CURSO DE LINGUAGEM PHP

Autor: Maurício Vivas de Souza Barreto
mauricio@cipsga.org.br

Abril de 2000

Maurício Vivas de Souza Barreto

mauricio@cipsga.org.br

vivas@usa.net

Abril de 2000

Projeto Supervisionado de Final de Curso

Este apostila de PHP e fruto do Projeto Supervisionado de Final de Curso de Maurício Vivas de Souza Barreto, tendo o mesmo sido submetido a uma banca examinadora composta pelo Professor Giovanny Lucero, Professora Ana Rosimeri e Professor Leonardo Nogueira Matos, da Universidade Federal de Sergipe, Centro de Ciências Exatas e Tecnologia do Departamento de Estatistica e Informática.

Indice

1. Introdução

O que é PHP?

PHP é uma linguagem que permite criar sites WEB dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links. A diferença de PHP com relação a linguagens semelhantes a Javascript é que o código PHP é executado no servidor, sendo enviado para o cliente apenas html puro. Desta maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente. Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.

O que diferencia PHP de um script CGI escrito em C ou Perl é que o código PHP fica embutido no próprio HTML, enquanto no outro caso é necessário que o script CGI gere todo o código HTML, ou leia de um outro arquivo.

O que pode ser feito com PHP?

Basicamente, qualquer coisa que pode ser feita por algum programa CGI pode ser feita também com PHP, como coletar dados de um formulário, gerar páginas dinamicamente ou enviar e receber *cookies*.

PHP também tem como uma das características mais importantes o suporte a um grande número de bancos de dados, como dBase, Interbase, mSQL, mySQL, Oracle, Sybase, PostgreSQL e vários outros. Construir uma página baseada em um banco de dados torna-se uma tarefa extremamente simples com PHP.

Além disso, PHP tem suporte a outros serviços através de protocolos como IMAP, SNMP, NNTP, POP3 e, logicamente, HTTP. Ainda é possível abrir *sockets* e interagir com outros protocolos.

Como surgiu a linguagem PHP?

A linguagem PHP foi concebida durante o outono de 1994 por **Rasmus Lerdorf**. As primeiras versões não foram disponibilizadas, tendo sido utilizadas em sua *home-page* apenas para que ele pudesse ter informações sobre as visitas que estavam sendo feitas. A primeira versão utilizada por outras pessoas foi disponibilizada em 1995, e ficou conhecida como "**P**ersonal **H**ome **P**age Tools" (ferramentas para página pessoal). Era composta por um sistema bastante simples que interpretava algumas *macros* e alguns utilitários que rodavam "por trás" das *home-pages*: um livro de visitas, um contador e algumas outras coisas.

Em meados de 1995 o interpretador foi reescrito, e ganhou o nome de **PHP/FI**, o "FI" veio de um outro pacote escrito por Rasmus que interpretava dados de formulários HTML (**F**orm **I**nterpreter). Ele combinou os scripts do pacote *Personal Home Page Tools* com o FI e adicionou suporte a mSQL, nascendo assim o PHP/FI, que cresceu bastante, e as pessoas passaram a contribuir com o projeto.

Estima-se que em 1996 PHP/FI estava sendo usado por cerca de 15.000 *sites* pelo mundo, e em meados de 1997 esse número subiu para mais de 50.000. Nessa época houve uma mudança no desenvolvimento do PHP. Ele

deixou de ser um projeto de Rasmus com contribuições de outras pessoas para ter uma equipe de desenvolvimento mais organizada. O interpretador foi reescrito por **Zeev Suraski** e **Andi Gutmans**, e esse novo interpretador foi a base para a versão 3.

Atualmente o uso do PHP3 vem crescendo numa velocidade incrível, e já está sendo desenvolvida a versão 4 do PHP.

2. Sintaxe Básica

Delimitando o código PHP

O código PHP fica embutido no próprio HTML. O interpretador identifica quando um código é PHP pelas seguintes tags:

```
<?php
comandos
?>

<script language="php">
comandos
</script>

<?
comandos
?>

<%
comandos
%>
```

O tipo de *tags* mais utilizado é o terceiro, que consiste em uma "abreviação" do primeiro. Para utilizá-lo, é necessário habilitar a opção *short-tags* na configuração do PHP. O último tipo serve para facilitar o uso por programadores acostumados à sintaxe de ASP. Para utilizá-lo também é necessário habilitá-lo no PHP, através do arquivo de configuração *php.ini*.

Separador de instruções

Entre cada instrução em PHP é preciso utilizar o ponto-e-vírgula, assim como em C, Perl e outras linguagens mais conhecidas. Na última instrução do bloco de script não é necessário o uso do ponto-e-vírgula, mas por questões estéticas recomenda-se o uso sempre.

Nomes de variáveis

Toda variável em PHP tem seu nome composto pelo caracter $ e uma string, que deve iniciar por uma letra ou o caracter "_". **PHP é case sensitive**, ou seja, as variáveis $vivas e $VIVAS são diferentes. Por isso é preciso ter muito cuidado ao definir os nomes das variáveis. É bom evitar os nomes em maiúsculas, pois como veremos mais adiante, o PHP já possui alguma variáveis pré-definidas cujos nomes são formados por letras maiúsculas.

Comentários

Há dois tipos de comentários em código PHP:

Comentários de uma linha:

Marca como comentário até o final da linha ou até o final do bloco de código PHP – o que vier antes. Pode ser delimitado pelo caracter "#" ou por duas barras (//).

```
Exemplo:
<?  echo "teste"; #isto é um teste  ?>
<?  echo "teste"; //este teste é similar ao anterior ?>
```

Comentários de mais de uma linha:

Tem como delimitadores os caracteres "/*" para o início do bloco e "*/" para o final do comentário. Se o delimitador de final de código PHP (?>) estiver dentro de um comentário, não será reconhecido pelo interpretador.

Exemplos:

```
<?
  echo "teste"; /* Isto é um comentário com mais
de uma linha, mas não funciona corretamente ?>
*/

<?
  echo "teste"; /* Isto é um comentário com mais
de uma linha que funciona corretamente
*/
?>
```

3. Criando os primeiros scripts

Primeiro Exemplo

Neste exemplo, criaremos um script com uma saída simples, que servirá para testar se a instalação foi feita corretamente:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

<?php
echo "Primeiro Script";
?>

</body>
</html>
```

Salve o arquivo como "primeiro.php3" no diretorio de documentos do Apache (ou o Web Server escolhido). Abra uma janela do navegador e digite o endereço "http://localhost/primeiro.php3". Verificando o código fonte da página exibida, temos o seguinte:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

Primeiro Script

</body>
</html>
```

Isso mostra como o PHP funciona. O script é executado no servidor, ficando disponível para o usuário apenas o resultado. Agora vamos escrever um script que produza exatamente o mesmo resultado utilizando uma variável:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

<?php
$texto = "Primeiro Script";
echo $texto;
?>

</body>
</html>
```

Utilizando formulários HTML

Ao clicar num botão “Submit” em um formulário HTML as informações dos campos serão enviadas ao servidor especificado para que possa ser produzida uma resposta. O PHP trata esses valores como variáveis, cujo nome é o nome do campo definido no formulário. O exemplo a seguir mostra isso, e mostra também como o código PHP pode ser inserido em **qualquer** parte do código HTML:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

<?php
if ($texto != "")
  echo "Você digitou \"$texto\"<br><br>";
?>

<form method=post action="<? echo $PATH_INFO; ?>">
<input type="text" name="texto" value="" size=10>
<br>
<input type="submit" name="sub" value="Enviar!">
</form>

</body>
</html>
```

Ao salvar o arquivo acima e carregá-lo no browser, o usuário verá apenas um formulário que contém um espaço para digitar o texto, como visto na figura 01. Ao digitar um texto qualquer e submeter o formulário, a resposta, que é o mesmo arquivo PHP (indicado pela constante $PATH_INFO, que retorna o nome do arquivo) será como na figura 02:

[Imagem16]	[Imagem17]
figura 01	*figura 02*

Isso ocorre porque o código PHP testa o conteúdo da variável $texto. Inicialmente ele é uma string vazia, e por isso nada é impresso na primeira parte. Quando algum texto é digitado no formulário e submetido, o PHP passa a tratá-lo como uma variável. Como no formulário o campo possui o nome “texto”, a variável com seu conteúdo será $texto. Assim, no próximo teste o valor da variável será diferente de uma string vazia, e o PHP imprime um texto antes do formulário.

Interagindo com o browser

PHP também permite interagir com informações do browser automaticamente. Por exemplo, o script a seguir mostra informações sobre o browser do usuário. As figuras 03 e 04 mostram o resultado visto no Netscape Communicator e o Microsoft Internet Explorer, respectivamente.

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

<? echo $HTTP_USER_AGENT; ?>

</body>
</html>
```

[Imagem18]

figura 03

[Imagem19]

figura 04

Observe que o resultado mostra características de cada browser, como a versão, e no caso do Communicator até o idioma ("en"). Com isso, se você criar uma página com recursos disponíveis somente no Internet Explorer, por exemplo, pode esconder o código dos outros browsers, com um código semelhante ao seguinte:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

<?
if  (strpos($HTTP_USER_AGENT,"MSIE 5") != 0) {
  echo "Você usa Internet Explorer";
} else {
  echo "Você não usa Internet Explorer";
}
?>

</body>
</html>
```

Neste exemplo, será apenas exibido um texto informando se está sendo utilizado o Microsoft Internet Explorer ou não, mas para outras funções poderia ser utilizado algo semelhante.

É bom notar o surgimento de mais uma função no código anterior: `strpos(string1,string2)`. Essa função retorna a posição da primeira aparição de string2 em string1, contando a partir de zero, e não retorna valor algum se não ocorrer. Assim, para testar se a string `$HTTP_USER_AGENT` contém a string `"MSIE"`, basta testar se `strpos` devolve algum valor.

Acessando Bancos de Dados

Neste documento todos os exemplos referentes a acesso de bancos de dados utilizarão o gerenciador de banco de dados MySQL, que pode ser copiado gratuitamente no site *http://www.mysql.org*.

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.

Conexão com o servidor

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

```
int mysql_connect(string /*host [:porta]*/ , string /*login*/ , string /*senha*/ );
```

Os parâmetros são bastante simples: o endereço do servidor(host), o nome do usuário (login) e a senha para a conexão. A função retorna um valor inteiro, que é o identificador da conexão estabelecida e deverá ser armazenado numa variável para ser utilizado depois. No nosso exemplo, temos como servidor de banco de dados a mesma máquina que roda o servidor http, como login o usuário "root" e senha "phppwd":

```
$conexao = mysql_connect("localhost", "root", "phppwd");
```

Assim, se a conexão for bem sucedida (existir um servidor no endereço especificado que possua o usuário com a senha fornecida), o identificador da conexão fica armazenado na variável `$conexão`.

Seleção do banco de dados

Uma vez conectado, é preciso selecionar o banco de dados existente no servidor com o qual desejamos trabalhar. Isso é feito através da função int `mysql_select_db`, que possui a seguinte sintaxe:

```
int mysql_select_db(string /*nome_base*/, int /*conexao*/ );
```

O valor de retorno é 0 se o comando falhar, e 1 em caso de sucesso. O nome da base de dados a selecionar é o primeiro parâmetro fornecido, seguido pelo identificador da conexão. Se este for omitido, o interpretador PHP tentará utilizar a última conexão estabelecida. Recomenda-se sempre explicitar esse valor, para facilitar a legibilidade do código. No nosso exemplo, a base de dados a ser selecionada possui o nome "ged":

```
mysql_select_db("ged", $conexao);
```

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

Execução de queries SQL

Após estabelecida a conexão e selecionada a base de dados a ser utilizada, quase toda a interação 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:

```
int mysql_query(string consulta, int [conexao] );
```

O valor de retorno é 0 se falhar ou 1 em caso de sucesso. Sucesso aqui significa que a consulta está 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. No caso da consulta ser um comando SELECT, o valor de retorno é um valor interno que identifica o resultado, que poderá ser tratado com a função `mysql_result()` e outras. A string query não deve conter ponto-e-vírgula no final do comando, e o identificador da conexão é opcional. Vamos criar uma tabela como exemplo:

```
$cria = "CREATE TABLE exemplo (codigo INT AUTO_INCREMENT PRIMARY KEY, nome CHAR(40), email CHAR(50))";

mysql_query($cria, $conexao);
```

Agora vejamos como ficou o código completo para executar uma query SQL numa base de dados mySQL, com um exemplo que cria uma tabela chamada exemplo e adiciona alguns dados:

```
$conexao = mysql_connect("localhost", "root", "phppwd");
mysql_select_db("ged", $conexao);

$cria = "CREATE TABLE exemplo (codigo INT AUTO_INCREMENT PRIMARY KEY, nome CHAR(40), email CHAR(50))";

$insere1 = "INSERT INTO exemplo (nome,email) VALUES ("Mauricio Vivas","vivas@usa.net");

$insere2 = "INSERT INTO exemplo (nome,email) VALUES ("Jose da Silva","jose@teste.com");

$insere3 = "INSERT INTO exemplo (nome,email) VALUES ("Fernando Henrique Cardoso","fhc@planalto.gov.br");

$insere4 = "INSERT INTO exemplo (nome,email) VALUES ("Bill Clinton","president@whitehouse.gov");

mysql_query($cria, $conexao);
mysql_query($insere1, $conexao);
mysql_query($insere2, $conexao);
mysql_query($insere3, $conexao);
mysql_query($insere4, $conexao);
```

Tratamento de resultados de 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. Duas maneiras interessantes de fazê-lo usam o comando `mysql_result` e o comando `mysql_fetch_row`, respectivamente.

O comando `mysql_result` tem a seguinte sintaxe:

```
int mysql_result(int resultado, int linha, mixed [campo]);
```

Onde `resultado` é o identificador do resultado, obtido com o retorno da função `mysql_query`, linha especifica a tupla a ser exibida, já que uma query SELECT pode retornar diversas tuplas, e `campo` é o identificador do campo a ser exibido, sendo o tipo descrito como `mixed` pela possibilidade de ser de diversos tipos (neste caso, inteiro ou string). Vejamos um exemplo utilizando a tabela criada anteriormente:

```
$consulta = “SELECT nome, email FROM exemplo WHERE email LIKE ‘vivas’”;

$resultado = mysql_query($consulta, $conexao);

printf("Nome: ", mysql_result($resultado,0,"nome"), “<br>\n”);
printf("e-mail: ", mysql_result($resultado,0,"email"),“<br>”);
```

Com o exemplo acima, o resultado será:

```
Nome: Mauricio Vivas<br>
e-mail: vivas@usa.net<br>
```

É importante notar que a utilização desta função é um pouco trabalhosa, já que no caso de um resultado com várias linhas é preciso controlar o número de linhas para tratá-las (pode-se utilizar a função `mysql_num_rows(int resultado)`, que retorna o número de linhas de um resultado), e no caso de uma alteração no nome do campo é preciso alterar também a maneira de tratá-lo. Por isso é mais aconselhável que se use uma outra função, como por exemplo `mysql_fetch_row`, que possui a seguinte sintaxe:

```
array mysql_fetch_row(int result);
```

A variável `resultado` é o identificador da memória de resultados, obtido como retorno da função `mysql_query`. O resultado produzido por esta função é de retirar a primeira linha da memória de resultados, se houver, e colocá-la num array. Assim torna-se mais fácil tratar um resultado com várias linhas, e sem utilizar os nomes dos campos na rotina de tratamento do resultado:

```
$consulta = “SELECT nome, email FROM exemplo”;

$resultado = mysql_query($consulta, $conexao);

echo "<table border=1>\n";
echo "<tr><td>Nome</td><td>e-mail</tr>\n";
while ($linha = mysql_fetch_row($resultado)) {
  printf("<tr><td>$linha[0]</td>”);
  printf("<td>$linha[1]</td></tr>”);
}
echo "</table>\n";
```

O código acima irá imprimir todos os registros da tabela exemplo numa tabela html. Se o programador desejar “pular” alguma(s) linha(s) do resultado, poderá utilizar a função `mysql_data_seek`, que tem por objetivo definir qual será a próxima linha da memória de resultados a ser impressa. Sua sintaxe é:

```
int mysql_data_seek(int resultado, int linha);
```

Sendo `resultado` o identificador do resultado e `linha` o numero da linha. Retorna 0 em caso de falha, e um valor diferente de zero em caso de sucesso.

Existem diversas outras funções para o tratamento de resultados, que armazenam as linhas em arrays e objetos, assim como outras funções para administrar o banco de dados, mas como este documento trata-se de uma introdução, inicialmente não tratará tópicos mais avançados.

4. Tipos

Tipos Suportados

PHP suporta os seguintes tipos de dados:

- Inteiro
- Ponto flutuante
- String
- Array
- Objeto

PHP utiliza checagem de tipos dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do script. Por este motivo não é necessário declarar o tipo de uma variável para usá-la. O interpretador PHP decidirá qual o tipo daquela variável, verificando o conteúdo em tempo de execução.

Ainda assim, é permitido converter os valores de um tipo para outro desejado, utilizando o *typecasting* ou a função `settype` (ver adiante).

Inteiros (integer ou long)

Uma variável pode conter um valor inteiro com atribuições que sigam as seguintes sintaxes:

```
$vivas = 1234; # inteiro positivo na base decimal
$vivas = -234; # inteiro negativo na base decimal
$vivas = 0234; # inteiro na base octal-simbolizado pelo 0
               # equivale a 156 decimal
$vivas = 0x34; # inteiro na base hexadecimal(simbolizado
               # pelo 0x) – equivale a 52 decimal.
```

A diferença entre inteiros simples e `long` está no número de bytes utilizados para armazenar a variável. Como a escolha é feita pelo interpretador PHP de maneira transparente para o usuário, podemos afirmar que os tipos são iguais.

Números em Ponto Flutuante (double ou float)

Uma variável pode ter um valor em ponto flutuante com atribuições que sigam as seguintes sintaxes:

```
$vivas = 1.234;
$vivas = 23e4; # equivale a 230.000
```

Strings

Strings podem ser atribuídas de duas maneiras:

a) utilizando aspas simples (') – Desta maneira, o valor da variável será exatamente o texto contido entre as aspas (com exceção de \\ e \' – ver tabela abaixo)

b) utilizando aspas duplas (") – Desta maneira, qualquer variável ou caracter de escape será expandido antes de ser atribuído.

Exemplo:

```
<?
$teste = "Mauricio";
$vivas = '---$teste--\n';
echo "$vivas";
?>
```

A saída desse script será "---$teste--\n".

```
<?
$teste = "Mauricio";
$vivas = "---$teste---\n";
echo "$vivas";
?>
```

A saída desse script será "---Mauricio--" (com uma quebra de linha no final).

A tabela seguinte lista os caracteres de escape:

Sintaxe	**Significado**
\n	Nova linha
\r	Retorno de carro (semelhante a \n)
\t	Tabulação horizontal
\\	A própria barra (\ )
\$	O símbolo $
\’	Aspa simples
\”	Aspa dupla

No apêndice 01 está disponível uma lista das funções utilizadas no tratamento de strings.

Arrays

Arrays em PHP podem ser observados como mapeamentos ou como vetores indexados. Mais precisamente, um valor do tipo array é um dicionário onde os índices são as chaves de acesso. Vale ressaltar que os índices podem ser valores de qualquer tipo e não somente inteiros. Inclusive, se os índices forem todos inteiros, estes não precisam formar um intervalo contínuo

Como a checagem de tipos em PHP é dinâmica, valores de tipos diferentes podem ser usados como índices de array, assim como os valores mapeados também podem ser de diversos tipos.

```
Exemplo:
<?
$cor[1] = “vermelho”;
$cor[2] = “verde”;
$cor[3] = “azul”;
$cor[“teste”] = 1;
?>

Equivalentemente, pode-se escrever:

<?
$cor = array(1 => “vermelho, 2 => “verde, 3 => “azul”, “teste => 1);
?>
```

Listas

As listas são utilizadas em PHP para realizar atribuições múltiplas. Através de listas é possível atribuir valores que estão num array para variáveis. Vejamos o exemplo:

```
Exemplo:

 list($a, $b, $c) = array(“a”, “b”, “c”);
```

O comando acima atribui valores às três variáveis simultaneamente. É bom notar que só são atribuídos às variáveis da lista os elementos do array que possuem índices inteiros e não negativos. No exemplo acima as três atribuições foram bem sucedidas porque ao inicializar um array sem especificar os índices eles passam a ser inteiros, a partir do zero. Um fator importante é que cada variável da lista possui um índice inteiro e ordinal, iniciando com zero, que serve para determinar qual valor será atribuído. No exemplo anterior temos $a com índice 0, $b com índice 1 e $c com índice 2. Vejamos um outro exemplo:

```
$arr = array(1=>”um”,3=>”tres”,”a”=>”letraA”,2=>”dois);
list($a,$b,$c,$d) = $arr;
```

Após a execução do código acima temos os seguintes valores:

```
$a == null
$b == “um”
$c == “dois”
$d == “tres”
```

Devemos observar que à variável $a não foi atribuído valor, pois no array não existe elemento com índice 0 (zero). Outro detalhe importante é que o valor “tres” foi atribuído à variável $d, e não a $b, pois seu índice é 3, o mesmo que $d na lista. Por fim, vemos que o valor “letraA” não foi atribuído a elemento algum da lista pois seu índice não é inteiro.

Os índices da lista servem apenas como referência ao interpretador PHP para realizar as atribuições, não podendo ser acessados de maneira alguma pelo programador. De maneira diferente do array, uma lista não pode ser atribuída a uma variável, servindo apenas para fazer múltiplas atribuições através de um array.

No apêndice 02 está disponível uma lista das funções mais comuns para o tratamento de arrays.

Objetos

Um objeto pode ser inicializado utilizando o comando *new* para instanciar uma classe para uma variável.

```
Exemplo:
class teste {
          function nada() {
               echo “nada”;
          }
}

$vivas = new teste;
$vivas -> nada();
```

A utilização de objetos será mais detalhada mais à frente.

Booleanos

PHP não possui um tipo booleano, mas é capaz de avaliar expressões e retornar *true* ou *false*, através do tipo `integer`: é usado o valor 0 (zero) para representar o estado *false*, e qualquer valor diferente de zero (geralmente 1) para representar o estado *true*.

Transformação de tipos

A transformação de tipos em PHP pode ser feita das seguintes maneiras:

Coerções

Quando ocorrem determinadas operações (“+”, por exemplo) entre dois valores de tipos diferentes, o PHP converte o valor de um deles automaticamente (coerção). É interessante notar que se o operando for uma variável, seu valor não será alterado.

O tipo para o qual os valores dos operandos serão convertidos é determinado da seguinte forma: Se um dos operandos for `float`, o outro será convertido para `float`, senão, se um deles for `integer`, o outro será convertido para `integer`.

```
Exemplo:
$vivas = “1”;        // $vivas é a string “1”
$vivas = $vivas + 1; // $vivas é o integer 2
$vivas = $vivas + 3.7;// $vivas é o double 5.7
$vivas = 1 + 1.5          // $vivas é o double 2.5
```

Como podemos notar, o PHP converte `string` para `integer` ou `double` mantendo o valor. O sistema utilizado pelo PHP para converter de *strings* para números é o seguinte:

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

```
Exemplos:
$vivas = 1 + “10.5”;      // $vivas == 11.5
$vivas = 1 + “-1.3e3”;    // $vivas == -1299
$vivas = 1 + “teste10.5”; // $vivas == 1
$vivas = 1 + “10testes”;  // $vivas == 11
$vivas = 1 + "  10testes";     // $vivas == 11
$vivas = 1 + "+  10testes";    // $vivas == 1
```

Transformação explícita de tipos

A sintaxe do *typecast* de PHP é semelhante ao C: basta escrever o tipo entre parenteses antes do valor

```
Exemplo:
$vivas = 15;             // $vivas é integer (15)
$vivas = (double) $vivas  // $vivas é double (15.0)
$vivas = 3.9              // $vivas é double (3.9)
$vivas = (int) $vivas     // $vivas é integer (3)
                          // o valor decimal é truncado
```

Os tipos de *cast* permitidos são:

(int), (integer) ⇒ muda para integer;

(real), (double), (float) ⇒ muda para float;

(string) ⇒ muda para string;

(array) ⇒ muda para array;

(object) ⇒ muda para objeto.

Com a função settype

A função `settype` converte uma variável para o tipo especificado, que pode ser “integer”, “double”, “string”, “array” ou “object”.

```
Exemplo:
$vivas = 15;              // $vivas é integer
settype($vivas,double)    // $vivas é double
```

5. Constantes

Constantes pré-definidas

O PHP possui algumas constantes pré-definidas, indicando a versão do PHP, o Sistema Operacional do servidor, o arquivo em execução, e diversas outras informações. Para ter acesso a todas as constantes pré-definidas, pode-se utilizar a função `phpinfo()`, que exibe uma tabela contendo todas as constantes pré-definidas, assim como configurações da máquina, sistema operacional, servidor http e versão do PHP instalada.

Definindo constantes

Para definir constantes utiliza-se a função `define`. Uma vez definido, o valor de uma constante não poderá mais ser alterado. Uma constante só pode conter valores escalares, ou seja, não pode conter nem um array nem um objeto. A assinatura da função `define` é a seguinte:

```
int define(string nome_da_constante, mixed valor);
```

A função retorna `true` se for bem-sucedida. Veja um exemplo de sua utilização a seguir:

```
define ("pi", 3.1415926536);
$circunf                    =                    2*pi*$raio;
```

6. Operadores

Aritméticos

Só podem ser utilizados quando os operandos são números (integer ou float). Se forem de outro tipo, terão seus valores convertidos antes da realização da operação.

+	adição
-	subtração
*	multiplicação
/	divisão
%	módulo

de strings

Só há um operador exclusivo para strings:

.	concatenação

de atribuição

Existe um operador básico de atribuição e diversos derivados. Sempre retornam o valor atribuído. No caso dos operadores derivados de atribuição, a operação é feita entre os dois operandos, sendo atribuído o resultado para o primeiro. A atribuição é sempre por valor, e não por referência.

=	atribuição simples
+=	atribuição com adição
-=	atribuição com subtração
*=	atribuição com multiplicação
/=	atribuição com divisão
%=	atribuição com módulo
.=	atribuição com concatenação

```
Exemplo:

$a = 7;
$a += 2; // $a passa a conter o valor 9
```

bit a bit

Comparam dois números bit a bit.

&	“e” lógico
|	“ou” lógico
^	ou exclusivo
~	não (inversão)
<<	shift left
>>	shift right

Lógicos

Utilizados para inteiros representando valores booleanos

and	“e” lógico
or	“ou” lógico
xor	ou exclusivo
!	não (inversão)
&&	“e” lógico
||	“ou” lógico

Existem dois operadores para “e” e para “ou porque eles têm diferentes posições na ordem de precedência.

Comparação

As comparações são feitas entre os valores contidos nas variáveis, e não as referências. Sempre retornam um valor booleano.

==	igual a
!=	diferente de
<	menor que

>	maior que
<=	menor ou igual a
>=	maior ou igual a

Expressão condicional

Existe um operador de seleção que é ternário. Funciona assim:

```
(expressao1)?(expressao2):( expressao3)
```

o interpretador PHP avalia a primeira expressão. Se ela for verdadeira, a expressão retorna o valor de expressão2. Senão, retorna o valor de expressão3.

de incremento e decremento

++	incremento
--	decremento

Podem ser utilizados de duas formas: antes ou depois da variável. Quando utilizado antes, retorna o valor da variável antes de incrementá-la ou decrementá-la. Quando utilizado depois, retorna o valor da variável já incrementado ou decrementado.

```
Exemplos:

$a = $b = 10; // $a e $b recebem o valor 10
$c = $a++; // $c recebe 10 e $a passa a ter 11
$d = ++$b; // $d recebe 11, valor de $b já incrementado
```

Ordem de precedência dos operadores

A tabela a seguir mostra a ordem de precedência dos operadores no momento de avaliar as expressões;

Precedência	Associatividade	Operadores
1.	esquerda	,
2.	esquerda	or
3.	esquerda	xor
4.	esquerda	and
5.	direita	print
6.	esquerda	= += -= *= /= .= %= &= != ~= <<= >>=
7.	esquerda	? :
8.	esquerda	||
9.	esquerda	&&
10.	esquerda	|
11.	esquerda	^
12.	esquerda	&
13.	não associa	== !=
14.	não associa	< <= > >=
15.	esquerda	<< >>
16.	esquerda	+ - .
17.	esquerda	* / %
18.	direita	! ~ ++ -- (int) (double) (string) (array) (object) @
19.	direita	[
20.	não associa	new

7. Estruturas de Controle

As estruturas que veremos a seguir são comuns para as linguagens de programação imperativas, bastando, portanto, descrever a sintaxe de cada uma delas, resumindo o funcionamento.

Blocos

Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Em comandos como `if`, `for`, `while`, `switch` e em declarações de funções blocos podem ser utilizados para permitir que um comando faça parte do contexto desejado. Blocos em PHP são delimitados pelos caracteres "{" e "}". A utilização dos delimitadores de bloco em uma parte qualquer do código não relacionada com os comandos citados ou funções não produzirá efeito algum, e será tratada normalmente pelo interpretador.

```
Exemplo:
if ($x == $y)
  comando1;
  comando2;
```

Para que comando2 esteja relacionado ao if é preciso utilizar um bloco:

```
if ($x == $y){
  comando1;
  comando2;
}
```

Comandos de seleção

Também chamados de condicionais, os comandos de seleção permitem executar comandos ou blocos de comandos com base em testes feitos durante a execução.

if

O mais trivial dos comandos condicionais é o `if`. Ele testa a condição e executa o comando indicado se o resultado for `true` (valor diferente de zero). Ele possui duas sintaxes:

```
if (expressão)
   comando;
```

```
if (expressão):
   comando;
   . . .
```

```
   comando;
endif;
```

Para incluir mais de um comando no `if` da primeira sintaxe, é preciso utilizar um bloco, demarcado por chaves.

O `else` é um complemento opcional para o `if`. Se utilizado, o comando será executado se a expressão retornar o valor `false` (zero). Suas duas sintaxes são:

```
if (expressão)
   comando;
else
   comando;
```

```
if (expressão):
   comando;
   . . .
   comando;
else
   comando;
   . . .
   comando;
endif;
```

A seguir, temos um exemplo do comando `if` utilizado com `else`:

```
if ($a > $b)
  $maior = $a;
else
  $maior = $b;
```

O exemplo acima coloca em $maior o maior valor entre $a e $b

Em determinadas situações é necessário fazer mais de um teste, e executar condicionalmente diversos comandos ou blocos de comandos. Para facilitar o entendimento de uma estrutura do tipo:

```
if (expressao1)
   comando1;
else
  if (expressao2)
    comando2;
  else
    if (expressao3)
      comando3;
    else
      comando4;
```

foi criado o comando, também opcional `elseif`. Ele tem a mesma função de um `else` e um `if` usados sequencialmente, como no exemplo acima. Num mesmo `if` podem ser utilizados diversos `elseif`'s, ficando essa utilização a critério do programador, que deve zelar pela legibilidade de seu script.

O comando `elseif` também pode ser utilizado com dois tipos de sintaxe. Em resumo, a sintaxe geral do comando `if` fica das seguintes maneiras:

```
if (expressao1)
  comando;
[ elseif (expressao2)
   comando; ]
[ else
   comando; ]


if (expressao1) :
  comando;
   . . .
  comando;
[ elseif (expressao2)
  comando;
   . . .
  comando; ]
[ else
  comando;
   . . .
  comando; ]
endif;
```

switch

O comando `switch` atua de maneira semelhante a uma série de comandos `if` na mesma expressão. Frequentemente o programador pode querer comparar uma variável com diversos valores, e executar um código diferente a depender de qual valor é igual ao da variável. Quando isso for necessário, deve-se usar o comando `switch`. O exemplo seguinte mostra dois trechos de código que fazem a mesma coisa, sendo que o primeiro utiliza uma série de `if`'s e o segundo utiliza `switch`:

```
if ($i == 0)
   print “i é igual a zero”;
elseif ($i == 1)
   print “i é igual a um”;
elseif ($i == 2)
   print “i é igual a dois”;

switch ($i) {
case 0:
     print “i é igual a zero”;
     break;
```

```
case 1:
     print “i é igual a um”;
     break;
case 2:
     print “i é igual a dois”;
     break;
}
```

É importante compreender o funcionamento do `switch` para não cometer enganos. O comando `switch` testa linha a linha os cases encontrados, e a partir do momento que encontra um valor igual ao da variável testada, passa a executar todos os comandos seguintes, mesmo os que fazem parte de outro teste, até o fim do bloco. por isso usa-se o comando `break`, quebrando o fluxo e fazendo com que o código seja executado da maneira desejada. Veremos mais sobre o `break` mais adiante. Veja o exemplo:

```
switch ($i) {
case 0:
     print “i é igual a zero”;
case 1:
     print “i é igual a um”;
case 2:
     print “i é igual a dois”;
    }
```

No exemplo acima, se $i for igual a zero, os três comandos “print” serão executados. Se $i for igual a 1, os dois últimos “print” serão executados. O comando só funcionará da maneira desejada se $i for igual a 2.

Em outras linguagens que implementam o comando switch, ou similar, os valores a serem testados só podem ser do tipo inteiro. Em PHP é permitido usar valores do tipo string como elementos de teste do comando switch. O exemplo abaixo funciona perfeitamente:

```
switch ($s) {
case “casa”:
     print “A casa é amarela”;
case “arvore”:
     print “a árvore é bonita”;
case “lampada”:
     print “joao apagou a lampada”;
    }
```

comandos de repetição

while

O `while` é o comando de repetição (laço) mais simples. Ele testa uma condição e executa um comando, ou um bloco de comandos, até que a condição testada seja falsa. Assim como o `if`, o `while` também possui duas sintaxes alternativas:

```
while (<expressao>)
   <comando>;
```

```
while (<expressao>):
   <comando>;
    . . .
   <comando>;
endwhile;
```

A expressão só é testada a cada vez que o bloco de instruções termina, além do teste inicial. Se o valor da expressão passar a ser `false` no meio do bloco de instruções, a execução segue até o final do bloco. Se no teste inicial a condição for avaliada como `false`, o bloco de comandos não será executado.

O exemplo a seguir mostra o uso do `while` para imprimir os números de 1 a 10:

```
$i = 1;
while ($i <=10)
  print $i++;
```

do... while

O laço `do..while` funciona de maneira bastante semelhante ao `while`, com a simples diferença que a expressão é testada ao final do bloco de comandos. O laço `do..while` possui apenas uma sintaxe, que é a seguinte:

```
do {
   <comando>
   . . .
   <comando>
} while (<expressao>);
```

O exemplo utilizado para ilustrar o uso do `while` pode ser feito da seguinte maneira utilizando o `do.. while`:

```
$i = 0;
do {
  print ++$i;
} while ($i < 10);
```

for

O tipo de laço mais complexo é o `for`. Para os que programam em C, C++ ou Java, a assimilação do funcionamento do for é natural. Mas para aqueles que estão acostumados a linguagens como Pascal, há uma grande mudança para o uso do for. As duas sintaxes permitidas são:

```
for (<inicializacao>;<condicao>;<incremento>)
   <comando>;
```

```
for (<inicializacao>;<condicao>;<incremento>) :
   <comando>;
   . . .
   <comando>;
endfor;
```

As três expressões que ficam entre parênteses têm as seguintes finalidades:

Inicialização: comando ou sequencia de comandos a serem realizados antes do inicio do laço. Serve para inicializar variáveis.

Condição: Expressão booleana que define se os comandos que estão dentro do laço serão executados ou não. Enquanto a expressão for verdadeira (valor diferente de zero) os comandos serão executados.

Incremento: Comando executado ao final de cada execução do laço.

Um comando `for` funciona de maneira semelhante a um `while` escrito da seguinte forma:

```
<inicializacao>
while (<condicao>) {
comandos
...
<incremento>
}
```

Quebra de fluxo

Break

O comando `break` pode ser utilizado em laços de `do`, `for` e `while`, além do uso já visto no comando `switch`. Ao encontrar um `break` dentro de um desses laços, o interpretador PHP para imediatamente a execução do laço, seguindo normalmente o fluxo do script.

```
while ($x > 0) {
  ...
  if ($x == 20) {
```

```
        echo "erro! x = 20";
        break;
    ...
}
```

No trecho de código acima, o laço `while` tem uma condição para seu término normal ($x <= 0), mas foi utilizado o break para o caso de um término não previsto no início do laço. Assim o interpretador seguirá para o comando seguinte ao laço.

Continue

O comando `continue` também deve ser utilizado no interior de laços, e funciona de maneira semelhante ao `break`, com a diferença que o fluxo ao invés de sair do laço volta para o início dele. Vejamos o exemplo:

```
for ($i = 0; $i < 100; $i++) {
  if ($i % 2) continue;
  echo " $i ";
}
```

O exemplo acima é uma maneira ineficiente de imprimir os números pares entre 0 e 99. O que o laço faz é testar se o resto da divisão entre o número e 2 é 0. Se for diferente de zero (valor lógico `true`) o interpretador encontrará um `continue`, que faz com que os comandos seguintes do interior do laço sejam ignorados, seguindo para a próxima iteração.

8. Funções

Definindo funções

A sintaxe básica para definir uma função é:

```
function nome_da_função([arg1, arg2, arg3]) {
        Comandos;
        ... ;
        [return <valor de retorno>];
}
```

Qualquer código PHP válido pode estar contido no interior de uma função. Como a checagem de tipos em PHP é dinâmica, o tipo de retorno não deve ser declarado, sendo necessário que o programador esteja atento para que a função retorne o tipo desejado. É recomendável que esteja tudo bem documentado para facilitar a leitura e compreensão do código. Para efeito de documentação, utiliza-se o seguinte formato de declaração de função:

```
tipo function nome_da_funcao(tipo arg1, tipo arg2, ...);
```

Este formato só deve ser utilizado na documentação do script, pois o PHP não aceita a declaração de tipos. Isso significa que em muitos casos o programador deve estar atento ao tipos dos valores passados como parâmetros, pois se não for passado o tipo esperado não é emitido nenhum alerta pelo interpretador PHP, já que este não testa os tipos.

Valor de retorno

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum.

Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou arrays.

Argumentos

É possível passar argumentos para uma função. Eles devem ser declarados logo após o nome da função, entre parênteses, e tornam-se variáveis pertencentes ao escopo local da função. A declaração do tipo de cada argumento também é utilizada apenas para efeito de documentação.

Exemplo:

```
function imprime($texto){
   echo $texto;
}

imprime("teste de funções");
```

Passagem de parâmetros por referência

Normalmente, a passagem de parâmetros em PHP é feita por valor, ou seja, se o conteúdo da variável for alterado, essa alteração não afeta a variável original.

Exemplo:

```
function mais5($numero) {
   $numero += 5;
}

$a = 3;
mais5($a); //$a continua valendo 3
```

No exemplo acima, como a passagem de parâmetros é por valor, a função mais5 é inútil, já que após a execução sair da função o valor anterior da variável é recuperado. Se a passagem de valor fosse feita por referência, a variável $a teria 8 como valor. O que ocorre normalmente é que ao ser chamada uma função, o interpretador salva todo o escopo atual, ou seja, os conteúdos das variáveis. Se uma dessas variáveis for passada como parâmetro, seu conteúdo fica preservado, pois a função irá trabalhar na verdade com uma cópia da variável. Porém, se a passagem de parâmetros for feita por referência, toda alteração que a função realizar no valor passado como parâmetro afetará a variável que o contém.

Há duas maneiras de fazer com que uma função tenha parâmetros passados por referência: indicando isso na declaração da função, o que faz com que a pasagem de parâmetros sempre seja assim; e também na própria chamada da função. Nos dois casos utiliza-se o modificador "&". Vejamos um exemplo que ilustra os dois casos:

```
function mais5(&$num1, $num2) {
   $num1 += 5;
   $num2 += 5;
}

$a = $b = 1;
mais5($a, $b); /* Neste caso, só $num1 terá seu valor alterado, pois a passagem por referência está definida na declaração da função. */


mais5($a, &$b); /* Aqui as duas variáveis terão seus valores alterados. */
```

Argumentos com valores pré-definidos (default)

Em PHP é possível ter valores *default* para argumentos de funções, ou seja, valores que serão assumidos em caso de nada ser passado no lugar do argumento. Quando algum parâmetro é declarado desta maneira, a passagem do mesmo na chamada da função torna-se opcional.

```
function teste($vivas = “testando”) {
   echo $vivas;
}

teste(); // imprime “testando”
teste(“outro teste”); // imprime “outro teste”
```

É bom lembrar que quando a função tem mais de um parâmetro, o que tem valor *default* deve ser declarado por último:

```
function teste($figura = circulo, $cor) {
  echo “a figura é um “, $figura, “ de cor “ $cor;
}

teste(azul);
/* A função não vai funcionar da maneira esperada, ocorrendo um erro no interpretador. A declaração correta é: */

function teste2($cor, $figura = circulo) {
  echo “a figura é um “, $figura, “ de cor “ $cor;
}

teste2(azul);

/* Aqui a funcao funciona da maneira esperada, ou seja, imprime o texto: “a figura é um círculo de cor azul” */
```

Contexto

O contexto é o conjunto de variáveis e seus respectivos valores num determinado ponto do programa. Na chamada de uma função, ao iniciar a execução do bloco que contém a implementação da mesma é criado um novo contexto, contendo as variáveis declaradas dentro do bloco, ou seja, todas as variáveis utilizadas dentro daquele bloco serão eliminadas ao término da execução da função.

Escopo

O escopo de uma variável em PHP define a porção do programa onde ela pode ser utilizada. Na maioria dos casos todas as variáveis têm escopo global. Entretanto, em funções definidas pelo usuário um escopo local é criado. Uma variável de escopo global não pode ser utilizada no interior de uma função sem que haja uma declaração.

```
Exemplo:
$vivas = “Testando”;

function Teste() {
  echo $vivas;
}

Teste();
```

O trecho acima não produzirá saída alguma, pois a variável $vivas é de escopo global, e não pode ser referida num escopo local, mesmo que não haja outra com nome igual que cubra a sua visibilidade. Para que o script funcione da forma desejada, a variável global a ser utilizada deve ser declarada.

```
Exemplo:
$vivas = “Testando”;

function Teste() {
  global $vivas;
  echo $vivas;
}

Teste();
```

Uma declaração “global” pode conter várias variáveis, separadas por vírgulas. Uma outra maneira de acessar variáveis de escopo global dentro de uma função é utilizando um array pré-definido pelo PHP cujo nome é $GLOBALS. O índice para a variável referida é o proprio nome da variável, sem o caracter $. O exemplo acima e o abaixo produzem o mesmo resultado:

```
Exemplo:
$vivas = "Testando";

function Teste() {
  echo $GLOBALS["vivas"]; // imprime $vivas
  echo $vivas; // não imprime nada
}

Teste();
```

9. Variáveis

O modificador static

Uma variável estática é visível num escopo local, mas ela é inicializada apenas uma vez e seu valor não é perdido quando a execução do script deixa esse escopo. Veja o seguinte exemplo:

```
function Teste() {
  $a = 0;
  echo $a;
  $a++;
}
```

O último comando da função é inútil, pois assim que for encerrada a execução da função a variável $a perde seu valor. Já no exemplo seguinte, a cada chamada da função a variável $a terá seu valor impresso e será incrementada:

```
function Teste() {
  static $a = 0;
  echo $a;
  $a++;
}
```

O modificador static é muito utilizado em funções recursivas, já que o valor de algumas variáveis precisa ser mantido. Ele funciona da seguinte forma: O valor das variáveis declaradas como estáticas é mantido ao terminar a execução da função. Na próxima execução da função, ao encontrar novamente a declaração com static, o valor da variável é recuperado.

Em outras palavras, uma variável declarada como `static` tem o mesmo "tempo de vida" que uma variável global, porém sua visibilidade é restrita ao escopo local em que foi declarada e só é recuperada após a declaração.

Exemplo:

```
function Teste() {
  echo "$a";
  static $a = 0;
  $a++;
}
```

O exemplo acima não produzirá saída alguma. Na primeira execução da função, a impressão ocorre antes da atribuição de um valor à função, e portanto o conteúdo de $a é nulo (string vazia). Nas execuções seguintes da função Teste() a impressão ocorre antes da recuperação do valor de `$a`, e portanto nesse momento seu valor ainda é nulo. Para que a função retorne algum valor o modificador `static` deve ser utilizado.

Variáveis Variáveis

O PHP tem um recurso conhecido como variáveis variáveis, que consiste em variáveis cujos nomes também são variáveis. Sua utilização é feita através do duplo cifrão ($$).

```
$a = "teste";
$$a = "Mauricio Vivas";

O exemplo acima e equivalente ao seguinte:

$a = "teste";
$teste = "Mauricio Vivas";
```

Variáveis enviadas pelo navegador

Para interagir com a navegação feita pelo usuário, é necessário que o PHP possa enviar e receber informações para o software de navegação. A maneira de enviar informações, como já foi visto anteriormente, geralmente é através de um comando de impressão, como o *echo*. Para receber informações vindas do navegador através de um *link* ou um formulário html o PHP utiliza as informações enviadas através da URL. Por exemplo: se seu script php está localizado em `"http://localhost/teste.php3"` e você o chama com a url `"http://localhost/teste.php3?vivas=teste"`, automaticamente o PHP criará uma variável com o nome $vivas contendo a string "teste". Note que o conteúdo da variável está no formato `urlencode`. Os formulários html já enviam informações automaticamente nesse formato, e o PHP decodifica sem necessitar de tratamento pelo programador.

URLencode

O formato `urlencode` é obtido substituindo os espaços pelo caracter "+" e todos os outros caracteres não alfa-numéricos (com exceção de "_") pelo caracter "%" seguido do código ASCII em hexadecimal.

Por exemplo: o texto "Testando 1 2 3 !!" em urlencode fica "Testando+1+2+3+%21%21"

O PHP possui duas funções para tratar com texto em urlencode. Seguem suas sintaxes:

```
string urlencode(string texto);
string urldecode(string texto);
```

Essas funções servem respectivamente para codificar ou decodificar um texto passado como argumento. Para entender melhor o que é um argumento e como funciona uma função, leia o tópico “funções”.

Variáveis de ambiente

O PHP possui diversas variáveis de ambiente, como a `$PHP_SELF`, por exemplo, que contém o nome e o path do próprio arquivo. Algumas outras contém informações sobre o navegador do usuário, o servidor http, a versão do PHP e diversas informações. Para ter uma listagem de todas as variáveis e constantes de ambiente e seus respectivos conteúdos, deve-se utilizar a função `phpinfo()`.

Verificando o tipo de uma variável

Por causa da tipagem dinâmica utilizada pelo PHP, nem sempre é possível saber qual o tipo de uma variável em determinado instantese não contar com a ajuda de algumas funções que ajudam a verificar isso. A verificação pode ser feita de duas maneiras:

Função que retorna o tipo da variável

Esta função é a `gettype`. Sua assinatura é a seguinte:

```
string gettype(mixed var);
```

A palavra “mixed” indica que a variável var pode ser de diversos tipos.

A função `gettype` pode retornar as seguintes strings: `“integer”, “double”, “string”, “array”, “object” e “unknown type”.`

Funções que testam o tipo da variável

São as funções `is_int, is_integer, is_real, is_long, is_float, is_string, is_array e is_object`. Todas têm o mesmo formato, seguindo modelo da assinatura a seguir:

```
int is_integer(mixed var);
```

Todas essas funções retornam `true` se a variável for daquele tipo, e `false` em caso contrário.

Destruindo uma variável

É possível desalocar uma variável se ela não for usada posteriormente através da função `unset`, que tem a seguinte assinatura:

```
int unset(mixed var);
```

A função destrói a variável, ou seja, libera a memória ocupada por ela, fazendo com que ela deixe de existir. Se mais na frente for feita uma chamada á variável, será criada uma nova variável de mesmo nome e de conteúdo vazio, a não ser que a chamada seja pela função `isset`. Se a operação for bem sucedida, retorna `true`.

Verificando se uma variável possui um valor

Existem dois tipos de teste que podem ser feitos para verificar se uma variável está setada: com a função `isset` e com a função `empty`.

A função isset

Possui o seguinte protótipo:

```
int isset(mixed var);
```

E retorna `true` se a variável estiver setada (ainda que com uma string vazia ou o valor zero), e `false` em caso contrário.

A função empty

Possui a seguinte assinatura:

```
int empty(mixed var);
```

E retorna `true` se a variável não contiver um valor (não estiver setada) ou possuir valor 0 (zero) ou uma string vazia. Caso contrário, retorna `false`.

10. Classes e Objetos

Classe

Uma classe é um conjunto de variáveis e funções relacionadas a essas variáveis. Uma vantagem da utilização é poder usufruir do recurso de encapsulamento de informação. Com o encapsulamento o usuário de uma classe não precisa saber como ela é implementada, bastando para a utilização conhecer a interface, ou seja, as funções disponíveis. Uma classe é um tipo, e portanto não pode ser atribuída a uma variável. Para definir uma classe, deve-se utilizar a seguinte sintaxe:

```
class Nome_da_classe {
   var $variavel1;
   var $variavel2;
   function funcao1 ($parametro) {
    /* === corpo da função === */
   }
}
```

Objeto

Como foi dito anteriormente, classes são tipos, e não podem ser atribuídas a variáveis. Variáveis do tipo de uma classe são chamadas de objetos, e devem ser criadas utilizando o operador `new`, seguindo o exemplo abaixo:

```
$variavel = new $nome_da_classe;
```

Para utilizar as funções definidas na classe, deve ser utilizado o operador "`->`", como no exemplo:

```
$variavel->funcao1(
```

A variável $this

Na definição de uma classe, pode-se utilizar a variável `$this`, que é o próprio objeto. Assim, quando uma classe é instanciada em um objeto, e uma função desse objeto na definição da classe utiliza a variável `$this`, essa variável significa o objeto que estamos utilizando.

Como exemplo da utilização de classes e objetos, podemos utilizar a classe conta, que define uma conta bancária bastante simples, com funções para ver saldo e fazer um crédito.

```
class conta {
   var $saldo;
   function saldo() {
     return $this->saldo;
   }
   function credito($valor) {
     $this->saldo += $valor;
```

```
   }
}

$minhaconta = new conta;
$minhaconta->saldo();     // a variavel interna não foi
                          // inicializada, e não contém
                          // valor algum
$minhaconta->credito(50);
$minhaconta->saldo(); // retorna 50
```

SubClasses

Uma classe pode ser uma extensão de outra. Isso significa que ela herdará todas as variáveis e funções da outra classe, e ainda terá as que forem adicionadas pelo programador. Em PHP não é permitido utilizar herança múltipla, ou seja, uma classe pode ser extensão de apenas uma outra.Para criar uma classe extendida, ou derivada de outra, deve ser utilizada a palavra reservada `extends`, como pode ser visto no exemplo seguinte:

```
class novaconta extends conta {
  var $numero;
  function numero() {
     return $this->numero;
  }
}
```

A classe acima é derivada da classe conta, tendo as mesmas funções e variáveis, com a adição da variável `$numero` e a função `numero()`.

Construtores

Um construtor é uma função definida na classe que é automaticamente chamada no momento em que a classe é instanciada (através do operador `new`). O construtor deve ter o mesmo nome que a classe a que pertence. Veja o exemplo:

```
class conta {
   var $saldo;

   function conta () {
     $this.saldo = 0;
   }

   function saldo() {
     return $this->saldo;
   }
   function credito($valor) {
     $this->saldo += $valor;
   }
}
```

Podemos perceber que a classe conta agora possui um construtor, que inicializa a variável `$saldo` com o valor 0.

Um construtor pode conter argumentos, que são opcionais, o que torna esta ferramenta mais poderosa. No exemplo acima, o construtor da classe conta pode receber como argumento um valor, que seria o valor inicial da conta.

Vale observar que para classes derivadas, o construtor da classe pai não é automaticamente herdado quando o construtor da classe derivada é chamado.

12. Conclusões

A realização deste Projeto Supervisionado possibilitou o estudo da linguagem PHP, que se mostrou uma ferramenta poderosa e simples de utilizar na construção de sites para a World Wide Web dinâmicos, possibilitando uma maior interação com o usuário e a armazenagem das informações em Bancos de Dados.

Após a conclusão da aplicação, tornou-se claro que a combinação de scripts *server-side*, como é o PHP, com scripts *client-side*, como JavaScript, por exemplo, possibilita um maior aproveitamento dos recursos disponíveis para criar páginas dinâmicas, e no processo de criação deve-se ponderar bastante para concluir qual dos dois tipos de scripts deve ser utilizado para determinado fim.

Entre as linguagens de script *server-side*, PHP surgiu como uma ótima opção, por diversos motivos: o custo de aquisição, que não existe; a portabilidade, permitindo que uma aplicação seja desenvolvida em uma plataforma para ser executada em outra; a simplicidade, já que os scripts ficam no próprio código html, e possuem uma sintaxe bastante simples; a possibilidade de trabalhar com diversos bancos de dados e servidores http, além do grande número de funções pré-definidas, entre outras coisas.

Por esses e outros motivos, é possível afirmar que o estudo sobre PHP foi bastante enriquecedor, por ter produzido uma documentação em português para a linguagem e ter motivado o aluno a continuar se dedicando ao tema.

13. Bibliografia e Referências

A pesquisa foi baseada no manual de PHP, disponível em ***www.php.net***, e em diversos tutoriais disponíveis no site ***www.phpbuilder.com***. Esses dois endereços contém uma vasta documentação sobre a linguagem, além de endereços para listas de discussão, onde pode-se solicitar ajuda de programadores mais experientes.

Uma boa referência em português é a lista *"PHP para quem fala Português"*, que pode ser assinada no endereço ***www.egroups.com/group/php-pt/***.

Em inglês, além dos endereços citados acima, uma boa fonte é o site *PHPWizard*, que pode ser encontrado em ***www.phpwizard.net***.

APÊNDICE 01 - Funções para tratamento de strings

Funções relacionadas a HTML

htmlspecialchars

```
string htmlspecialchars(string str);
```

Retorna a string fornecida, substituindo os seguintes caracteres:

- & para '&'
- " para '"'
- < para '<'
- > para '>'

htmlentities

```
string htmlentities(string str);
```

Funciona de maneira semelhante ao comando anterior, mas de maneira mais completa, pois converte todos os caracteres da string que possuem uma representação especial em html, como por exemplo:

- º para 'º'
- ª para 'ª'
- á para 'á'
- ç para 'ç'

nl2br

```
string nl2br(string str);
```

Retorna a string fornecida substituindo todas as quebras de linha ("\n") por quebras de linhas em html ("
").

```
Exemplo:
echo nl2br("Mauricio\nVivas\n");

Imprime:
Maurício<br>Vivas<br>
```

get_meta_tags

```
array get_meta_tags(string arquivo);
```

Abre um arquivo html e percorre o cabeçalho em busca de “meta” tags, retornando num array todos os valores encontrados.

```
Exemplo:
No arquivo teste.html temos:
...
<head>
<meta name="author" content="jose">
<meta name="tags" content="php3 documentation">
...
</head><!-- busca encerra aqui -->
...

a execução da função:

get_meta_tags(“teste.html”);

retorna o array:

array(“author”=>”jose”,”tags”=>"php3 documentation");
```

strip_tags

```
string strip_tags(string str);
```

Retorna a string fornecida, retirando todas as tags html e/ou PHP encontradas.

```
Exemplo:
strip_tags('<a href="teste1.php3">testando</a><br>');

Retorna a string “testando”
```

urlencode

```
string urlencode(string str);
```

Retorna a string fornecida, convertida para o formato urlencode. Esta função é útil para passar variáveis para uma próxima página.

urldecode

```
string urldecode(string str);
```

Funciona de maneira inversa a urlencode, desta vez decodificando a string fornecida do formato urlencode para texto normal.

Funções relacionadas a arrays

Implode e join

```
string implode(string separador, array partes);
string join(string separador, array partes);
```

As duas funções são idênticas. Retornam uma string contendo todos os elementos do array fornecido separados pela string também fornecida.

```
Exemplo:
$partes = array("a", "casa número", 13, "é azul");
$inteiro = join(" ",$partes);

$inteiro passa a conter a string:
“a casa número 13 é azul”
```

split

```
array split(string padrao, string str, int [limite]);
```

Retorna um array contendo partes da string fornecida separadas pelo padrão fornecido, podendo limitar o número de elementos do array.

```
Exemplo:
$data = “11/14/1975”;
$data_array = split(“/”,$data);

O código acima faz com que a variável $data_array receba o valor:
array(11,14,1975);
```

explode

```
array explode(string padrao, string str);
```

Funciona de maneira bastante semelhante à função `split`, com a diferença que não é possível estabelecer um limite para o número de elementos do array.

Comparações entre strings

similar_text

```
int similar_text(string str1, string str2, double [porcentagem]);
```

Compara as duas strings fornecidas e retorna o número de caracteres coincidentes. Opcionalmente pode ser fornecida uma variável, passada por referência *(ver tópico sobre funções)*, que receberá o valor percentual de igualdade entre as strings. Esta função é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

```
Exemplo:

$num = similar_text("teste", "testando",&$porc);

As variáveis passam a ter os seguintes valores:

$num == 4; $porc == 61.538461538462
```

strcasecmp

```
int strcasecmp(string str1, string str2);
```

Compara as duas strings e retorna 0 (zero) se forem iguais, um valor maior que zero se `str1 > str2`, e um valor menor que zero se `str1 < str2`. Esta função é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

strcmp

```
int strcasecmp(string str1, string str2);
```

Funciona de maneira semelhante à função `strcasecmp`, com a diferença que esta é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

strstr

```
string strstr(string str1, string str2);
string strchr(string str1, string str2);
```

As duas funções são idênticas. Procura a primeira ocorrência de `str2` em `str1`. Se não encontrar, retorna uma string vazia, e se encontrar retorna todos os caracteres de `str1` a partir desse ponto.

```
Exemplo:

strstr("Mauricio Vivas", "Viv"); // retorna “Vivas”
```

stristr

```
string strstr(string str1, string str2);
```

Funciona de maneira semelhante à função `strstr`, com a diferença que esta é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

strpos

```
int strpos(string str1, string str2, int [offset] );
```

Retorna a posição da primeira ocorrência de `str2` em `str1`, ou zero se não houver. O parâmetro opcional `offset` determina a partir de qual caracter de str1 será efetuada a busca. Mesmo utilizando o offset, o valor de retorno é referente ao início de str1.

strrpos

```
int strrpos(string haystack, char needle);
```

Retorna a posição da última ocorrência de `str2` em `str1`, ou zero se não houver.

Funções para edição de strings

chop

```
string chop(string str);
```

Retira espaços e linhas em branco do final da string fornecida.

```
Exemplo:
chop(“   Teste      \n  \n  “); // retorna “   Teste”
```

ltrim

```
string ltrim(string str);
```

Retira espaços e linhas em branco do final da string fornecida.

```
Exemplo:
ltrim(“   Teste  \n  \n “); // retorna “Teste  \n  \n”
```

trim

```
string trim(string str);
```

Retira espaços e linhas em branco do início e do final da string fornecida.

```
Exemplo:
trim("   Teste  \n  \n "); // retorna "Teste"
```

strrev

```
string strrev(string str);
```

Retorna a string fornecida invertida.

```
Exemplo:

strrev("Teste"); // retorna "etseT"
```

strtolower

```
string strtolower(string str);
```

Retorna a string fornecida com todas as letras minúsculas.

```
Exemplo:

strtolower("Teste"); // retorna "teste"
```

strtoupper

```
string strtoupper(string str);
```

Retorna a string fornecida com todas as letras maiúsculas.

```
Exemplo:

strtolower("Teste"); // retorna "TESTE"
```

ucfirst

```
string ucfirst(string str);
```

Retorna a string fornecida com o primeiro caracter convertido para letra maiúscula.

```
Exemplo:
ucfirst("teste de funcao"); // retorna "Teste de funcao"
```

ucwords

```
string ucwords(string str);
```

Retorna a string fornecida com todas as palavras iniciadas por letras maiúsculas.

```
Exemplo:
ucwords("teste de funcao"); // retorna "Teste De Funcao"
```

str_replace

```
string str_replace(string str1, string str2, string str3);
```

Altera todas as ocorrências de `str1` em `str3` pela string `str2`.

Funções diversas

chr

```
string chr(int ascii);
```

Retorna o caracter correspondente ao código ASCII fornecido.

ord

```
int ord(string string);
```

Retorna o código ASCII correspontente ao caracter fornecido.

echo

```
echo(string arg1, string [argn]... );
```

Imprime os argumentos fornecidos.

print

```
print(string arg);
```

Imprime o argumento fornecido.

strlen

```
int strlen(string str);
```

Retorna o tamanho da string fornecida.

APÊNDICE 02 - Funções para tratamento de arrays

Funções Genéricas

Array

```
array array(...);
```

É a função que cria um array a partir dos parâmetros forncidos. É possível fornecer o índice de cada elemento. Esse índice pode ser um valor de qualquer tipo, e não apenas de inteiro. Se o índice não for fornecido o PHP atribui um valor inteiro sequencial, a partir do 0 ou do último índice inteiro explicitado. Vejamos alguns exemplos:

```
Exemplo 1
$teste = array("um", "dois","tr"=>"tres",5=>"quatro","cinco");

Temos o seguinte mapeamento:
0 => “um” (0 é o primeiro índice, se não houver um explicito)
1 => “dois” (o inteiro seguinte)
“tr” => “tres”
5 => “quatro” (valor explicitado)
6 => “cinco” (o inteiro seguinte ao último atribuído, e não o próximo valor, que seria 2)

Exemplo 2
$teste = array("um", 6=>"dois","tr"=>"tres",5=>"quatro","cinco");

Temos o seguinte mapeamento:
0 => “um”
6 => “dois”
“tr” => tres
5 => “quatro” (seria 7, se não fosse explicitado)
7 => “cinco” (seria 6, se não estivesse ocupado)
```

Em geral, não é recomendável utilizar arrays com vários tipos de índices, já que isso pode confundir o programador. No caso de realmente haver a necessidade de utilizar esse recurso, deve-se ter bastante atenção ao manipular os índices do array.

range

```
array range(int minimo, int maximo);
```

A função range cria um array cujos elementos são os inteiros pertencentes ao intervalo fornecido, inclusive. Se o valor do primeiro parâmetro for maior do que o do segundo, a função retorna `false` (valor vazio).

shuffle

```
void shuffle(array &arr);
```

Esta função “embaralha” o array, ou seja, troca as posições dos elementos aleatoriamente e não retorna valor algum.

sizeof

```
int sizeof(array arr);
```

Retorna um valor inteiro contendo o número de elementos de um array. Se for utilizada com uma variável cujo valor não é do tipo array, retorna 1. Se a variável não estiver setada ou for um array vazio, retorna 0.

Funções de “navegação”

Toda variável do tipo array possui um ponteiro interno indicando o próximo elemento a ser acessado no caso de não ser especificado um índice. As funções seguintes servem para modificar esse ponteiro, permitindo assim percorrer um array para verificar seu conteúdo (chaves e elementos).

reset

```
mixed reset(array arr);
```

Seta o ponteiro interno para o primeiro elemento do array, e retorna o conteúdo desse elemento.

end

```
mixed end(array arr);
```

Seta o ponteiro interno para o último elemento do array, e retorna o conteúdo desse elemento.

next

```
mixed next(array arr);
```

Seta o ponteiro interno para o próximo elemento do array, e retorna o conteúdo desse elemento.

Obs.: esta não é uma boa função para determinar se um elemento é o último do array, pois pode retornar `false` tanto no final do array como no caso de haver um elemento vazio.

prev

```
mixed prev(array arr);
```

Seta o ponteiro interno para o elemento anterior do array, e retorna o conteúdo desse elemento. Funciona de maneira inversa a `next`.

pos

```
mixed pos(array arr);
```

Retorna o conteúdo do elemento atual do array, indicado pelo ponteiro interno.

key

```
mixed key(array arr);
```

Funciona de maneira bastante semelhante a `pos`, mas ao invés de retornar o elemento atual indicado pelo ponteiro interno do array, retorna seu índice.

each

```
array each(array arr);
```

Retorna um array contendo o índice e o elemento atual indicao pelo ponteiro interno do array. o valor de retorno é um array de quatro elementos, cujos índices são 0, 1, “key” e “value”. Os elementos de índices 0 e “key” armazenam o índice do valor atual, e os elementos de índices 1 e “value” contém o valor do elemento atual indicado pelo ponteiro.

Esta função pode ser utilizada para percorrer todos os elementos de um array e determinar se já foi encontrado o último elemento, pois no caso de haver um elemento vazio, a função não retornará o valor `false`. A função `each` só retorna `false` depois q o último elemento do array foi encontrado.

Exemplo:

```
/*função que percorre todos os elementos de um array e imprime seus índices e valores */
function imprime_array($arr) {
  reset($arr);
  while (list($chave,$valor) = each($arr))
    echo “Chave: $chave. Valor: $valor”;
}
```

Funções de ordenação

São funções que servem para arrumar os elementos de um array de acordo com determinados critérios. Estes critérios são: manutenção ou não da associação entre índices e elementos; ordenação por elementos ou por índices; função de comparação entre dois elementos.

sort

```
void sort(array &arr);
```

A função mais simples de ordenação de arrays. Ordena os elementos de um array em ordem crescente, sem manter os relacionamentos com os índices.

rsort

```
void rsort(array &arr);
```

Funciona de maneir ainversa à função `sort`. Ordena os elementos de um array em ordem decrescente, sem manter os relacionamentos com os índices.

asort

```
void asort(array &arr);
```

Tem o funcionamento bastante semelhante à função `sort`. Ordena os elementos de um array em ordem crescente, porém mantém os relacionamentos com os índices.

arsort

```
void arsort(array &arr);
```

Funciona de maneira inversa à função `asort`. Ordena os elementos de um array em ordem decrescente e mantém os relacionamentos dos elementos com os índices.

ksort

```
void ksort(array &arr);
```

Função de ordenação baseada nos índices. Ordena os elementos de um array de acordo com seus índices, em ordem crescente, mantendo os relacionamentos.

usort

```
void usort(array &arr, function compara);
```

Esta é uma função que utiliza outra função como parâmetro. Ordena os elementos de um array sem manter os relacionamentos com os índices, e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

uasort

```
void uasort(array &arr, function compara);
```

Esta função também utiliza outra função como parâmetro. Ordena os elementos de um array e mantém os relacionamentos com os índices, utilizando para efeito de comparação uma função definida pelo usuário, que deve comparar dois elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

uksort

```
void uksort(array &arr, function compara);
```

Esta função ordena o array através dos índices, mantendo os relacionamentos com os elementos., e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois índices do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

Sobre o autor da Apostila

Esta apostila é de autoria ***de*** **Maurício Vivas de Souza Barreto**
mauricio@cipsga.org.br **.....**

GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and

standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page.
For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto
adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

O. If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made

by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document. If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the
Document under the terms of section 4. Replacing Invariant Sections with translations requires especial permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents.
To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document
> under the terms of the GNU Free Documentation License, Version 1.1
> or any later version published by the Free Software Foundation;
> with the Invariant Sections being LIST THEIR TITLES, with the
> Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
> A copy of the license is included in the section entitled "GNU
> Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Comitê de Incentivo a Produção do Software Gratuito e Alternativo

Fundado em 29 de janeiro de 1999.

1ª Diretoria

Djalma Valois Filho
Diretor Executivo
dvalois@cxpostal.com

José Luiz Nunes Poyares
Diretor Administrativo

Paulo Roberto Ribeiro Guimarães
Diretor Institucional

CIPSGA
Rua Professora Ester de Melo, numero 202,
Parte, Benfica, Rio de Janeiro, RJ, CEP. 20930-010;
Telefone (Fax/Dados): 021-5564201;
e-mail: administracao@cipsga.org.br
CNPJ: 03179614-0001/70