ao mesmo mês de 2007, para 22 horas e 24 minutos, o que coloca o Brasil como o país de maior tempo de navegação entre os pesquisados, seguido pelos Estados Unidos, cujo tempo médio foi 19 horas e 52 minutos.”(IT WEB, 2008)

¹ Quarto Canal = Internet.





Tendo como base este importante cenário, a disciplina será conduzida no intuito de apresentar uma ferramenta poderosa no desenvolvimento de aplicações pela internet juntamente com ferramentas de banco de dados o que auxiliará no dinamismo, interação e desenvolvimento de sistemas que são veiculados pela Internet.





1

INTRODUÇÃO

Será tratado neste momento informações sobre a linguagem de programação que será utilizada nesta disciplina, características e pré-requisitos.

A linguagem PHP – *Hipertext Preprocessor* será utilizada para aplicar os conceitos das linguagens de programação para Internet e a criação de aplicações práticas.

O QUE É O PHP?

É uma Linguagem de Programação que cria scripts, executados do lado do servidor, para aplicações *Web* e esses *scripts* são embutidos no código HTML.

O PHP é compatível com os melhores, mais conhecidos e utilizados Servidores *Web* (Principalmente o Apache). É importante compreender que os scripts em PHP são interpretados “pelo Servidor” à medida que as páginas são requisitadas pelo usuário.

Alguns dizem que o PHP é “... o ASP de código fonte aberto”, pelo fato da semelhante funcionalidade ou algum conceito da *Microsoft*. Mas na realidade isso é um conceito falso, pois o PHP foi desenvolvido antes do ASP. Podemos entender o PHP como pequenos programas ou *scripts* executados dentro de páginas HTML no servidor antes de serem enviados ao navegador. Para exemplificar podemos imaginar a inclusão de cabeçalhos e rodapés comuns a todas as páginas de um Site ou o armazenamento de informações, postadas por um formulário, em um Banco de Dados.

A aparência e/ou o *Layout* não estão relacionados diretamente ao PHP, o PHP torna-se invisível ao usuário final, pois o resultado da execução do script em PHP é o puro e simples HTML.





BREVE HISTÓRICO DO PHP

Em 1994, Rasmus Lerdorf criou um *WRAPPER* de CGI que auxiliava a monitorar as pessoas que acessavam seu *Site* pessoal. No ano seguinte montou um pacote chamado “*Personal Home Page Tools* (Conhecido também como *PHP Construction Kit*) devido à demanda de usuários que por acaso ou por indicações, deparavam-se com o PHP”.

Logo após foi lançada uma nova versão com o nome de PHP/FI e incluía o *Form Interpreter*, que analisava sintaticamente consultas SQL.

Em 1997 o PHP era utilizado em aproximadamente 50.000 *Sites* e assim tornava-se uma estrutura muito grande para ser administrada por apenas uma pessoa.

Em 1998 juntamente com a publicidade maciça do código-aberto, houve a fase de expressivo crescimento do PHP chegando, no 4º trimestre deste ano, a mais de 100.000 *Sites* que utilizavam PHP de alguma forma.

Na metade de 2002 já existia mais de dois milhões de *Sites* que utilizavam PHP, em 2003 existia mais de nove milhões, em 2004 dezesseis milhões e em 2007?



Acesse: <http://www.php.net/usage.php>

O QUE PODEMOS FAZER COM O PHP?

Quando se fala em desenvolvimento de aplicações para *WEB* utilizando PHP tem que se ter a idéia de que qualquer programa desenvolvido em outra linguagem (ASP, .NET, CGI e etc...) sob a estrutura da *web* pode ser feito também em PHP, como por exemplo coletar dados de um formulário, gerar páginas dinamicamente, enviar e receber *cookies* ou trabalhar com sessões.

Podemos caracterizar como sendo uma das mais importantes características do PHP o suporte quase todos os banco de dados disponíveis, como *dBase*, *Interbase*, *mSQL*, *mySQL*, *Oracle*, *Sybase*, *PostgreSQL* e vários outros. É extremamente simples construir uma página baseada em um banco de dados com PHP.





O PHP tem suporte a vários tipos de serviços através da utilização de protocolos como IMAP, SNMP, NNTP, POP3 e, logicamente, HTTP. Podemos também abrir *sockets*² e interagir com outros protocolos.

ALGUMAS VANTAGENS DO PHP

- Sem custo;
- Conjunto Apache/PHP/MySql executam perfeitamente em Hardwares de processamento inferior e baratos;
- PHP sendo uma linguagem de criação de scripts embutida torna-se agradável por não precisar ser compilada em código binário antes de ser utilizada, bastando ser escrita e executada;
- O PHP é interpretado, embora exista uma pré-compilação de nível intermediário para maior velocidade dos scripts mais complexos;
- É Multiplataforma, executa nativamente em todas as versões populares do Unix (incluindo Mac OS) e Windows;
- Estável, o servidor não necessita ser reinicializado freqüentemente e não sobre alterações radicais e incompatíveis de uma versão para outra;
- Rápido, o PHP é surpreendente rápido, principalmente quando executado como um módulo Apache no Unix;
- Comunica-se facilmente com outros programas e protocolos, normalmente possui funções nativas da linguagem, não necessitando a instalação de “pacotes” para determinadas aplicações.

PRÉ-REQUISITOS

Um bom programador em PHP precisa sentir-se muito à vontade quando edita HTML manualmente;

Mínima experiência em Linguagem de Programação com sintaxe semelhante ao C.

² Socket - É um canal de comunicação entre computadores em uma rede e identifica uma conexão entre eles, normalmente entre um cliente e um servidor. Através dos sockets os computadores podem trocar informações através de uma rede.





2

SCRIPTS *WEB* DO LADO DO SERVIDOR

Iremos tratar neste capítulo assuntos referentes a scripts do lado do servidor e seus relacionamentos com tecnologias de HTML estática e salientar algumas tecnologias comuns que ocorrem do lado do cliente.

HTML ESTÁTICA

O mais simples modelo *Web* é uma pagina completamente estática que é baseada em texto e inteiramente escrita em HTML. A Figura 1 é um exemplo simples de página somente HTML.



Figura 1 - Exemplo de HTML estática





O código-fonte a seguir é referente à Figura 1.

Utilizando Somente HTML.



```
<html>
<head>
<title>Página Estática</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<font face="Arial, Helvetica, sans-serif">
<div align="center">
Centro Universitário do Sul de Minas - UNIS/MG</div><br>
<b>Curso Superior de Bacharelado em Sistemas de Informação</b><br>
Acesse:<br>
<a href="http://www.sabe.br">http://www.sabe.br</a>
</font>
</body>
</html>
```

Depois que o computador do cliente faz uma solicitação HTTP para o Servidor pela *Web* ou por uma *Intranet*, conforme Figura 2, o Servidor simplesmente enviar todo o “texto” encontrado no arquivo para este ser interpretado pelo *Browser* do computador cliente.



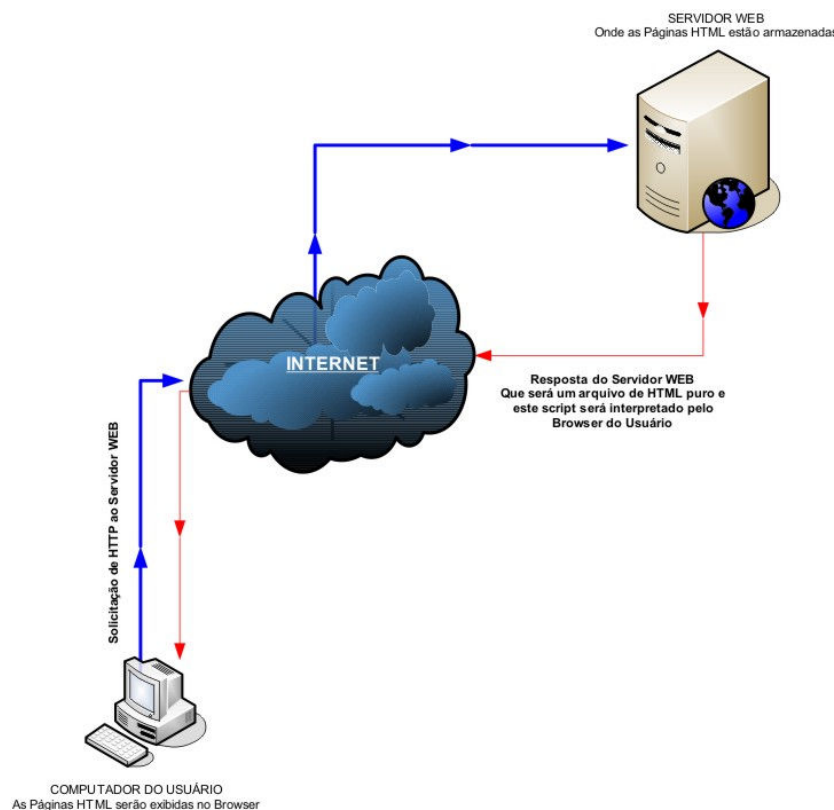


Figura 2 - Solicitação e Resposta de HTTP simples

Conforme mencionado, após estes dados retornarem à máquina do cliente, o *Browser* fará o melhor possível para representar a página de acordo com a sua compreensão de acordo com o código-fonte. Vale ressaltar que o conteúdo do arquivo HTML que está no servidor será exatamente o mesmo do código-fonte na página do cliente.

Uma HTML estática, muito simples, como o código apresentado, oferece certas vantagens, tais como:

- É exibida adequadamente por qualquer navegador.
- É exibida adequadamente por vários tipos de dispositivos.
- As solicitações são cumpridas rapidamente e utilizam recursos mínimos.
- HTML é fácil de aprender ou de produzir automaticamente.
- Os desenvolvedores Web pode fazer pequenas e rápidas alterações nas páginas individuais.
- É claro também que HTML estática tem suas limitações o que acarreta em desvantagens, por exemplo:
- Dificulta o controle do design e layout.





- Não permite um escalonamento³ para um grande número de páginas.
- Não é muito interativa.
- Não suporta alterações rápidas de conteúdo ou de personalizações.
- Não é muito atraente.

Por estes fatores HTML estática tornou-se uma marca de amadorismo ou rigor ideológico (como é o caso de algumas *home pages* de profissionais das áreas relacionadas à Tecnologia da Informação que todas as páginas da *Web* devem ser em HTML e ser legíveis em todos os dispositivos).

Várias tecnologias adicionais foram desenvolvidas em resposta a essas limitações, incluindo *JavaScript*, *Cascading Style Sheets* e *applets Java* no *script* do lado cliente e do lado servidor com conectividade a banco de dados.



Você pode evitar muita dor de cabeça se aproveitar seu tempo entendendo exatamente qual é a funcionalidade de cada uma dessas tecnologias e o que elas podem ou não adicionar ao seu *site Web*.

A principal questão que deve ser feita a respeito de qualquer tarefa relacionada à *Web* é: **ONDE A COMPUTAÇÃO ESTÁ ACONTECENDO, NO CLIENTE OU NO SERVIDOR?**

TECNOLOGIAS DO LADO CLIENTE

As adições mais comuns à HTML simples estão do lado do cliente. Dentre estes suplementos existem extensões de formatação como *Cascading Style Sheets* (CSS) e *Dynamic HTML* (DHTML), e linguagens de *scripts* do lado do cliente como *JavaScript*, *applets Java* e *Flash*. O suporte para todas essas tecnologias é ou não é, dependendo do caso, integrado ao navegador *Web*. Vejamos alguns exemplos na Tabela 1.

³ Escalonamento – vem de escalabilidade que é uma característica desejável em todo o sistema, em uma rede ou em um processo, que indica sua habilidade de manipular uma porção crescente de trabalho de forma uniforme, ou estar preparado para crescer.





Extensões de HTML do lado Cliente		
Tecnologias do lado cliente	Utilização Principal	Efeitos do Exemplo
Cascading Style Sheets (CSS) e Dynamic HTML(DHTML)	Formatar páginas: controlar tamanho, cor, posicionamento, layout e sincronização de elementos	Sobrepor fontes de diferentes cores/tamanhos Camadas, posicionamento exato
Script do lado cliente (JavaScript, VBScript)	Tratamento de eventos: controlar consequências de eventos definidos	Link que muda de cor no mouseover Calculadora de hipoteca
Applets Java	Distribuir pequenas aplicações independentes	Logotipo móvel Palavras cruzadas
Animações em Flash	Animação	Desenhos animados de curta metragem

Tabela 1 - Tecnologias do lado cliente

O código-fonte a seguir é referente à Figura 1

Utilizando HTML e CSS.



```
<html>
<head>
<title>Página Estática</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<style type="text/css">
<!--
.style1 {font-family: Arial, Helvetica, sans-serif}
.style2 {font-family: Arial, Helvetica, sans-serif; font-weight: bold; }
-->
</style>
</head>
<body>
<div align="center" class="style1">
Centro Universitário do Sul de Minas - UNIS/MG</div><br>
<span class="style2">Curso Superior de Bacharelado em Sistemas de
Informação</span><br>
Acesse:<br>
<a href="http://www.sabe.br" class="style1">http://www.sabe.br</a>
</body>
</html>
```





Copie o código do exemplo da página 11, abra o notepad e salve com a extensão htm.

Depois copie o código do exemplo da página 14, abra o notepad e salve com outro nome e com a extensão htm.

Abra os dois arquivos em um navegador de sua preferência e aponte as diferenças percebidas por você, poste sua opinião no portfólio vinculada a Atividade 1.

É importante salientar que as tecnologias do lado cliente é dependem inteiramente do navegador, o que pode ser considerada uma situação não favorável. Existem grandes variações na capacidade de cada navegador e até entre versões de uma mesma marca. Algumas pessoas também têm o hábito de alterar as configurações de seus navegadores de maneiras estranhas, por exemplo, a desativação do *JavaScript* por razões de segurança, o que torna impossível visualizar *sites* que utilizam excessivamente este recurso para navegação.

SCRIPTS DO LADO SERVIDOR

Os *scripts* do lado do servidor são a parte mais atraente e fascinante do desenvolvimento *Web*. Ao contrário da HTML os *scripts* do lado do servidor são invisíveis para os usuários.

Abaixo a Figura 2 representa o esquema do fluxo de dados dos *scripts* do lado do servidor.



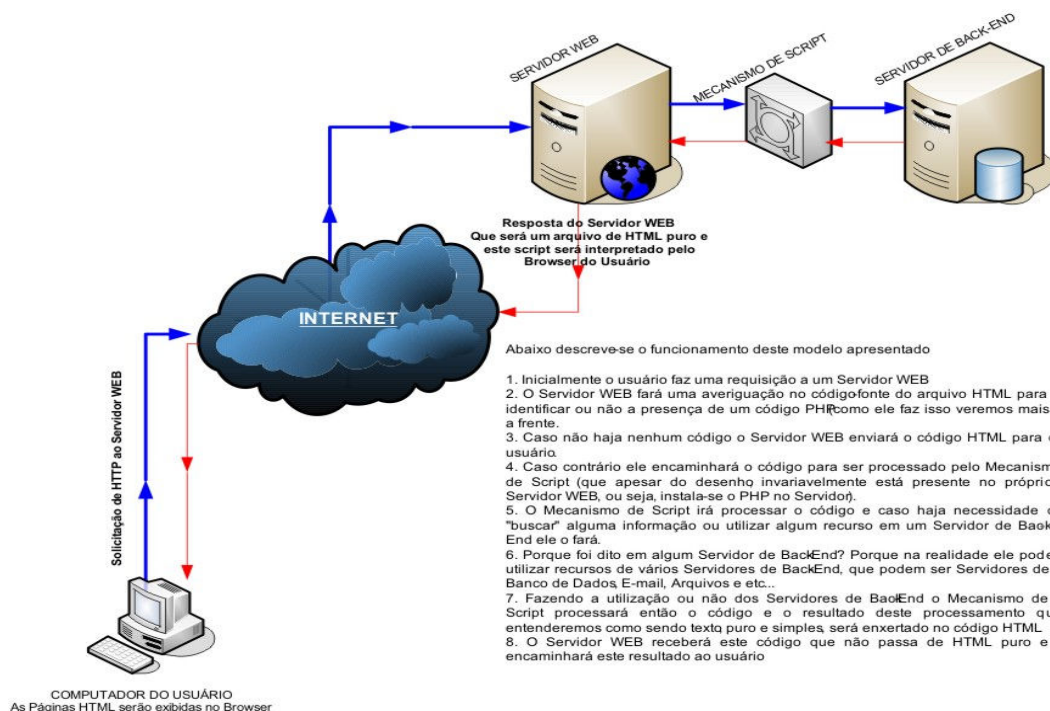


Figura 3 - Tarefas do lado do Servidor

Os *scripts* do lado do servidor *Web* tratam principalmente de conexões de *sites Web* aos Servidores de *Back-End*, como os banco de Dados. Isso permite os seguintes tipos de comunicação de duas vias

Servidor para Cliente: Páginas *Web* podem ser montadas a partir da saída do servidor de *Back-End*.

Cliente para Servidor: Informações inseridas pelo cliente podem ser manipuladas.

Exemplos comuns de interação entre cliente e servidor são formulários *on-line* e algumas listas suspensas (*Drop-Down*) que o *script* monta dinamicamente no servidor.

Os produtos de *script* do lado do servidor consistem de duas partes principais: a linguagem de *script* e o Mecanismo de *script* (que pode ou não ser integrado ao Servidor *Web*). O mecanismo analisa sintaticamente e interpreta páginas escritas na linguagem.

A Figura 4 mostra um exemplo de *script* do lado do Servidor – uma página montada instantaneamente (*on-the-fly*) a partir de um banco de dados, seguido pelo código-fonte do lado Servidor e pelo código-fonte do lado cliente.



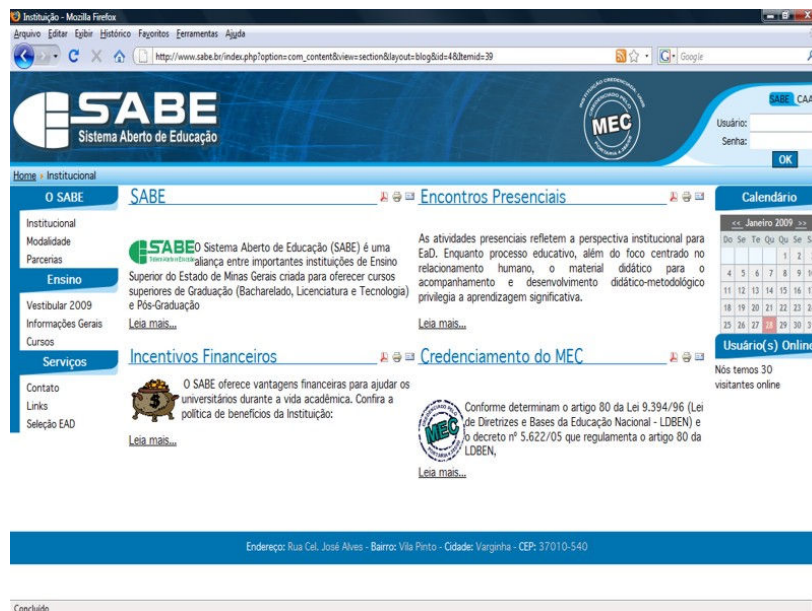


Figura 4 - Exemplo de Script do lado Servidor

A partir de então, pode-se perceber que os exemplos apresentados não possibilitam a visualização dos *scripts* de Servidor na máquina do cliente. Depois que o código sai do Servidor *Web*, o código aparece na outra extremidade como HTML puro, isso também significa que não é possível afirmar qual linguagem de *script* do lado Servidor foi utilizada, a menos que haja algo indicando no cabeçalho ou URL.



Existem métodos do lado cliente e métodos do lado Servidor para a realização de várias tarefas. Enviar e-mail, por exemplo, no lado cliente é abrir o software de cliente de correio com uma mensagem de correio eletrônico em branco, pré-endereçada, depois que o usuário clicar em um link MAILTO, enquanto no lado Servidor faz-se o usuário preencher um formulário, cujo conteúdo é formatado como correio eletrônico e enviado via um servidor SMTP. Pode-se escolher entre métodos de cliente e Servidor para identificação do navegador (*Browser-sniffing*), validação de formulários, listas suspensas e cálculo aritmético.

Às vezes identifica-se pequenas diferenças, mas significativas na funcionalidade (listas suspensas podem ser montadas dinamicamente do lado Servidor, mas não do lado cliente), mas nem sempre é possível.





O que significa dinâmica? Uma distinção básica, frequentemente repetida, existe entre uma página *Web* “estática” e a “dinâmica”, porém dinâmica pode significar quase todo que vai além da HTML básica. Os desenvolvedores *Web* utilizam o tempo para descrever funções do lado cliente e do Servidor. No cliente, o termo pode significar apresentações multimídia, manchetes que podem ser roladas, páginas que se atualizam automaticamente ou elementos que aparecem e desaparecem. No Servidor, o termo geralmente significa que o conteúdo é montado instantaneamente.

PORQUE SCRIPT DO LADO SERVIDOR?

O *script* do lado cliente parece bom, mas no Servidor permanece bom.

Um aspecto a ser levado em conta é que um *script* no lado Servidor não vai lhe ajudar caso haja necessidade de utilização de tomadas 3D em tempo real, ou seja, quanto mais rápida a necessidade de resposta e quanto mais intensa for à utilização de imagens em um projeto, menos adequado é o PHP para este projeto. No momento a *Web* ainda pode ser considerada um canal lento para esses propósitos (embora alguns pioneiros da banda larga estejam ávidos para mudar isto).

Por outro lado, as linguagens do lado Servidor, como o PHP, atendem perfeitamente à maioria dos aspectos verdadeiramente úteis a *Web*, como por exemplo:

- Sites de conteúdo (tanto produção como exibição).
- Recursos comunitários (Fóruns, BBSs⁴ e assim por diante).
- Correio Eletrônico (correio da *Web*, encaminhamento de correio, envio de correio a partir de uma aplicação *Web*).
- Sistemas de atendimento ao cliente e suporte técnico.
- Redes de publicidade.
- Aplicações de negócios distribuídos pela *Web*.
- Diretórios e listas de membros de associações.
- Pesquisas, enquetes e testes.
- Preenchimento e envio de formulários *on-line*.

⁴ BBS - (acrônimo inglês de *bulletin board system*) é um sistema informático, um software, que permite a ligação (conexão) via telefone a um sistema através do seu computador e interagir com ele, tal como hoje se faz com a internet.





- Tecnologias personalizadas.
- *Groupware* (software de grupo de trabalho).
- Catálogos e brochuras *on-line* e *sites* informacionais.
- Qualquer outra aplicação que precise conectar um Servidor de *Back-End* (Banco de dados, LDAP e etc...) a um Servidor *Web*.

Agora que já se obteve uma boa compreensão das diferenças entre as tecnologias do lado cliente e do lado Servidor, vamos à parte prática.





3

INSTALANDO O PHP

Antes de qualquer coisa é importante entendermos que a instalação do PHP se faz, em seus reais fins, em um Servidor *Web*. Normalmente esta aplicação é feita sobre uma plataforma *Linux* de qualquer distribuição e “roda” no conjunto *Apache* (Servidor Web), PHP (Mecanismo de *Script*), *MySQL* (Sistema de gerenciamento de banco de dados - SGBD) e o *phpMyadmin* (programa de computador desenvolvido em PHP para administração do *MySQL* pela Internet).

Entretanto para iniciarmos nossos estudos e desenvolvimentos em PHP recomendamos que você instale a principio na sua máquina de desenvolvimento pessoal, de modo que tenha mais contato e entendimento do ambiente de desenvolvimento.

Pensando em inicialmente termos um ambiente para desenvolvimento é válido salientar que o PHP pode ser instalado em qualquer plataforma: *UNIX*, *Mac Os* ou *Windows*.

INSTALAÇÃO DO PHP NO LINUX

Abaixo o passo a passo da instalação do servidor *Apache*, PHP, *MySQL* e *phpMyadmin* na distribuição *Linux Ubuntu*.

Atualize a lista do apt:

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

Agora chegou a hora de instalar o *Apache 2* e o PHP 5.

```
$ sudo apt-get install apache2 php5
```

Para testar se tudo foi instalado corretamente, crie o arquivo *index.php* com as informações do PHP5:

```
$ sudo vi /var/www/index.php
```

Copie o conteúdo abaixo para o novo arquivo:





```
<?PHP
```

```
    phpinfo();
```

```
?>
```

Salve o arquivo e acesse o endereço `http://localhost/`. Se as informações do PHP aparecerem, tudo está instalado corretamente.

Com o *Apache* e o PHP rodando corretamente, é o momento de instalar o *MySQL*.

```
$ sudo apt-get install mysql-server-5.0 php5-mysql
```

Agora troque a senha do administrador *root* do *MySQL*:

```
$ sudo mysqladmin -u root seu_password
```

Para que o serviço rode corretamente junto com o PHP5, reinicie o *Apache*:

```
$ sudo /etc/init.d/apache2 restart
```

Para administrarmos o *MySQL*, instale o *phpMyAdmin*, que é uma das melhores ferramentas para este fim.

```
$ sudo apt-get install phpmyadmin
```

Acesso o endereço `http://localhost/phpmyadmin/` e forneça o *login root* e a senha que você definiu no *MySQL*.

Pronto, seguindo estes 10 passos você já poderá iniciar os seus desenvolvimentos. No entanto, vale ressaltar que somente os arquivos que estiverem salvos no caminho `/var/www/` serão interpretados, pois esta pasta é considerada a pasta do servidor, fora dela nada será executado.

INSTALAÇÃO DO PHP NO WINDOWS

É possível fazer a instalação do PHP, *Apache*, *MySQL* e o *phpMyAdmin* por meio de pacotes ou manualmente, no caso optarmos pelos pacotes os quais os mais utilizados e que também são gratuitos pode ser encontrados nos seguintes endereços para *download*:

- PHPTriad

http://downloads.sourceforge.net/phptriad/phptriad2-2-1.exe?modtime=1013817600&big_mirror=0





- EasyPHP

http://sourceforge.net/project/showfiles.php?group_id=14045&package_id=103438&release_id=651947

- XAMPP

http://sourceforge.net/project/downloading.php?group_id=61776&use_mirror=ufpr&filename=xampp-win32-1.7.0-installer.exe&72111003

Sugerimos a utilização do XAMPP pela sua estabilidade, facilidade de instalação, facilidade de utilização, por apresentar resultados interessantes, além de ser o mais utilizado pelos programadores PHP que utilizam como ambiente de desenvolvimento a plataforma *Windows*.



Acesse o SABE e vá até a midiateca. Acesse o vídeo “Instalando o PHP no Windows” que está na pasta “DAI - Desenvolvimento de Aplicações para Internet - Prof. Alan Souza Prado” e realize os procedimentos conforme indicado.





4

ADICIONANDO O PHP À HTML

Após as explicações e preparações preliminares começaremos a criar nossos primeiros *scripts* em PHP, aprenderemos o modo do PHP, as *tags* de PHP e a incluir outros arquivos nos *scripts* em PHP.

SUA HTML JÁ É COMPATÍVEL COM PHP!

O PHP é perfeitamente compatível com HTML. Na realidade, de modo geral, o PHP é incorporado pelo HTML, ele “pega carona” com algumas partes mais inteligentes do padrão HTML, como formulários e cookies.

Tudo que é compatível com o HTML do lado cliente também é com PHP. É válido salientar que é possível utilizar qualquer técnica de desenvolvimento de Páginas Web e simplesmente adicionar o PHP a elas, caso sinta-se a vontade para trabalhar com conjuntos de softwares gráficos e de multimídia, vá em frente, não se iniba. A questão central é que não é necessário trocar de ferramenta nem a ordem do fluxo de trabalho para se ter sucesso com o PHP.

ESCAPANDO DA HTML

A questão principal a ser analisada é que como o analisador de sintaxe de PHP reconhece o código PHP dentro do seu documento HTML? O acontece é que nós somos quem determinamos quando o programa deve entrar em ação, utilizando as *tags* de PHP especiais no início e no final de cada seção de PHP. Este processo recebe o nome de escapando da HTML ou, alternativamente, escapando para o PHP.

Tudo dentro destas *tags* é entendido pelo analisador de sintaxe de PHP como sendo código do PHP. Tudo fora destas *tags* não será relacionado ao servidor e





será simplesmente ignorado, sendo o cliente quem determina se é HTML, Java Script ou outra coisa.

O PHP é incorporado ao HTML, ou seja, páginas em PHP são páginas HTML que incorporam o modo PHP quando necessário.

Para isso indicamos, utilizando *tags*, na página criada que a partir daquele ponto, até o fechamento desta *tag*, deve-se executar o código no servidor.

Há quatro estilos de *tags* de PHP e diferentes raciocínios para utilizá-los. Entretanto parte da decisão é simplesmente uma questão de preferência pessoal, sendo assim fica ao nosso critério individual escolher qual usar.

FORMA CANÔNICA

<?PHP

Comandos

?>

FORMA ABREVIADA

<?

Comandos

?>

ESTILO ASP

<%

Comandos

%>

ESTILO SCRIPT HTML

<script language="PHP">

Comandos

</script>

O estilo de *tags* mais utilizado é o apresentado no item 5.2, devemos atentar que para utilizá-lo é necessário habilitar a opção *short-tags* na configuração do





PHP, no caso da instalação do Xampp esta opção já vem habilitada. A opção 5.3 serve para facilitar o uso por programadores acostumados à sintaxe de ASP, no entanto utilizá-lo também é necessário habilitá-lo no PHP, através do arquivo de configuração *php.ini*.



Importante ressaltar que em qualquer momento em um *script* PHP, você estará no modo PHP ou fora dele no HTML, não existe a possibilidade de um meio termo, automaticamente estando entre as *tags* do PHP o conteúdo é interpretado como PHP e tudo fora delas é HTML.





5

SINTAXE BÁSICA

É muito importante que percebamos que a linguagem PHP é bastante condescendente. As linguagens de programação variam em termos de rigor de sintaxe que impõem. Meticulosidade excessiva pode ser uma coisa boa, para auxiliar e certificar que o código que se escreve é realmente aquele que se deseja. Entretanto a filosofia do PHP está no outro viés, pois o ideal desta linguagem está relacionada à construção de páginas *Web* “simples e funcionais” e enfatiza mais a conveniência para o programador do que a correção, ele requer o mínimo e faz o melhor para tentar descobrir o que se pretende. Isso implica que certos recursos sintáticos que aparecem em outras linguagens como a **declaração de variáveis e o protótipo de função não são necessários**.

Caso a expressão *parse error* (erro de análise sintática), apareça na tela do browser ao invés da página desejada, isso significa que você violou as regras de tal forma que o PHP desistiu de sua página.

PHP E LINGUAGEM C

A sintaxe do PHP é, genericamente falando, como a Linguagem de Programação C. Caso você já saiba C, facilitará muito, pois, estando em dúvida sobre como uma instrução deve ser escrita, experimente primeiro a maneira como você escreveria em C.

PHP NÃO CONSIDERA MAIS DE UM ESPAÇO EM BRANCO

Não importa a quantidade de caracteres em branco (espaço) que você insere entre as “palavras reservadas” do PHP, pois, um ou vários espaços serão interpretados da mesma forma.





```
<html>
<head>
<title>PHP não considera mais de um espaço em branco </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = 7; // ou
    $var  =    7;
    $var          =

        7;
?>
</body>
</html>
```

Importante percebermos que não importa os caracteres em branco, quantas linhas estamos utilizando, pois, o valor de \$var, nestes casos, vai ser sempre 7.

DISTINÇÃO ENTRE LETRAS MAIÚSCULAS E MINÚSCULAS

Apesar do PHP não ser exigente, é importante percebermos que a linguagem faz distinção na utilização de maiúsculo e minúsculo, principalmente quando usamos variáveis.



```
<html>
<head>
<title>Distinção entre letras maiúsculas e minúsculas </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = "SIM";
    $Var = "NÃO";
?>
</body>
</html>
```

No exemplo acima teremos duas variáveis diferentes com conteúdos diferentes.





Vale ressaltar que diferentemente da Linguagem C, na utilização de nomes de funções e na maioria das construções básicas da linguagem (if, then, else, while e outras) essa distinção não é feita, podendo-se utilizar maiúsculo ou minúsculo.

INSTRUÇÕES SÃO TERMINADAS POR PONTO-E-VÍRGULA

Toda expressão que é passada para a linguagem, como por exemplo a atribuição de valores a variáveis devem ser terminadas por ponto-e-vírgula, que indica o final desta instrução.



```
<html>
<head>
<title>Instruções são terminadas por ponto-e-vírgula </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    echo 'Curso de Bacharelado em Sistemas de Informação';
?>
</body>
</html>
```

Deve-se perceber que linhas de comando, de controle, não necessitam ser terminadas por ponto-e-vírgula



```
<html>
<head>
<title>Instruções são terminadas por ponto-e-vírgula </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    if($x == $x1)
    { //aqui não se utilize ponto-e-vírgula
        echo 'Com Ponto-e-Vírgula'; //aqui se utilize ponto-e-
vírgula
    } //aqui não se utilize ponto-e-vírgula
```





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

EXPRESSÕES SÃO AVALIADAS

Sempre que o interpretador encontra uma expressão no código, essa expressão é imediatamente avaliada. Isso significa que o PHP calcula valores para os menores elementos da expressão e então sucessivamente combina esses valores, que são conectados por operadores ou funções, até produzir um valor inteiro para a expressão.



```
<html>  
<head>  
<title>Expressões são avaliadas </title>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
</head>  
<body>  
<?PHP  
    $total = 4 * 4 + 2 * 6 - 10 / 2;  
    (= 16 + 12 - 5)  
    (= 23)  
    //O valor da variável $total será 23.  
?>  
</body>  
</html>
```

PRECEDÊNCIA

As maneiras particulares como os operadores agrupam as expressões são chamadas de regras de precedência. Com isso operadores que têm precedência mais alta são avaliados primeiro.

Caso queira você pode memorizar as regras, como o fato do * sempre tem precedência mais alta que +, mas o que se deve utilizar no caso de dúvida são os parênteses, para agrupar as expressões.





```
<html>
<head>
<title>Precedência</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $total = 3 + 7 * 3 + 5; // assim $total será igual a 29
    $total = (3 + 7) * (3 + 5); // assim $total será igual a 80
?>
</body>
</html>
```

Você pode estar se perguntando sobre o que acontece quando temos operadores com a mesma precedência?



```
<html>
<head>
<title>Precedência</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $total = 3 / 4 / 5; //O resultado será 0,15 ou 3,75
?>
</body>
</html>
```

Existe uma vasta lista de regras quanto à precedência, mas uma regra é imprescindível para se lembrar que é: a associatividade ocorre normalmente da esquerda para a direita, sendo assim o resultado da expressão acima seria 0,15.

COMENTÁRIOS

A inclusão de comentários dentro dos códigos é invisível para os usuários finais e não são executadas pelo PHP.

Os comentários são de valor inestimável para outras pessoas que lerão o código, pois facilitam a compreensão do que se estava pensando quando o código foi feito, mesmo que essa pessoa seja você mesmo.

Existem três formas de inserir comentários entre seus códigos:





```
<html>
<head>
<title>Comentários</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    /*
    $var = "Olá!!!";
    */
?>
</body>
</html>
```

Esse comentário é inserido para que seja desconsiderado um “bloco” de instruções, textos ou etc. Tudo o que estiver entre /* e */ será considerado como comentário.

Os outros dois tipos, # e //, são utilizados para comentar linhas.



```
<html>
<head>
<title> Comentários</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    #Escrever Oi
    echo "Oi!!!";
?>

<?PHP
    //Escrever Oi
    echo "Oi!!!";
?>
</body>
</html>
```

VARIÁVEIS

A maneira utilizada para se armazenar alguma informação no meio de um programa PHP é através de variáveis.

- O que se deve conhecer sobre variáveis:





- Todas variáveis são precedidas de cifrão(\$).
- O valor mais recente atribuído a uma variável será o atual.
- São atribuídas com o operador =.
- Não precisam ser conhecidas antes da atribuição.
- Não possuem nenhum valor intrínseco além do atual.

Em PHP os tipos das variáveis são associados a valores, sendo assim não é necessário declará-las anteriormente e para utilizar uma variável o primeiro passo é atribuir um valor a ela.

Para atribuir uma variável basta especificar um nome precedido de \$ adicionar o sinal de = e então a expressão.



```
<html>
<head>
<title>Variáveis</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = 5 + 0.26;
?>
</body>
</html>
```

Existe uma função em PHP que testa uma variável para verificar se algum valor lhe foi atribuído. Esta função é chamada **isset**, que retornará um valor booleano *True* caso já tenha sido declarada ou *False* caso não.



```
<html>
<head>
<title>Variáveis</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = "OI";
    if (isset($var))
    {
        echo $var;
    }
    else
    {
        echo "Variável não atribuída";
    }
?>
```





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

Podemos também com o comando **UnSet** restaurar uma variável para o estado não-atribuído.

```
<html>  
<head>  
<title>Variáveis</title>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
</head>  
<body>  
<?PHP  
    $var = "OI";  
    unset($var);  
    if (isset($var))  
    {  
        echo $var;  
    }  
    else  
    {  
        echo "Variável não atribuída";  
    }  
?>  
</body>  
</html>
```



ESCOPO DE VARIÁVEIS

Escopo de variável é o termo que usamos para as regras sobre os nomes (de variáveis ou funções) que possuem o mesmo significado em dois lugares diferentes e nomes que são grafados exatamente da mesma forma e significam coisas diferentes.

Em PHP se você atribuir uma variável no início do *script* implementado, o nome da variável terá o mesmo significado no restante do arquivo e, caso a variável não for re-atribuída, terá o mesmo valor quando o restante do *script* executar (exceto dentro do corpo de funções).

A atribuição de uma variável com o mesmo nome em arquivos diferentes do PHP, nem mesmo em utilizações repetidas do mesmo terão o seu valor afetado.






É visível que em determinadas situações vê-se a necessidade de manter as informações contidas em uma variável por mais tempo para gerar páginas *Web* específicas, na realidade há uma série de técnicas para se conseguir isto, podemos dar exemplo das variáveis GET e POST, armazenar informações em um Banco de dados, associá-las a uma sessão e armazená-las em arquivos na máquina do cliente utilizando *cookies*.

ESCOPO DE VARIÁVEIS EM FUNÇÕES

Variáveis atribuídas dentro de uma função são locais a essa função, a menos que você faça uma declaração especial em uma função ela não terá acesso a essas variáveis globais que são definidas fora das funções, mesmo que estejam no mesmo arquivo.

ESCOPO DE VARIÁVEIS ENTRE TAGS

Uma dúvida freqüente aos iniciantes é se o escopo da variável persiste entre *tags*, por exemplo:



```
<html>
<head>
<?PHP
    $tit = "Título da Página";
?>
<title>
<?PHP
    echo $tit;
?>
</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>

<body>
<?PHP
    echo "O título da página é $tit";
?>
</body>
</html>
```

Sendo assim devemos entender que as variáveis persistem por todo um thread de execução do PHP. Por isso percebemos que o único efeito das tags é





permitir ao mecanismo do PHP saber se você quer que o código seja interpretado como PHP, ou seja, enviado como HTML.

CONSTANTES

O PHP oferece, além das variáveis, constantes que possuem um mesmo valor por todo seu tempo de vida. Importante notar que as constantes não possuem \$ antes de seus nomes e convencionou-se que são escritas em letras maiúsculas. As constantes podem apenas conter valores escalares (números e strings). Possuem escopo global, ou seja, são acessadas em todas as partes do script depois que foram definidas (mesmo dentro de funções).

Podemos definir constantes utilizando a forma define().



```
<html>
<head>
<title>Constantes</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    define(TESTE,8);
?>
</body></html>
```

Desta forma seria atribuído o valor 8 para a constante TESTE em toda parte do código. Não há nenhuma maneira de alterar essa atribuição depois que ela foi feita.

FUNÇÕES DE SAÍDA

A maioria das construções PHP é feita de forma invisível ao usuário, não imprimem nada na saída, a única maneira de seu código PHP exibir qualquer coisa em um browser de usuário é por meio de instruções que imprimem algo para a saída ou chamando funções que chamam instruções de impressão.

As duas construções mais simples para impressão na saída são echo e print. Como resultado podem ser usadas com parênteses ou sem.





ECHO

É a utilização mais simples para imprimir uma string.



```
<html>
<head>
<title>Funções de Saída</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = 'Olá colegas do SABE!!!';
    echo "A variável contém: <br>$var";
?>
</body>
</html>
```

PRINT

Este comando é muito semelhante ao echo com duas diferenças importantes:

- Aceita somente um argumento;
- Diferentemente do echo, retorna um valor 1 se a instrução print foi bem sucedida e 0 caso contrário;

ASPAS SIMPLES E ASPAS DUPLAS

O PHP realiza um pré-processamento de strings entre aspas duplas antes de construir o valor da própria string.

Para strings entre aspas simples o PHP não realiza nenhuma interpolação de variável, e presta atenção só às duas seqüências de escape.

Você pode inserir um ‘ literal no meio da string, “escapando” esse caractere com \‘ e um \ literal “escapando” esse caractere como \\. Com essas duas exceções, o PHP interpreta strings entre aspas simples como strings literais de caracteres que foram digitadas. Caso digite um caractere ‘\$’ nessa string e imprimi-la, visualizará ‘\$’ na janela do navegador.





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

<body>
<?PHP
    echo "Aluno: João da Silva \n";
    echo "Curso: Bacharelado em Sistemas de Informação \n";
    echo "SABE - Sistema Aberto de Educação";
?>
</body>
</html>
```

QUEBRA DE LINHAS

Para entendermos a quebra de linha e preciso distinguir entre saída do PHP (que normalmente é um código HTML) e a maneira como será representada pelo navegador. A forma mais comum é inserir a *tag* de quebra de linha da codificação HTML `
`, ao passo que caracteres de final de linhas em *strings* (`\n`) colocarão apenas interrupções de linha na origem do HTML, mas que não terá nenhum efeito na maneira como o texto será exibido pelo browser.

Código utilizando o `\n`

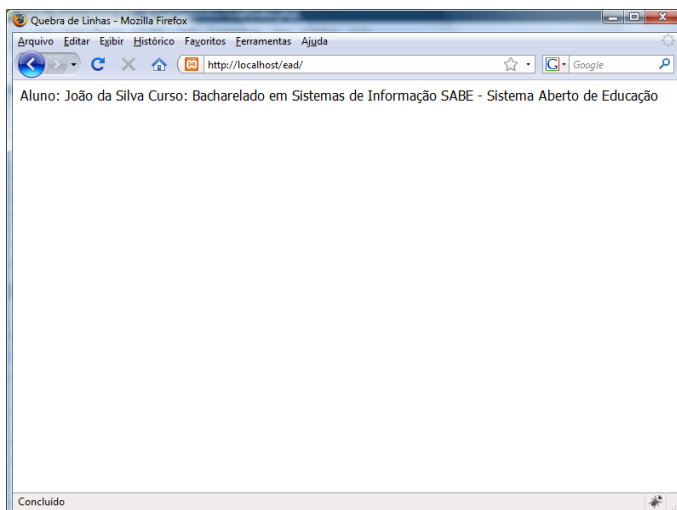


```
<html>
<head>
<title>Quebra de Linhas</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>

<body>
<?PHP
    echo "Aluno: João da Silva \n";
    echo "Curso: Bacharelado em Sistemas de Informação \n";
    echo "SABE - Sistema Aberto de Educação";
?>
</body>
</html>
```

Resultado na Tela do *Browser*





Resultado quando é solicitado ao *Browser* para exibir o código Fonte

```
<html>
<head>
<title>Quebra de Linhas</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
Aluno: João da Silva
Curso: Bacharelado em Sistemas de Informação
SABA - Sistema Aberto de Educação</body>
</html>
```

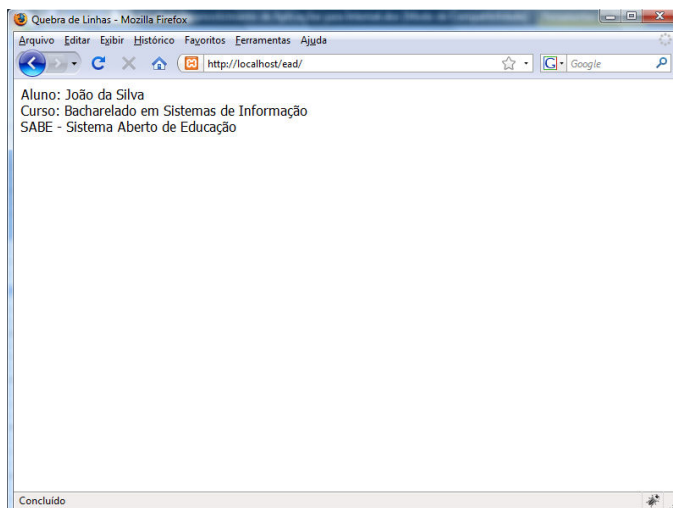
Código utilizando a *Tag*


```
<html>
<head>
<title>Quebra de Linhas</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    echo "Aluno: João da Silva <br>";
    echo "Curso: Bacharelado em Sistemas de Informação <br>";
    echo "SABA - Sistema Aberto de Educação";
?>
</body>
</html>
```



Resultado na Tela do *Browser*





Resultado quando é solicitado ao *Browser* para exibir o código Fonte

```
<html>
<head>
<title>Quebra de Linhas</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
Aluno: João da Silva <br>Curso: Bacharelado em Sistemas de Informação
<br>SABE - Sistema Aberto de Educação</body>
</html>
```





6

TIPOS DE VARIÁVEIS NO PHP

Como vimos no código PHP, não é necessária a declaração de variáveis, para isto basta atribuímos um valor a ela e assim o será entendido e atribuído à mesma.

O PHP utiliza uma checagem de tipos de forma dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do *script*.

O interpretador do PHP decidirá qual o tipo de uma variável para utilizá-la fazendo uma verificação em tempo de execução, converte os tipos automaticamente quando necessário, e podemos confiar que ele fará corretamente. Caso estejamos fazendo cálculos matemáticos com tipos numéricos misturados e o resultado desse cálculo seja do tipo número com ponto flutuante o PHP já fará esta atribuição.



```
<html>
<head>
<title>Tipos de Variáveis no PHP</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $cal = 1 + 1.56165;
?>
</body>
</html>
```

O número 1 será previamente convertido em ponto flutuante (float) antes que a adição seja realizada.





ATRIBUIÇÃO POR CONTEXTO

Analisemos o exemplo:



```
<html>
<head>
<title>Atribuição por contexto </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = substr(123456,2,3);
    echo "O valor de var é $var";
?>
</body>
</html>
```

A função `substr` pega o pedaço de uma string, com o ponto inicial 2 neste caso e o comprimento determinado pelas 3 próximas entradas para a função. Note que ao invés de passarmos uma string para a função passamos um número inteiro, quando isto for executado pelo servidor não ocorrerá nenhum erro.

O valor de `$var` é 345

Como `substr` espera uma string ao invés de inteiro, o PHP converte o número inteiro para string, o que `substr` divide em suas partes componentes.

Devido essa conversão automática é muito difícil o PHP fornecer um erro de tipo.

RESUMO DOS TIPOS

O PHP tem um total de oito tipos: números inteiros, números de dupla precisão, booleanos, strings, arrays, objetos, NULL e recursos.

- Inteiros são os números integrais, sem ponto de fração decimal.
- Números de dupla precisão (*doubles*) são números de ponto flutuante, como 2,156.
- Os booleanos têm apenas dois possíveis valores: VERDADEIRO ou FALSO.
- *NULL* é um tipo especial que só tem o valor *NULL*.
- *Strings* são seqüências de caracteres, como 'Ola, tudo bem'.
- *Arrays* são coleções identificadas e indexadas a outros valores.





- Objetos são instâncias de classe definida pelo programador, as quais podem empacotar tanto outros tipos de valores como funções que são específicas à classe.
- Recursos são variáveis especiais que armazenam referências para recursos externos ao PHP (conexão com banco de dados).

Desses tipos os primeiros cinco são tipos simples e os dois seguintes (*arrays* e objetos) são **tipos compostos**. Os tipos compostos podem conter outros valores arbitrários, o que não é possível com os tipos simples.

O tipo **recursos** é um tipo especial que o programador não manipula diretamente, além de solicitar recursos via funções especiais e passá-los para outras funções que precisam deles.

INTEIROS

São os mais simples e corresponde aos inteiros simples, tanto positivos como negativos.



```
<html>
<head>
<title>Inteiros</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = 486846;
    $Var = -1236 + 1123;
?>
</body>
</html>
```

Os inteiros podem ser lidos de três formatos, que correspondem a base decimal, octal e hexadecimal. O formato decimal e o padrão o octal é indicado com um 0 inicial e o hexadecimal com um 0x. Qualquer dos formatos pode ser positivo ou negativo.



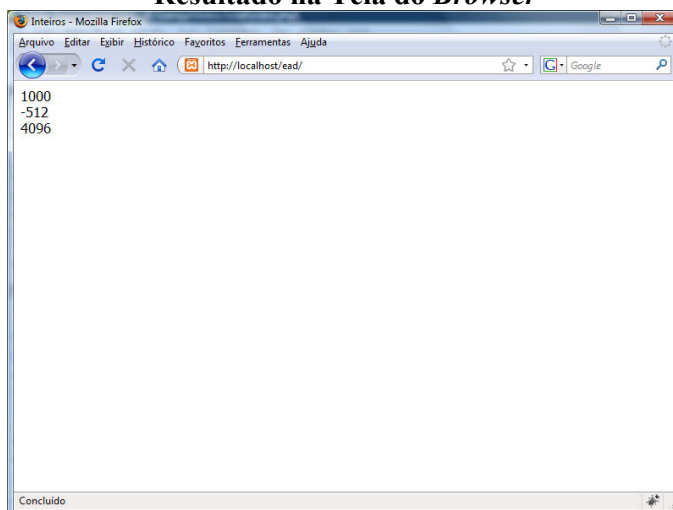
```
<html>
<head>
<title>Inteiros</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    echo $var = 1000;
    echo "<br>";
```





```
echo $Var = -01000;  
echo "<br>";  
echo $vAr = 0x1000;  
?>  
</body>  
</html>
```

Resultado na Tela do *Browser*



Observe que o formato de leitura afeta somente o modo como o numero inteiro é convertido ao ser lido, o valor que é armazenado não lembra em qual base ele foi atribuído.

NÚMEROS DE DUPLA PRECISÃO



```
<html>  
<head>  
<title>Números de Dupla Precisão</title>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
</head>  
<body>  
<?PHP  
    $var = 12.65;  
    $Var = 0.84;  
    $vAr = 2.0;  
    $vaR = -0.12345;  
?>  
</body>  
</html>
```





BOOLEANOS

São valores verdadeiros ou falsos, que são utilizados em construções de controle como parte de “teste” de uma instrução *if*. Valores booleanos podem ser combinados utilizando operadores lógicos para fazer expressões booleanas mais complexas.

O PHP fornece um par de constantes especialmente para a utilização como booleanos: *TRUE* e *FALSE*.



```
<html>
<head>
<title>Booleanos</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    if (TRUE)
    {
        echo "Verdadeiro";
    }
    else
    {
        echo "Falso";
    }
?>
</body>
</html>
```

Existem algumas regras para determinar a “verdade” de qualquer valor que já não seja do tipo booleano:

- O valor numérico, é um valor falso, se for exatamente igual a 0 é verdadeiro.
- Uma string vazia (conter 0 caracteres) ou for “0” é falsa caso contrário é verdadeira.
- Os valores tipo NULL são sempre falsos.
- O valor sendo composto (*Arrays* ou objetos) ele é falso se não tiver outro valor, caso contrário é verdadeiro.
- Os recursos válidos são verdadeiros.





```
<html>
<head>
<title>Booleanos</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $ver_numerico      = 2 + 1.165;
    $ver_string        = "Verdade";
    $ver_array[3]      = "Verdade também";
    $fal_array          = array( );
    $fal_null           = NULL;
    $fal_numerico       = 1256 - 1256;
    $fal_string         = "";
?>
</body>
</html>
```

Não é aconselhável utilizar booleanos para números de dupla precisão, pois é perigoso utilizar expressões que podem causar erros booleanos pelo seu arredondamento.

NULL

O tipo NULL possui apenas um valor possível, no caso NULL. Sua atribuição é a seguinte:



```
<html>
<head>
<title>NULL</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = NULL;
?>
</body>
</html>

<?PHP
    $var = NULL;
?>
```





O *NULL* representa a falta de um valor. No PHP uma variável que for atribuída com o valor *NULL* é quase indistinguível de uma variável que não foi atribuída. Propriedades de uma variável com atributo *NULL*:

- Ela é avaliada como *FALSE* em um contexto booleano;
- Retorna *FALSE* quando testada com *IsSet()*. Nenhum outro tipo possui esta propriedade;
- Não será impresso avisos caso variáveis sejam passadas em funções e posteriormente forem recuperadas, diferentemente de quando tentamos fazer o mesmo com uma variável que nunca foi atribuída;

No exemplo a seguir, caso a variável não tenha sido definida e dependendo das configurações de relatório de erro, pode-se imprimir um aviso.



```
<html>
<head>
<title>NULL</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    if ((funcao_teste($var))
    {
        echo "Oi!!!";
    }
?>
</body>
</html>
```

Neste próximo caso não será impresso um aviso de variável não-limitada.



```
<html>
<head>
<title>NULL</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = NULL;
    if ((funcao_teste($var))
    {
        echo "Oi!!!";
    }
?>
</body>
</html>
```





STRINGS

Strings são seqüências de caracteres alfa-numéricos.



```
<html>
<head>
<title>NULL</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var  = "Está é uma string.";
    $var2 = 'Está também é uma string';
    $var3 = ""; //Está é uma string vazia
    $var4 = "1235689";
    $var5 = "12FF35AA689";
?>
</body>
</html>
```

As *strings* podem ser incluídas entre aspas simples e duplas, salientando que a diferença será no comportamento da leitura. *Strings* entre aspas simples são tratadas literalmente, ao passo que entre aspas duplas substituem variáveis por seus valores.

STRINGS ENTRE ASPAS SIMPLES

Strings entre aspas simples lêem e armazenam seus caracteres literalmente.

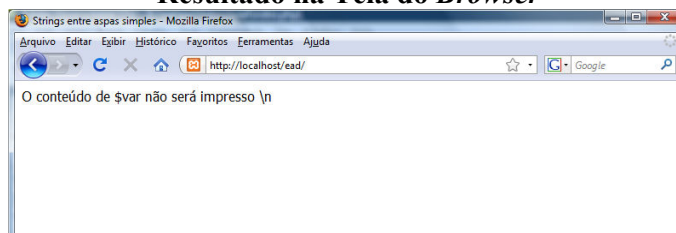


```
<html>
<head>
<title>Strings entre aspas simples</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = 'O conteúdo de $var não será impresso \n';
    echo $var;
?>
</body>
</html>
```





Resultado na Tela do *Browser*



Aspas simples obedecem à regra geral que entre elas, aspas de um tipo diferente, não a interrompem.

```
<html>
<head>
<title>Strings entre aspas simples </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = 'O conteúdo de $var não será impresso \n';
    echo $var;
?>
</body>
</html>
```



Resultado na Tela do *Browser*



Para inserirmos uma aspa simples (apóstrofe) entre aspas simples utiliza-se \.

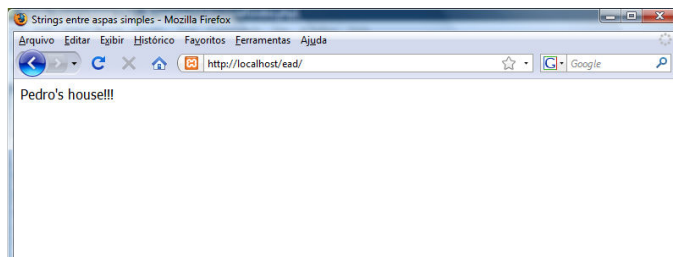
```
<html>
<head>
<title>Strings entre aspas simples </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var = 'Pedro\'s house!!!';
    echo $var;
?>
```





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

Resultado na Tela do *Browser*



STRINGS ENTRE ASPAS DUPLAS

Strings entre aspas duplas são pré-processadas pelo servidor das seguintes formas:

- Algumas seqüências de caracteres iniciadas com \ são substituídas por caracteres especiais;
- Nomes de variáveis precedidos de \$ entre aspas duplas são substituídos pelos seus valores representados como string.

As substituições de seqüência de escape são:

- \n é substituído pelo caractere de nova linha;
- \r é substituído pelo caractere CR (carriage-return [retorno de carro]);
- \t é substituído pelo caractere de tabulação;
- \\$ é substituído pelo sinal de cifrão (\$);
- \" é substituído por uma única aspa dupla (");
- \\ é substituído por uma única barra (\);

Devemos perceber que como entre aspas simples, entre aspas duplas aspas de tipo diferente podem ser utilizadas livremente.

INTERPOLAÇÃO VARIÁVEL

Sempre quando um símbolo \$ “não escapado” aparece entre aspas duplas, o PHP tenta interpretar o que vem a seguir como um nome de variável e





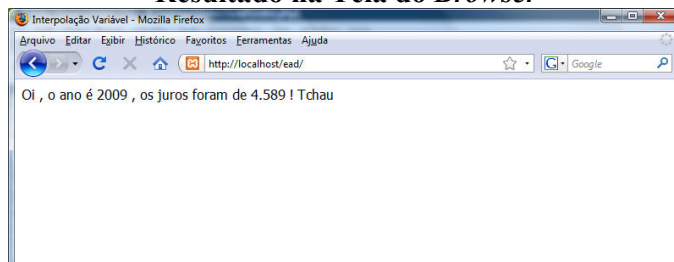
interpola o valor atual dessa variável na string. O tipo de substituição que ocorre depende de como a variável foi atribuída:

- Se a variável for uma string, esta string é interpolada (encaixada) na string entre aspas;
- Se a variável foi atribuída com não-string, o valor é convertido em string e então esse valor é interpolado;
- Caso a variável não tenha sido atribuída, o PHP não interpola uma string vazia.

```
<html>
<head>
<title>Interpolação Variável</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var  = "Oi";
    $var1 = "Tchau";
    $var2 = 2009;
    $var3 = 4.589;
    $var4 = "$var , o ano é $var2 , os juros foram de $var3 ! $var1";
    echo $var4;
?>
</body>
</html>
```



Resultado na Tela do *Browser*



Toda interpretação de strings entre aspas duplas acontece quando ela é lida, não quando é impressa. No caso acima, se dentro do mesmo script, porém em outro local onde as variáveis fossem re-atribuídas, o resultado dessa impressão, após essa re-atribuição seria diferente.



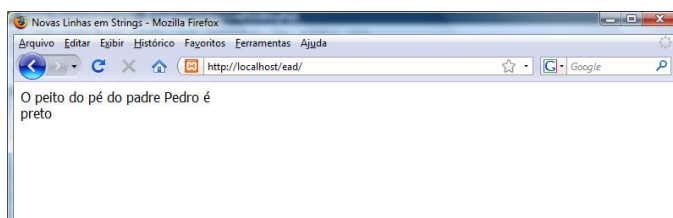


NOVAS LINHAS EM STRINGS

Embora o PHP ofereça uma sequência de escape (\n) para caracteres de nova linha, é bom saber que podemos inserir novas linhas no meio de strings, esta capacidade é conveniente para criar strings HTML, pois de qualquer jeito os browsers ignorarão a quebra de linha e assim podemos formatar nossas strings com quebras de linha para tornar mais curtas nossas linhas de código do PHP.

```
<?PHP
echo "<html><head><title>Novas Linhas em Strings</title><meta http-
equiv='Content-Type' content='text/html; charset=utf-8'>
</head><body>O peito \n do pé do \n padre Pedro é <br>
preto</body></html>";
?>
```

Resultado na Tela do *Browser*



Resultado quando é solicitado ao *Browser* para exibir o código Fonte

```
<html><head><title>Novas Linhas em Strings</title><meta http-
equiv='Content-Type' content='text/html; charset=utf-8'>
</head><body>O peito
do pé do
padre Pedro é <br> preto</body></html>
```

LIMITES

Não há limites de para comprimento de strings. Dentro do limite de memória você deve ser capaz de criar strings que são arbitrariamente longas.





ARRAYS

O array é uma maneira de se agrupar alguns valores diferentes, indexando-os por um numero que ficam entre colchetes ([0],[1]...). Podemos também armazenar tipos diferentes dentro de um mesmo array.



```
<html>
<head>
<title>Arrays</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var[0]= "Olá";
    $var[1]= 1.10;
    $var[2]= NULL;
    $var[5]= -1456;
?>
</body></html>
```

O array é um recurso muito útil e interessante em PHP. Embora pareça com arrays de outras linguagens, são implementados de maneira bem diferente.

Na maioria das linguagens, você menciona um array com uma instrução, por exemplo:

```
int var_array[10];
```

O exemplo acima equivale a um bloco de 10 variáveis contínuas de inteiros, que podem ser acessadas por índices de var_array que vai de 0 a 9.

No PHP os arrays são associativos, sendo assim você não precisa se preocupar com os valores dos índices, como:



```
<html>
<head>
<title>Arrays</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    var_array[1000000];
?>
</body>
</html>
```






Neste caso não será feita uma alocação de um milhão de slots de array, provavelmente ainda não existem slots indexados por números mais baixos, portanto, esse procedimento não consome nenhuma memória.

ARRAYS COM ÍNDICES DE STRING

Valores de string também podem ser utilizadas como índices de um array em PHP.



```
<html>
<head>
<title>Arrays</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var_array['nome'] = "Joaquim Silva";
    $var_array['ende']  = "Rua Jonas Dias";
?>
</body>
</html>
```

Além disso arrays com índices numéricos e de string podem ser usados no mesmo array sem nenhum conflito entre si.

TESTANDO OS TIPOS

Pelo fato das variáveis poderem mudar de tipo em suas re-atribuições, às vezes é necessário identificar o tipo de um valor no tempo de execução do programa.

FUNÇÃO	COMPORTAMENTO
<code>gettype(arg)</code>	Retorna uma string que representa o tipo de <i>arg</i> .
<code>is_int(arg)</code> <code>is_integer(arg)</code>	Retorna verdadeiro se <i>arg</i> for um inteiro e falso se não.





is_long(arg)	
is_double(arg) is_float(arg) is_real(arg)	Retorna verdadeiro se <i>arg</i> for um número de dupla precisão e falso se não.
is_bool(arg)	Retorna verdadeiro se <i>arg</i> for um valor booleano e falso se não.
is_null(arg)	Retorna verdadeiro se <i>arg</i> for NULL e falso se não.
is_string(arg)	Retorna verdadeiro se <i>arg</i> for uma string e falso se não.
is_array(arg)	Retorna verdadeiro se <i>arg</i> for um array e falso se não.
is_object(arg)	Retorna verdadeiro se <i>arg</i> for um objeto e falso se não.
is_resource(arg)	Retorna verdadeiro se <i>arg</i> for um recurso e falso se não.

COMPORTAMENTO DAS CONVERSÕES DE TIPOS

Regras gerais para conversão em PHP:

- Inteiro para número de dupla precisão: 4 torna-se 4.0;
- Dupla precisão para inteiro: Parte fracionária é descartada;
- Números para booleano: *FALSE* se igual a 0, caso contrário *TRUE*;
- Números para *string*: A *string* a ser criada é precisamente idêntica ao modo como o número é impresso;
- Booleano para número: 1 se *TRUE* e 0 se *FALSE*;
- Booleano para *string*: '1' se *TRUE* e *string* vazia se *FALSE*;
- *NULL* para número: 0;
- *NULL* para booleano: *FALSE*;
- *String* para número: Realiza uma conversão de tipo;
- *String* para booleano: *FALSE* se a *string* é vazia ou ' ' e *TRUE* caso contrário;
- Número ou *String* para *array*: Como criar um novo *array* com o valor atribuído ao índice 0;
- *Array* para número: Indefinido;





- *Array* para booleano: *FALSE* se o *array* não tem nenhum elemento e *TRUE* caso contrário;
- *Array* para string: *'Array'*;
- Objeto para número: Indefinido;
- Objeto para booleano: *TRUE* caso o objeto contiver qualquer variável que tiver um valor e *FALSE* caso contrário;
- Objeto para string: *'Object'*;
- Recurso para booleano: *FALSE*;
- Recurso para número: Indefinido;
- Recurso para *string*: *'Resource id #1'* – não se deve contar com isso.

Algumas conversões têm resultado indefinido, isso ocorre porque os desenvolvedores do PHP não fazem um compromisso quanto ao comportamento que será obtido.

CONVERSÕES EXPLÍCITAS

Existem três maneiras a manipulação de tipos e a chamada de variáveis `settype()`:

- As funções `intval()`, `doubleval()` e `strval()`;
- Qualquer expressão pode ser precedida por uma conversão de tipo (nome do tipo entre parênteses), que converte o resultado para o tipo desejado;
- Qualquer variável pode ser fornecida como um primeiro argumento para o `settype()`, que mudará o tipo desta variável para o tipo identificado no segundo argumento da *string*.



```
<html>
<head>
<title>Conversões Explícitas</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $var  = intval (strval (doubleval("Olá!!!")));
    $var2 = "1234";
    settype($var2, "double");
    settype($var2, "string");
    settype($var2, "int");
?>
</body>
```





</html>

Alguns nomes alternativos podem ser usados em conversões (mas não em `settype()`):

- `int` ao invés de *integer*;
- *float* ou *real* ao invés de *double*;
- *bool* ao invés de *boolean*.

```
<html>
<head>
<title>Conversões Explícitas</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $tipo[0]      = 123;
    $tipo[1]      = 3.65465;
    $tipo[2]      = "Variável string";
    $tipo[3]      = "48.5454 e string";
    $tipo[4]      = array(90,80,70);

    echo "<table border=1 width='70%'>\n\t<tr>\n";
    echo "\t\t<td>ORIGINAL</td>\n";
    echo "\t\t<td>INT</td>\n";
    echo "\t\t<td>DOUBLE</td>\n";
    echo "\t\t<td>STRING</td>\n";
    echo "\t\t<td>ARRAY</td>\n\t</tr>";

    for ($index=0; $index<5; $index++)
    {
        echo "\n\t<tr>\n\t\t<td>$tipo[$index]</td>\n";
        $conversao = (int) $tipo[$index];
        echo "\t\t<td>$conversao</td>\n";
        $conversao = (double) $tipo[$index];
        echo "\t\t<td>$conversao</td>\n";
        $conversao = (string) $tipo[$index];
        echo "\t\t<td>$conversao</td>\n";
        $conversao = (array) $tipo[$index];
        echo "\t\t<td>$conversao</td>\n\t<tr>";
    }
    echo "</table>";
?>
</body>
</html>
```





CONVERSÕES ÚTEIS

- `ord()` – Retorna o valor ASCII do caractere;
- `ceil()` – Arredonda frações para cima;



```
<html>
<head>
<title>Conversões Úteis</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    echo ceil(5.6); // Saída é 6
    echo ceil(-5.6); // Saída é -5
?>
</body>
</html>
```

- `floor()` – Arredonda frações para baixo;



```
<html>
<head>
<title>Conversões Úteis</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    echo floor(5.6); // Saída é 5
    echo floor(-5.6); // Saída é -6
?>
</body>
</html>
```

- `round()` – Arredonda um número;



```
<html>
<head>
<title>Conversões Úteis</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
```





```
echo round(5.6); // Saída é 6
echo round (-5.6); // Saída é -6
echo round (-5.654654, 4); // Saída é -5,6546
?>
</body>
</html>
```

- chr() – Retorna uma string de um único caractere contendo o caracter especificado pelo ASCII;
- implode() – Junta elementos de uma matriz em uma string;



```
<html>
<head>
<title>Conversões Úteis</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $array = array('nome', 'email', 'fone');
    $var = implode(",", $array);
    echo $var; // Saída é lastname,email,phone
?>
</body>
</html>
```

- explode() - Divide uma string em strings;



```
<html>
<head>
<title>Conversões Úteis</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head><body>
<?PHP
    $pizza = "piece1 piece2 piece3 piece4 piece5 piece6";
    $var = explode(" ", $pizza);
    echo $var[0]; // Saída é piece1
    echo $var[1]; // Saída é piece2
?></body></html>
```





7

OPERADORES LÓGICOS

OPERADOR	COMPORTAMENTO
and	É verdadeiro se e somente se ambos os argumentos forem verdadeiros.
or	É verdadeiro se um ou ambos os argumentos forem verdadeiros.
!	É verdadeiro se um único argumento à direita for falso; e falso se seu argumento for verdadeiro.
xor	É verdadeiro se qualquer um, mas não ambos os argumentos forem verdadeiros.
&&	Idêntico a and, mas associa os argumentos mais fortemente
 	Idêntico a or, mas associa os argumentos mais fortemente

PRECEDÊNCIA DOS OPERADORES LÓGICOS

Assim como qualquer operador, alguns operadores lógicos têm precedência mais alta que outros, embora a precedência possa sempre ser anulada agrupando subexpressões utilizando parênteses. A precedência dos operadores lógicos de forma decrescente é: !, &&, ||, and, xor, or.





8

OPERADOR	NOME	COMPORTAMENTO
==	Igual	Verdadeiro se os argumentos forem iguais entre si.
!=	Não Igual	Falso se os argumentos forem iguais entre si.
<	Menor que	Verdadeiro se o argumento da esquerda for menor que o argumento da direita.
>	Maior que	Verdadeiro se o argumento da esquerda for maior que o argumento da direita.
<=	Menor que ou igual a	Verdadeiro se o argumento da esquerda for menor ou igual ao que o argumento da direita.
>=	Maior que ou igual a	Verdadeiro se o argumento da esquerda for maior ou igual ao que o argumento da direita.
===	Idêntico	Verdadeiro se os argumentos forem iguais entre si e do mesmo tipo.

Um equívoco muito comum é confundir o operador de atribuição (=) com o operador de operação (==). A atribuição `if ($var1 = $var2)` vai configurar, talvez inesperadamente, a variável `$var1` como sendo idêntica a `$var2`.

COMPARAÇÃO DE STRINGS

As comparações de string fazem distinção entre maiúsculo e minúsculo.





9

ESTRUTURAS DE CONTROLE

DESVIANDO

As principais estruturas para desvio em PHP são if e switch. Switch é uma alternativa útil para certas situações em que você precisa de vários desvios possíveis com base em um único valor.

IF/ELSE

```
<html>
<head>
<title>if/else</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    if (comparação)
    {
        instrução;
    }
    #-----
    if (comparação1)
    {
        instrução;
    }
    else
    {
        instrução;
    }
    #-----
    if (comparação2)
    {
        instrução;
    }
    elseif (comparação3)
    {
        instrução;
    }
?>
</body>
</html>
```





Quando uma instrução if é processada, a expressão é avaliada e o resultado é interpretado como um valor booleano.

SWITCH

```
<html>
<head>
<title>Switch</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
$dia = date("d");
switch($dia)
{
    case 01:
        echo "Dia primeiro";
        break;
    case 02:
        echo "Dia dois";
        break;
    case 03:
        echo "Dia três";
        break;
    case 04:
        echo "Dia quatro";
        break;
    case 05:
        echo "Dia cinco";
        break;
}
?>
</body>
</html>
```



Neste caso utilizamos uma variável para ser avaliada, mas podemos utilizar uma expressão desde que seja avaliada como um valor simples, isto é, um inteiro, número de dupla precisão ou string.

Importante dizer que caso se esqueça de colocar a instrução break todos os casos executarão depois de um caso ser correspondido.





LOOPS


Loops limitados x Loops não-limitados

Um loop limitado é um loop que executará um número fixo de vezes. Ao examinar o código, pode-se identificar quantas vezes este loop se repetirá, e a linguagem garantirá que não sejam feitos loops além do necessário.

Um loop não-limitado é aquele que se repete até que alguma condição se torne verdadeira ou falsa, e essa condição depende da ação do código dentro do loop.

Diferentemente de outras linguagens, o PHP não tem nenhuma construção específica para loops limitados – while, do while, for são construções não-limitadas, mas um loop não-limitado pode fazer as mesmas coisas que um loop limitado.

WHILE



```
<html>
<head>
<title>While</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    while(condicao)
    {
        instrucao;
    }
?>
</body>
</html>
```

O loop while avalia a expressão condicao como um booleano, se for verdadeiro executa a instrução e então volta a avaliar a condicao. Se a condicao for falsa, o loop é encerrado.

O próximo exemplo mostra uma forma de limitar um while.





```
<html>
<head>
<title>While</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $cont = 1;
    while ($cont <= 100)
    {
        echo $cont."<br>";
        $cont++;
    }
?>
</body>
</html>
```

DO-WHILE

A construção do do-while é semelhante ao while, exceto que o teste acontece no final do loop.



```
<html>
<head>
<title>do while</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    do
    {
        instrucao
    }
    while (expressao)
?>
</body>
</html>
```

Assim a instrução é executada uma vez e então a expressao é avaliada. Caso seja verdadeira e instrucao será repetida até que a expressao se torne falsa.



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





```
<body>
<?PHP
    $cont = 100;
    do
    {
        echo $cont."<br>";
    }
    while ($cont <=10)
?>
</body>
</html>
```

No caso acima será escrito na tela o valor que \$cont só uma vez.

FOR



```
<html>
<head>
<title>For</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    for (expressão-inicial; verificação-de-término; expressão-de-final-do-
loop)
    {
        instrução;
    }
?>
</body>
</html>
```

Ao iniciar, primeiro a expressão-inicial é avaliada somente uma vez, então a verificação-do-término é avaliada, e se for falsa, a instrução for é concluída. Se for verdadeira, a instrução é executada. Por fim, a expressão-de-final-do-loop é executada e inicia o ciclo outra vez com a verificação-do-término.



```
<html>
<head>
<title>do while</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    for ($cont=1;$cont<=20;$cont++)
    {
```





```
        echo $cont."<br>";  
    }  
?>  
</body>  
</html>
```





10

UTILIZANDO FUNÇÕES

SINTAXE BÁSICA PARA UTILIZAR UMA FUNÇÃO



```
<html>
<head>
<title>Função</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    nome_da_função(expressão1, expressão2, ..., expressão)
?>
</body>
</html>
```

Essa sintaxe inclui o nome da função seguido de uma lista entre parênteses de expressões de entrada, que são chamadas argumentos para a função. As funções podem ser chamadas com zero ou mais argumentos, dependendo de sua definição.

No caso do PHP quando encontra uma função, primeiramente ele avalia cada expressão de argumento e depois utiliza esses valores como entrada na função. Depois que a função é executada, o valor de retorno, caso haja algum, é o resultado de toda a expressão da função.

Exemplo de funções pré-definidas:

`sqrt(9)` // Raiz quadrada

`rand(10, 10 + 10)` // Número aleatório entre 10 e 20

`strlen("String com 24 caracteres")` // Retorna o número 22

`pi()` // Retorna o valor aproximado de pi

Atentando para que essas funções são chamadas com 1,2 e nenhum argumento.






Valores de retorno x efeitos colaterais


Existem duas razões para você incluir uma chamada de função em PHP, um seria para o valor de retorno e o outro seria para os efeitos colaterais.

Uma forma de se obter o valor de retorno é atribuí-lo a uma variável:



```
<html>
<head>
<title>Função</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $val_pi = pi();
?>
</body>
</html>
```

Ou embuti-los em expressões mais complexas, como:



```
<html>
<head>
<title>Função</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    $expre = sqrt($var) * sqrt($var2);
?>
</body>
</html>
```

Funções também são utilizadas para uma ampla variedade de efeitos colaterais, inclusive gravação em arquivos, manipulação de banco de dados e impressão para a janela do navegador. É muito comum ter uma função para a realização de efeitos colaterais que retornam um valor que mostram se a função foi ou não bem sucedida.





DEFININDO AS SUAS PRÓPRIAS FUNÇÕES

Você pode produzir sites Web úteis e interessantes utilizando simplesmente as construções básicas e o grande número de funções pré-definidas da linguagem. Entretanto, se você achar que os arquivos de código estão ficando maiores, mais difíceis de entender e mais difíceis de gerenciar, isso pode ser um indício que você deve começar a embrulhar alguns dos códigos em funções.

Sintaxe de definição de função



```
<html>
<head>
<title>Função</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    function nome($argumento1, $argumento2, ...)
    {
        instrução1;
        instrução2;
        ...
    }
?>
</body>
</html>
```

A definição das funções é dividida em 4 partes:

- A palavra especial function;
- O nome que deseja dar a função;
- A lista de parâmetros da função – variáveis separadas por vírgula;
- O corpo da função;





11

INCLUDE E REQUIRE

Esses comandos importam o conteúdo de um outro arquivo para o que está sendo executado. Isso se torna muito útil por possibilitar a utilização de, por exemplo, funções construídas por você, em várias páginas sem precisar digitá-las novamente no início de cada página que as utilizará. Outra vantagem é que para a atualização dessa função você necessitará alterar apenas um arquivo.



```
<html>
<head>
<title>Função</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<?PHP
    include "func_data.inc";
    require "func_data2.inc";
?>
</body>
</html>
```





12

PASSANDO INFORMAÇÕES ENTRE PÁGINAS

Com relação à maneira como a Web funciona é sempre bom lembrarmos que o protocolo HTTP é um protocolo sem informações de estado, isto é, que a cada solicitação HTTP, que na maioria das vezes se traduz em cada tela HTML solicitada e fornecida, esta é independente de todas as outras, ou seja, uma não leva nenhuma informação consistente sobre a identidade do solicitante e não tem nenhuma memória. Cada solicitação gera um processo discreto, que se ocupa somente de sua tarefa “humilde” de atender e fornecer a somente um arquivo solitário que depois é automaticamente eliminada.

Mesmo se pensar em um site em um só sentido de navegação (a Página 1 conduz a Página 2, que conduz a Página 3 e assim por diante) o protocolo HTTP nunca saberá ou se importará que alguém navegando na Página 2 tenha vindo da Página 1. Você não pode configurar uma variável com um valor na Página 1 e esperar que ele seja importado para a Página 2 pelas exigências da própria HTML.

O que se pode fazer é utilizar a HTML para exibir um formulário e alguém pode inserir algumas informações utilizando-o, mas a menos que você empregue algum meio extra para passar as informações para a outra página ou para o programa, a variável simplesmente desaparecerá no espaço logo que você se mover para a outra página.

Sendo assim é aí que entra uma tecnologia de processamento de formulário como o PHP. O PHP vai capturar a variável passada de uma página para a próxima e vai torná-la disponível para utilização posterior. O PHP é realmente muito bom nesse tipo de função de passagem de dados, o que agiliza e facilita o seu emprego em uma ampla variedade de tarefas relacionadas com sites *Web*.

Os formulários em HTML são especialmente úteis para passar alguns valores de uma determinada página para outra. Existem outras maneiras mais persistentes de manter o estado em muitas *pageviews*, ou visualizações de páginas, como *cookies* e sessões. No entanto neste momento focalizaremos as técnicas básicas de passagem de informações entre páginas *Web*, que utilizam os métodos *GET* e *POST* em HTTP para criar páginas geradas dinamicamente e tratar dados de formulários.





ARGUMENTOS GET

O método *GET* passa argumentos de uma página para a próxima como parte de consulta *Uniform Resource Indicator*, também chamada *Uniform Resource Locator* ou URL. Quando utilizado para tratamento de formulário, *GET* acrescenta o nome indicado de variáveis e valores ao URL designado no atributo *ACTION* com um separador de ponto de interrogação e envia tudo para o agente de processamento, que é, nesse caso, um servidor *Web*.

Abaixo um formulário HTML de exemplo que utiliza o método *GET*, salve o arquivo com o nome de `get.php`.



```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de Método GET</title>
</head>
<body>
<form name="form1" method="GET" action="http://localhost/get2.php">
  <select name="Cargo">
    <option value="Gerente de TI">Gerente de TI</option>
    <option value="Gerente de Redes">Gerente de Redes</option>
    <option value="Gerente de Suporte">Gerente de Suporte</option>
  </select>
  <input type="submit" value="Envia">
</form>
</body>
</html>
```

Quando o usuário faz uma seleção e clica no botão Submit, o navegador agrupa esses elementos nesta ordem, sem espaços entre os elementos:

- O URL entre aspas depois da palavra *ACTION* (<http://localhost/get2.php>)
- Um ponto de interrogação (?) que denota que os caracteres a seguir constituem uma *string GET*.
- Uma variável *NAME*, um sinal de igual e o *VALUE* correspondente.
- Um “e” comercial (&) e o próximo par *NAME-VALUE* (*Submit=Select*). Isso pode ser repetido tantas vezes quanto o limite de comprimento de *string* de consulta do servidor permitir.

Assim o navegador constrói a *string* de URL:





`http://localhost/get2.php?cargo=Gerente de TI`

Em seguida, encaminha esse URL para o seu próprio espaço de endereçamento como uma nova solicitação. O script do PHP para o qual o formulário precedente é enviado (`get2.php`) irá capturar as variáveis *GET* do final da *string* de solicitação e fará algo útil com elas. Nesse caso, conectar um dos dois valores em uma *string* de texto.



```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de Método GET</title>
</head>
<body>
<?PHP
echo $_GET['Cargo'];

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

Existem algumas desvantagens que devemos salientar sobre o método *GET*:

- Não é adequado para *logins* porque o nome de usuário e a senha são completamente visíveis na tela, bem como potencialmente armazenados na memória do navegador cliente como uma página visitada.
- Cada envio de *GET* é registrado no *log* de servidor *Web*, incluído o conjunto de dados.
- Como o método *GET* atribui dados a uma variável ambiente de servidor, o comprimento do URL é limitado. Você poderá ter visto algo que lembra URLs muito longos utilizando *GET*, mas por certo não gostaria de tentar passar um texto de 300 palavras formatado em HTML usando esse método.



A especificação original de HTML defendia que as *strings* de consulta fossem limitadas a 255 caracteres. Embora essa estrutura tenha sido relaxada mais tarde para um mero “encorajamento” de um limite de 255 caracteres, utilizar uma *string* mais longa é procurar problemas.





Embora o método *GET* real de tratamento de formulário seja depreciado, o estilo de URL associado a ele revela-se muito útil para a navegação de site. Isso é especialmente verdadeiro nos sites gerados dinamicamente, como aqueles freqüentemente construídos com o PHP, uma vez que o estilo de variável acrescentado do URL funciona particularmente bem com um sistema de desenvolvimento de conteúdo baseado em modelos.



```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Método GET</title>
</head>
<body>
<table width="100%" border="1">
<tr>
<td width="30%"><a href="get3.php?nome=pagina1">Página 1</a></td>
<td rowspan="3"><div align="center">
<?PHP
include ("$_GET['nome'].inc");
?>
</div></td>
</tr>
<tr>
<td width="30%"><a href="get3.php?nome=pagina2">Página 2</a></td>
</tr>
<tr>
<td width="30%"><a href="get3.php?nome=pagina3">Página 3</a></td>
</tr>
</table>
</body>
</html>
```

ARGUMENTOS POST

O método *POST* é o método de envio de formulários mais utilizado hoje, o conjunto de dados está incluído no corpo do formulário quando ele é encaminhado para o agente de processamento, no caso o PHP. Nenhuma alteração visível ao URL resultará de acordo com os diferentes dados enviados.

O método POST apresenta as seguintes vantagens:

- É mais seguro que *GET* porque informações inseridas pelo usuário nunca são visíveis na *string* de consulta URL, nos *logs* do servidor,





ou na tela, se forem tomadas certas precauções como sempre utilizar o tipo de entrada HTML password para senhas.

- Há um limite muito maior na quantidade de dados que pode ser passada (cerca de 2 kilobytes em vez de aproximadamente duzentos caracteres).
- Os resultados em um determinado momento não podem ser marcados.
- Esse método pode ser incompatível com certas configurações de firewall, o que elimina os dados de formulários como uma medida de segurança.



```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de Método POST</title>
</head>
<body>
<form name="form1" method="POST" action="http://localhost/get2.php">
  <select name="Cargo">
    <option value="Gerente de TI">Gerente de TI</option>
    <option value="Gerente de Redes">Gerente de Redes</option>
    <option value="Gerente de Suporte">Gerente de Suporte</option>
  </select>
  <input type="submit" value="Envia">
</form>
</body>
</html>
```



```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Exemplo de Método POST</title>
</head>
<body>
<?PHP
echo $_POST['Cargo'];

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





13

SESSÃO EM PHP

Inicialmente é necessário entender o que se quer dizer quando falamos em sessão. De forma coloquial diz-se que uma sessão de navegação é o período de tempo que uma determinada pessoa, em um determinado computador, vê um determinado número de páginas em seu browser até o momento que ela o fecha. Mas para que usáramos isso? Bom, imagine que você possui um hotel com um site na internet, e para deseja fazer uma análise aprofundada do objetivo de uma sessão de cinco páginas de duração, no meio de uma sessão real de um usuário que começou com um portal de turismo e terminou reservando suas férias com um concorrente.

COMO É O FUNCIONAMENTO DAS SESSÕES EM PHP

Para entrarmos mais na sintaxe do PHP chamaremos as sessões de *sessions*.

Assim, uma *Session* é uma variável global a qual é eliminada quando o usuário fecha o *browser*.

Por exemplo, quando fazemos uma página e existe a necessidade de setar um número qualquer para o usuário e esse número só será necessário em outra página muito a frente. Ao invés de ficarmos carregando `pagina.php?num=1`, criaremos uma *session* e somente nos preocuparemos com o número setado no final do *script*.

Sendo assim vamos para o funcionamento prático:

SINTAXE E EXPLICAÇÕES:



<?PHP

//Inicia a sessão de nome "log"
`session_start("log");`

//Verifica se a variável de sessão "codUusario" não esta registrada(se já foi





```
criada)
if(!session_is_registered("codUsuario"))
{

    //Registra(cria) as variaveis de sessão
    session_register("codUsuario","nomeUsuario");

}
//Verifica se a variável de sessão "codUsuario" está vazia
if(empty($_SESSION["codUsuario"]))
{

    //Atribui os dados as variaveis de sessão
    $_SESSION["codUsuario"]=15;
    $_SESSION["nomeUsuario"]="João da Silva";

}

echo "Bom dia Sr(a). ".$_SESSION["nomeUsuario"]." !";

//Exclui a variavel de sessão "codUsuario"
session_unregister("codUsuario");

//Destroi a sessão
session_destroy();
?>
```

Vale ressaltar que caso não tivesse utilizado as funções `session_unregister` e `session_destroy()`, estes valores estariam armazenados pelos mecanismos do *browser* até que o mesmo fosse fechado pelo usuário.



Um exemplo claro de funcionalidade para as *sessions* é quando se está em algum site que necessita de uma autenticação, ou seja, que o usuário entre com um *login* e uma senha. À medida que o usuário navega pela área que lhe é restringida ao seu usuário, a cada clique é feita uma validação para averiguar se a autenticação foi realizada e se o usuário tem direito a acessar aquela página. Por isso, que quando se está em um computador que foi utilizado por outra pessoa e esta acessou o site de um banco e posteriormente, ou clicou em sair (o que seria o mesmo que “destruir a sessão”), ou fechou o *browser*, mesmo quando se tenta acessar os endereços das áreas restritas que ficam no histórico é pedido uma nova autenticação.





14

COOKIE EM PHP

Um *cookie* é um pequeno fragmento de informação que é retido, ou armazenado, na máquina cliente, quer na memória da aplicação navegadora, quer em um pequeno arquivo gravado no disco rígido do usuário.

Este arquivo contém um par nome/valor – configurar um *cookie* significa associar um valor com um nome e armazenar esse par no lado do cliente. Obter ou ler um *cookie* significa utilizar o nome para recuperar o valor.

COMO É O FUNCIONAMENTO DE UM COOKIE EM PHP

Para tratar o cookie com uma linguagem mais direta pode-se dizer que o cookie serve para que se possa gravar uma informação na máquina cliente, ou em outro domínio, e que se possa recuperar esta informação posteriormente.

Um exemplo clássico desta utilização é para informar quantas vezes o usuário já acessou uma página ou para auxiliar em uma área restrita quanto utilizamos uma sessão.

Sendo assim vamos para o funcionamento prático:

SINTAXE E EXPLICAÇÕES:

```
setcookie(name=nome;value=valor; expires=data; path=caminho;  
domain=dominio; secure)
```



Onde:

name=nome - é a única informação obrigatória pois aqui definimos o nome do cookie.

value=valor - é o valor que vai ser atribuído a ele.

expires=data - Esse atributo especifica a data de validade, ou seja quando a data é alcançada o mesmo é excluído da máquina do usuário. Se ele não for definido o cookie será excluído normalmente quando o usuário fechar o browser (FORMATO: *DD-Mon-YYYY HH:MM:SS*).





path = caminho - Especifica o diretório em que o cookie será disponibilizado no servidor, quando ele não for informado o cookie será enviado para a pasta padrão do cliente.

domain = domínio - Especifica para qual domínio o cookie será enviado, normalmente ele é enviado para o computador que o gerou mas podemos determinar um outro domínio para o qual ele vai ser enviado.

secure - Os cookies marcados com este atributo somente serão enviados se a comunicação entre servidor e cliente for uma conexão segura ou seja apenas para servidores HTTPS (FORMATO: TRUE ou FALSE).

Agora um exemplo prático utilizando o cookie. A idéia é criar um cookie que guardará quantas vezes determinado “computador” acessou nosso site.

```
<?PHP
if ($_COOKIE["contador"])
{
    $N_ACESS=$_COOKIE["contador"]+1;
    setcookie("contador",$N_ACESS, time()+3600000);
}
else
{
    setcookie("contador",1, time()+3600000);
    $N_ACESS = 1;
}
?>
<html>
<head>
<title>Cookie</title>
</head>
<body>
<?PHP
echo $_COOKIE["contador"];
?>
</body>
</html>
```



Percebam que ao iniciar o cookie utilizei somente 3 parâmetros:

- o nome do cookie;
- o valor que ele conterá;
- e o tempo de “vida” deste cookie;

Para ser mais específico quanto a configuração do tempo de “vida” irei explicar o que a função time() faz:

“Retorna a hora atual medida no número de segundos desde a Era Unix (January 1 1970 00:00:00 GMT)” (PHP.net, 2008).

Sendo assim estou utilizando o tempo de vida do cookie para 1000 horas, ou 41,6 dias, (3600000 segundos) a partir de hoje.





15

PHP COM MYSQL

Para iniciar o tratamento do PHP com o banco de dados MYSQL é sempre bom lembrar que a interação bem como a integração entre os dois acontece de forma estável e simples.

Para interagir com uma base de dados SQL existem três comandos básicos que devem ser utilizados: um que faz a conexão com o servidor de banco de dados, um que seleciona a base de dados a ser utilizada e um terceiro que executa uma “*query*” SQL. Para a execução deste terceiro é importante ressaltar que as funções em PHP não se limitam a somente uma e sim a várias, onde será tratado das três que se pode dizer serem as mais utilizadas.

FUNÇÃO DE CONEXÃO COM O SERVIDOR MYSQL

A conexão com o servidor de banco de dados MYSQL em PHP é feita através do comando **mysql_connect**, que tem a seguinte sintaxe:

mysql_connect(host,login,senha);

Os parâmetros são bastante simples: o endereço do servidor(host), o nome do usuário do banco de dados (login) e a senha do banco de dados para a conexão.

A função retorna um valor inteiro, que é o identificador da conexão estabelecida e deverá ser armazenado numa variável (\$variavel) para ser utilizado depois.

O exemplo abaixo, tem-se como servidor de banco de dados a mesma máquina que roda o servidor http, como login o usuário “root” e senha “12345”:



<?PHP

\$con = mysql_connect(“localhost”, “root”, “12345”);

?>





Sendo bem sucedida a conexão (existir um servidor no endereço especificado que possua o usuário com a senha fornecida), o identificador da conexão fica armazenado na variável **\$con**.

FUNÇÃO DE SELEÇÃO DE BANCO DE DADOS COM O SERVIDOR MYSQL

Após a conexão é necessário a seleção de um banco de dados no servidor com o qual desejamos trabalhar. A função utilizada para isto é a **mysql_select_db**, que possui a seguinte sintaxe:

mysql_select_db(nome_base, conexão estabelecida com o banco);

Esta função retornará 0 se o comando falhar, e 1 em caso de sucesso. Será selecionado como primeiro parâmetro o nome da base de dados e logo em seguida o identificador da conexão que faz referência a esta base. Caso seja omitido o último parâmetro, o mecanismo de *script* tentará utilizar a última conexão estabelecida. É muito importante explicitar esse valor, para facilitar a legibilidade do código, além do que é factível a possibilidade de se estar trabalhando com mais de uma conexão a banco de dados distintos e até mesmo em servidores diferentes.

O exemplo abaixo seleciona uma base de dados chamada **sabe**:



<?PHP

\$sele_BD=mysql_select_db("sabe", \$con);

?>

Após a execução desse comando qualquer consulta executada para aquela conexão utilizará a base de dados selecionada.

EXECUTANDO "QUERIES" EM PHP

Depois dos procedimentos acima terem sido realizados e nenhum erro tenha ocorrido às interações com o servidor MYSQL pode ser feita através de





consultas escritas em SQL (Structured Query Language), com o comando `mysql_query`, que utiliza a seguinte sintaxe:

```
mysql_query(consulta, conexão estabelecida com o banco);
```

O retorno desta função será 0 caso ocorra alguma falha ou 1 caso a “querie” tenha sido executada com sucesso. Diz-se sucesso caso a consulta esteja sintaticamente correta e foi executada no servidor. Nenhuma informação sobre o resultado é retornada deste comando, ou até mesmo se o resultado é o esperado. Sendo a consulta um comando `SELECT`, o valor de retorno é um valor interno que identifica o resultado, que poderá ser tratado com a função `mysql_fetch_object()`, `mysql_fetch_array()`, `mysql_fetch_row()`, e outras. A string query não deve conter ponto-e-vírgula no final do comando e o identificador da conexão é opcional. Será utilizada a criação de uma tabela como exemplo:



<?PHP

```
mysql_query(“CREATE TABLE exemplo (codigo INT AUTO_INCREMENT  
PRIMARY KEY, nome CHAR(40), email CHAR(50))”, $con);
```

?>

Abaixo o exemplo completo de como seriam os passos para se adicionar dados em uma tabela criada, executando “queries” SQL numa base de dados MYSQL:



<?PHP

```
$con = mysql_connect(“localhost”, “root”, “12345”);  
$sele_BD=mysql_select_db(“sabe”, $con);  
mysql_query(“CREATE TABLE exemplo (codigo INT AUTO_INCREMENT  
PRIMARY KEY, nome CHAR(40), email CHAR(50))”, $con);  
mysql_query(“INSERT INTO exemplo (nome,email) VALUES  
(“Alan”, “alan@unis.edu.br”)”, $con);  
mysql_query(“INSERT INTO exemplo (nome,email) VALUES  
(“Tulio”, “tulio@unis.edu.br”)”, $con);  
mysql_query(“INSERT INTO exemplo (nome,email) VALUES  
(“Suporte”, “suporte@sabe.br”)”, $con);  
mysql_query(“INSERT INTO exemplo (nome,email) VALUES  
(“Junior”, “junior@unis.edu.br”)”, $con);  
mysql_query(“INSERT INTO exemplo (nome,email) VALUES  
(“Mara”, “mara@unis.edu.br”)”, $con);
```

?>






RESULTADOS DA QUERY SELECT

Ao executar uma *query* SQL SELECT através do comando `mysql_query`, o identificador do resultado deve ser armazenado numa variável que pode ser tratada de diversas formas. Três maneiras interessantes de fazê-lo usam o comando `mysql_fetch_row`, o comando `mysql_fetch_object` e o comando `mysql_fetch_array`.

A diferença entre as três principais funções de busca é pequena.


Sendo assim a função `mysql_fetch_row` pode ser utilizada conforme o exemplo:



```
<?PHP
$result = mysql_query("SELECT * from exemplo", $con);
While ($campo=mysql_fetch_row($result))
{
    echo "$campo[0] --- $campo[1] </ br>";
}
?>
```

O código acima irá imprimir todos os registros da tabela exemplo, cada um em uma linha, sendo de cada campo da tabela será separado pela *string* ---.

A função `mysql_fetch_object` realiza a mesma tarefa, exceto pelo fato da linha ser retornada como um objeto em vez de um *array*. Obviamente isso é útil para os programadores que utilizam o PHP em notação orientada a objetos. Veja o exemplo:



```
<?PHP
$result = mysql_query("SELECT * from exemplo", $con);
While ($campo=mysql_fetch_object($result))
{
    echo $campo->nome. " --- ";
    echo $campo->email. " </br> ";
}
?>
```

O código acima irá imprimir todos os registros da tabela exemplo, cada um em uma linha, sendo de cada campo da tabela será separado pela *string* ---.

A função de busca `mysql_fetch_array` oferece a escolha de resultados como um *array* associativo, isto significa que você referencia as saídas pelo nome do campo do banco de dados em vez do número. Veja o exemplo:





<?PHP

```
$result = mysql_query("SELECT * from exemplo", $con);
```

```
While ($campo=mysql_fetch_array($result))
```



```
{  
    echo $campo['nome']." --- ";  
    echo $campo['email']." < /br> ";  
}
```

?>

O resultado será o mesmo apresentado nas funções anteriores.





Referências Bibliográficas

BEIGHLEY, Lynn. **Use a cabeça! SQL**. ALTA BOOKS, 2008

CARVALHO, Alan. **Criando Sites Profissionais HTML 4.1 e CSS 2.1: Manual Completo**. BOOK EXPRESS.

DALL'OGGIO, Pablo. **PHP Programando com Orientação a Objetos**. NOVATEC, 2009

MCLAUGHLIN, Brett; POLLICE, Gary; WEST, David. **Use a Cabeça! Análisis e Projeto Orientado ao Objeto**. ALTA BOOKS, 2007

VENETIANER, Tomas - **Html : Desmistificando a Linguagem da Internet**. MAKRON BOOKS, 1996.

