ELABORATA

INFORMÁTICA

PHP

Curitiba / 2018

Srs. Alunos

A Elaborata Informática, com o objetivo de continuar prestando-lhe um excelente nível de atendimento e funcionalidade, informa a seguir algumas regras que devem ser observadas quando do uso do laboratório e seus equipamentos, visando mantê-los sempre em um perfeito estado de funcionamento para um melhor aproveitamento de suas aulas.

É proibido:

- Atender celular. Por favor, retire-se da sala, voltando assim que desligar.
- Fazer cópias ilegais de software (piratear), com quaisquer objetivos.
- Retirar da sala de treinamento quaisquer materiais, mesmo que a título de empréstimo.
- Divulgar ou informar produtos ou serviços de outras empresas sem autorização por escrito da direção Elaborata.
- Trazer para a sala de treinamento, qualquer tipo de equipamento pessoal de informática, como por exemplo:
  - Computadores de uso pessoal
  - Notebooks
  - Placas de vídeo
  - Placas de modem
  - Demais periféricos
  - Peças avulsas como memória RAM, ferramentas, etc.
- O consumo de alimentos ou bebidas
- Fumar

Atenciosamente

Elaborata Informática

Sumário


ELABORATA
INFORMÁTICA
CAPÍTULO 1
COMEÇANDO COM O PHP

PHP

1.1 A História do PHP

1.1.1 PHP/FI

O PHP sucede um produto mais antigo, chamado PHP/FI, criado em 1995 por Rasmus Lerdorf. Inicialmente fora usado como simples script Perl, para realizar estatísticas de acesso ao currículo online de Rasmus, que nomeou esta série de script como *Personal Home Page Tools*.

Uma vez que mais funcionalidades foram requeridas, Rasmus escreveu uma implementação C muito maior, capaz de se comunicar com uma base de dados, possibilitando aos usuários desenvolver aplicativos dinâmicos simples para web. Rasmus resolveu disponibilizar o código-fonte do PHP/FI para que todos pudessem ver e também usá-lo, bem como fixar *bugs* e melhorar o código.

PHP/FI, que significa *Personal Home Page/Forms Interpreter*, incluía algumas funcionalidades básicas do PHP, também conhecidas atualmente. Ele usava variáveis no estilo Perl, interpretação automática de variáveis vindas de formulários e sintaxe embutida no HTML. A sua própria sintaxe era similar a do Perl, entretanto muito mais limitada, simples e um pouco inconsistente.

1.1.2 PHP 3

Lançado em 1998, o PHP 3.0 foi a primeira versão, semelhante ao PHP que conhecemos hoje. Criada por Andi Gutmans e Zeev Suraski, em cooperação com Rasmus Lerdorf, a nova versão foi totalmente reescrita após descobrirem que o PHP/FI 2.0 poderia ajudá-los a desenvolver suas próprias aplicações de eCommerce em um projeto da Universidade.

Uma das maiores características do PHP 3.0 era sua forte capacidade de extensibilidade. Além de oferecer aos usuários finais uma infraestrutura sólida para diversos bancos de dados, protocolos e APIs, a extensibilidade do PHP 3.0 atraía dezenas de desenvolvedores a se juntarem e submeterem novos módulos. Esta é a chave do tremendo sucesso do PHP 3.0. Outras características introduzidas no PHP 3.0 foram o suporte à sintaxe para orientação a objetos e uma sintaxe muito mais poderosa e consistente.

Toda a nova versão da linguagem foi realizada sob um novo nome, que removeu a impressão do limitado uso pessoal que possuía o PHP/FI 2.0. Ela foi nomeada

simplesmente 'PHP', com o significado que é um acrônimo - PHP: *Hypertext Preprocessor*.

1.1.3 PHP 4

A versão 4 tinha como objetivos do projeto melhorar a performance de aplicações complexas e a modularidade do código-base do PHP. Tais aplicações foram possíveis por causa das novas características do PHP 3.0 e do suporte àa uma variedade de banco de dados de terceiros e APIs. Porém, a nova versão não foi projetada para trabalhar eficientemente com aplicações muito complexas.

A nova *engine* (de 'Zend Engine', que vem dos primeiros nomes, Zeev e Andi) fez desse objetivo um sucesso e foi introduzida em meados de 1999. O PHP 4, baseado nessa *engine* e acompanhado de uma série de novas características, foi oficialmente lançado em maio de 2000, quase dois anos após o seu predecessor, o PHP 3.0.

Além da relevante melhora da performance dessa versão, o PHP 4.0 incluiu outras características-chave, como o suporte para vários servidores Web, sessões HTTP, *buffer* de saída, maneiras mais seguras de manipular *input* de usuários e muitas construções novas na linguagem.

1.1.4 PHP 5

O PHP 5 inclui a nova Zend Engine 2.0, que é uma versão aprimorada da versão 4 do PHP. A principal diferença entre essas duas versões é que na 5 já existem maiores recursos e segurança, principalmente na Orientação a Objetos.

O PHP 5 foi lançado em julho de 2004, depois de um longo período de desenvolvimento.

A nova versão trouxe novos recursos, como suporte aprimorado, à orientação de objetos, a extensão PDO (PHP Data Objects) para fazer interface com bancos de dados e diferentes melhorias de desempenho.

O PHP 5 vem sendo melhorado desde então. A versão 5.4 já tem suporte nativo a UNICODE e teve diversos itens que comprometiam a segurança dos websites removidos, como o register_globals.

1.2 Benefícios do PHP

Podemos citar vários benefícios do trabalho com PHP. Falaremos sobre os principais, os que fazem a diferença na hora da escolha.

- **Custo:**

Para poder desenvolver em PHP, dispensa-se qualquer investimento em software, pois tanto o PHP quanto os softwares necessários são aqueles livre e gratuitos.

O PHP está licenciado como GPL – General Public License, a licença mais utilizada para software livre. Para maiores informações acesse: .www.php.net/license

- **Independente de sistema operacional:**

Executar o PHP em um computador independe do sistema operacional, pois os principais sistemas têm suporte a ele. Exemplos: Windows, Linux, Unix, Solaris, MAC.

- **Não necessita instalação no cliente:**

Não é necessário ter o PHP instalado em cada computador que irá acessar o website ou o sistema, mas somente no servidor que conterá os arquivos PHP.

- **Desempenho:**

Em testes de comparação de velocidade e performance, o PHP ganha com vantagem. Isso se deve ao fato de possuir seu núcleo escrito em linguagem C e ser melhor organizado.

- **Linguagem mais utilizada na Web:**

O PHP é a linguagem de programação mais utilizada para o desenvolvimento de websites e sistemas que funcionam pela internet.

- **Documentação abrangente em Português:**

Acesso à documentação em diversos idiomas no site do projeto: http://php.net/

1.3 PHP e HTML

Já sabemos que os navegadores web só entendem HTML. Isso não irá mudar. Mesmo com o uso do PHP os navegadores continuarão entendendo apenas HTML. Sendo assim, onde o PHP entra nessa história?

O PHP irá manipular o HTML no servidor e somente depois disso é que o enviará para o navegador.

Com isso, podemos depreender que o PHP deverá estar contido nos códigos HTML ou então gerar esses códigos para que os navegadores possam entender o que está sendo transmitido. Na maioria das vezes, o PHP está no meio do HTML ou são arquivos separados e o PHP manipula esses arquivos, para então enviar ao navegador o HTML manipulado.

1.4 O Ambiente do PHP

Para que o PHP funcione é indispensável que alguns *softwares* estejam instalados. Iremos conhecê-los a partir de agora.

1.4.1 Papel de cada componente envolvido no ambiente

1.4.1.1 Sistema Operacional

O sistema operacional é o requisito mínimo exigido, pois sem ele nenhum *software* pode ser instalado.

De acordo com o que já foi mencionado, vários sistemas operacionais possuem suporte ao PHP, como por exemplo: Linux, Windows, MAC, Unix e Solaris. No entanto, alguns deles necessitam de licença para que possam ser utilizados.

1.4.1.2 PHP

O interpretador PHP é o responsável pela execução do código-fonte, manipulação do HTML, acesso ao banco de dados, envio de e-mails, geração de PDF, entre outras ações. Sem ele não teremos a interpretação dos códigos PHP.

1.4.1.3 Servidor WEB

O servidor web é responsável por receber as requisições enviadas pelos navegadores ao servidor e acionar o PHP quando for preciso realizar a interpretação do mesmo. Além disso, envia o HTML manipulado novamente para o navegador.

O mais utilizado e aquele que apresenta melhor desempenho com o PHP é o Apache, que é livre e possui versões para vários sistemas operacionais. Mas não é o único servidor Web que pode ser usado, existem outros como o IIS, que só funciona quando instalado no Windows.

Outros servidores web:

- nginx
- GWS
- Resin
- Light HTTP Server (lighttpd)
- Sun Java System Web Server

1.4.1.4 Banco de dados

Banco de dados é o único componente opcional, pois consegue desenvolver sistemas em PHP sem a utilização de um deles. Porém, dificilmente será desenvolvido algo sem um banco de dados. É empregado para guardar informações do sistema, como por exemplo: usuários que podem ter acesso, notícias, clientes de uma empresa, etc.

Um dos bancos de dados mais usados com o PHP é o MySQL, que também é gratuito para a maioria dos casos, mas ele é seguido de perto pelo PostgreSQL.

Alguns bancos de dados suportados pelo PHP:

- MySQL
- PostgreSQL
- Oracle
- SQL Server
- dBase
- Informix
- DB2
- Progress
- Conexão ODBC
- SQLite

1.5 Introdução ao WAMP e LAMP

Para poder configurar um servidor para suportar PHP deve-se realizar a instalação dos componentes desse servidor, um a um. Sendo assim, faz-se necessário: instalar o servidor Web Apache (que está disponível para *download* no site oficial da Fundação Apache, www.apache.org), tanto para servidores Windows quanto Linux; fazer a instalação do interpretador PHP (disponível para *download* em www.php.net), tanto para servidores Windows quanto Linux e, finalmente; instalar o banco de dados (usaremos o MySQL como exemplo neste curso, que está disponível em www.mysql.com), tanto para servidores Windows quanto Linux.

Instalar e configurar os três componentes demanda um determinado tempo e, algumas vezes, paciência. Para evitar uma perda de tempo excessiva, utilizaremos o WAMP para realizar a instalação no Windows. Para fazer a instalação no Linux, bastará instalar os pacotes que constituem o LAMP.

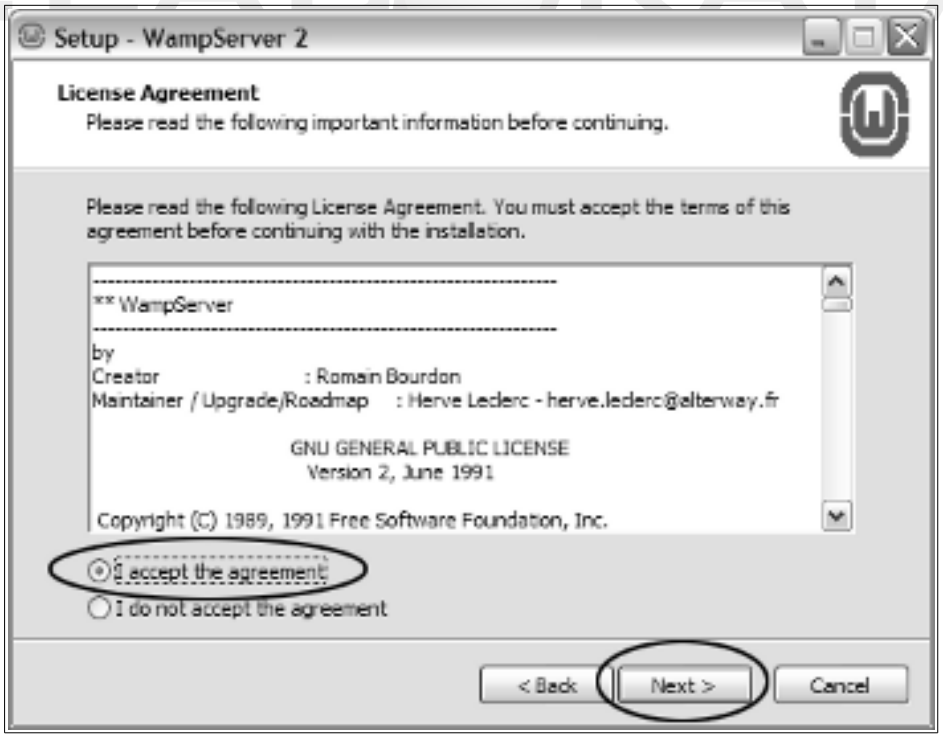
O nome WAMP é formado pelas iniciais de **W**indows, **A**pache, **M**ySQL e **P**HP. LAMP é formado pelas iniciais de **L**inux, **A**pache, **M**ySQL e **P**HP. Em síntese, são pacotes que fazem a instalação dos componentes do servidor de uma única vez, ao invés de instalar cada um separadamente.

1.5.1 Instalação do WAMP

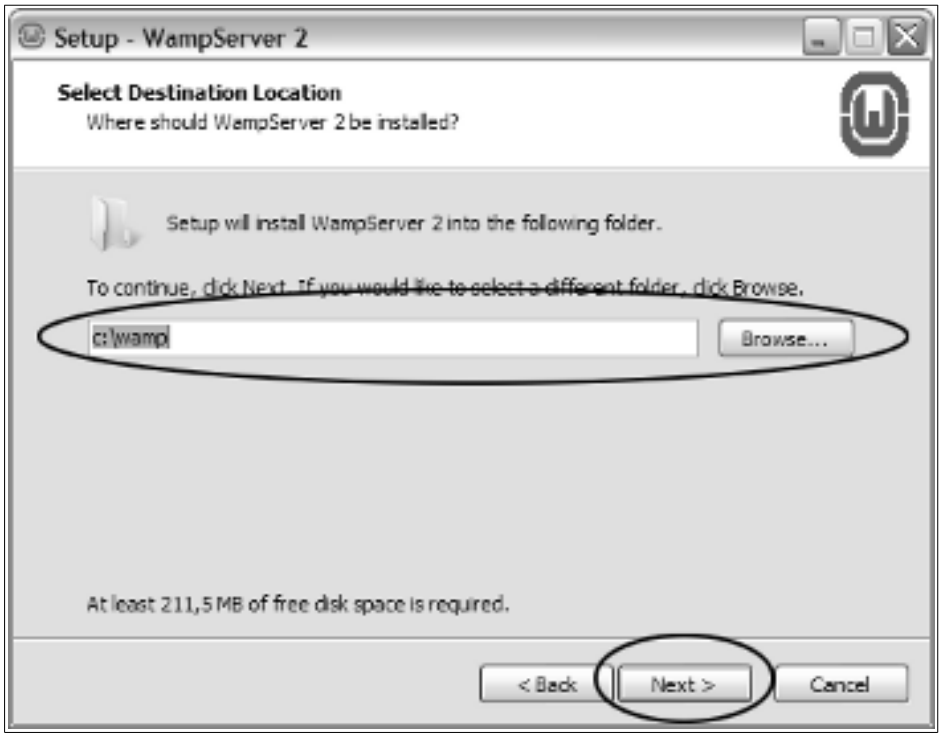
O primeiro passo para realizar a instalação é o *download* do WAMP. Pode ser obtido através do site www.wampserver.com. Na página inicial do site, entrar em *downloads* e efetuar o *download* da última versão disponível.

Na primeira tela, pressionar o botão **Next >**.

Na segunda tela, ler e marcar a opção de aceitação do acordo. Depois, clicar em **Next >**.



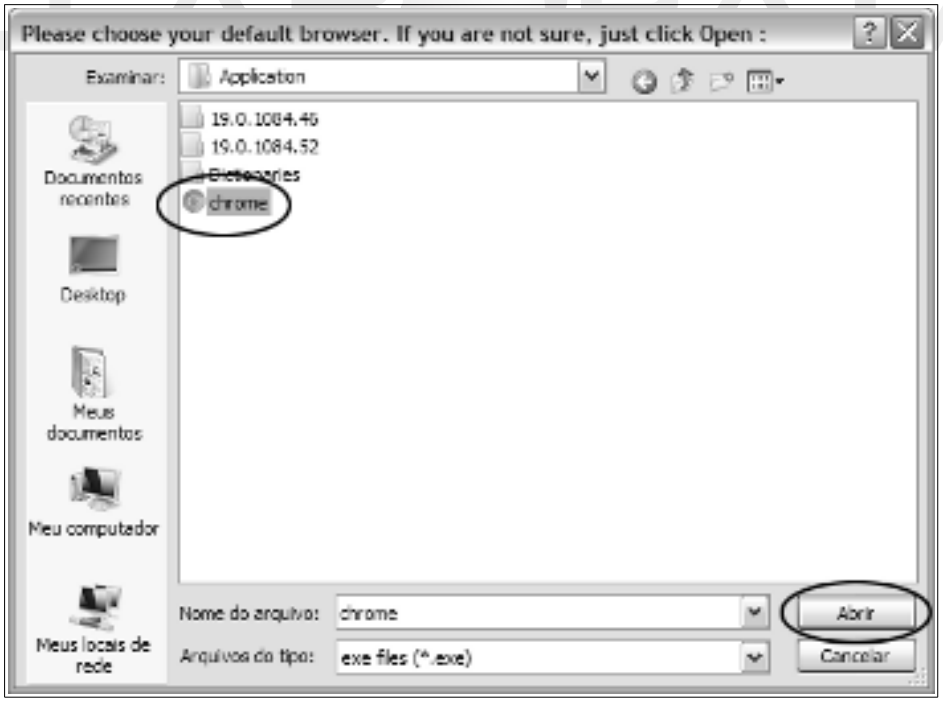
Na terceira janela, selecionar o local onde o WAMP será instalado. Se preferir, manter a sugestão que o instalador oferece, pressionando **Next>**.



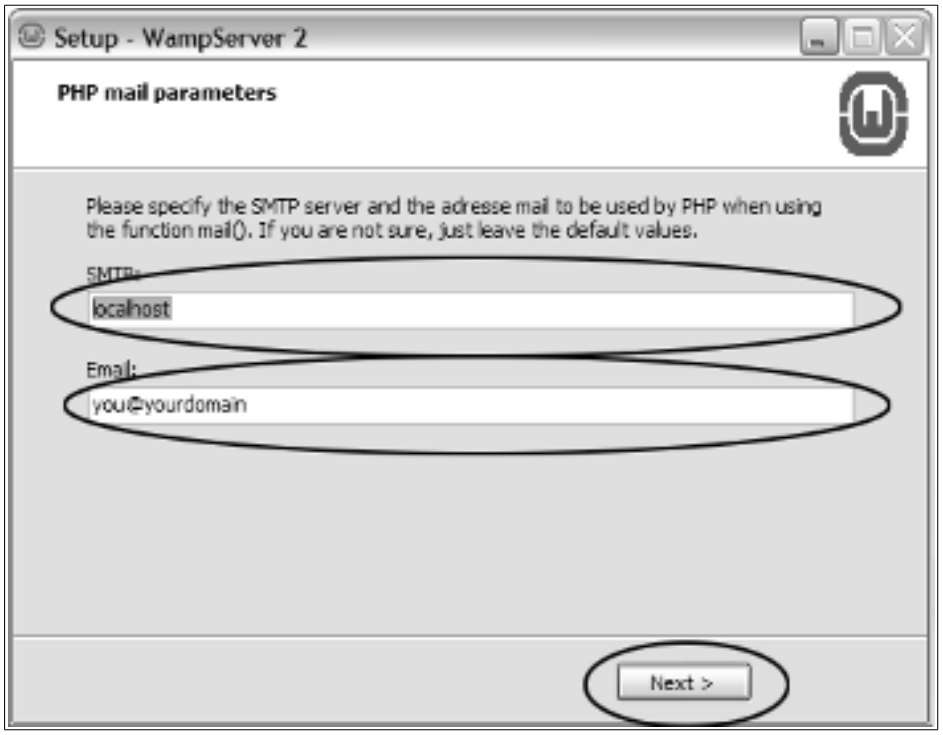
Na quarta tela, marcar as duas opções disponíveis e depois clicar em **Next >**.

Na quinta tela, pressionar o botão **Install**.

Na sexta tela deverá ser informado o navegador que utiliza, selecionando o executável dele.



Na sétima tela, podem ser mantidos os valores que estiverem sugeridos, mas caso queira, pode ser configurado um serviço de envio de e-mail da sua preferência.



Na oitava tela, pressionar **Finish** e a instalação do servidor estará concluída.

1.5.2 Instalação do LAMP

Para instalar o LAMP, emprega-se o console de comando do Linux. Será preciso instalar os três *softwares* separadamente, entretanto de forma fácil e rápida.

O exemplo que utilizaremos é para a distribuição Debian GNU/Linux.

Acessar o console, obedecendo os seguintes passos:

```
root@maquina:~# aptitude install apache2 php5 php5-mysql mysql-server
phpmyadmin
```

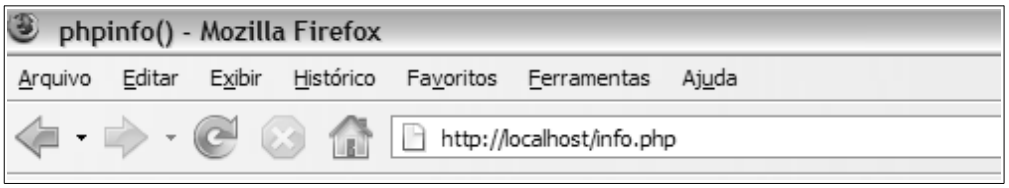
1.5.3 Acessando o servidor após a instalação

Todos os programas feitos em PHP devem estar em uma pasta específica no servidor. No caso do WAMP, é a pasta **www**, que está dentro daquela onde foi instalado o servidor.

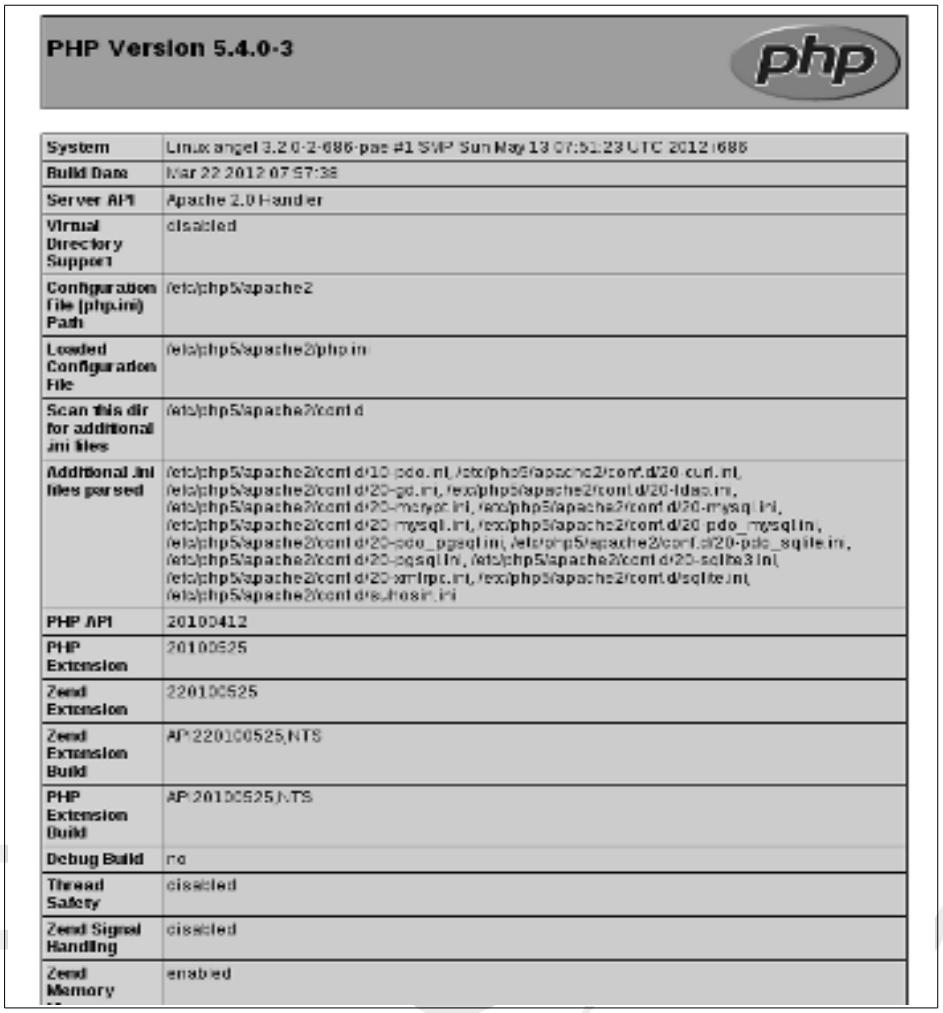
Para acessar os arquivos, utilizaremos um navegador como o Mozilla Firefox, por exemplo. Na barra de endereço deve ser informada a palavra **localhost**, ou **127.0.0.1**, ou o IP do computador, seguindo-se uma barra (/) e o nome do arquivo.

Como exemplo, iremos criar um arquivo chamado **info.php**. Nele usaremos a função **phpinfo()**, esta indicará que a instalação foi bem sucedida e que o servidor estará habilitado para ser utilizado.

```
<?php
      phpinfo();
?>
```



Ao acessar a página **info.php**, observar-se-á exatamente conforme mostrado abaixo:



PHP Version 5.4.0-3

System	Linux angel 3.2.0-2-686-pae #1 SMP Sun May 13 07:51:23 UTC 2012 i686
Build Date	Mar 22 2012 07:57:38
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-curl.ini, /etc/php5/apache2/conf.d/20-gd.ini, /etc/php5/apache2/conf.d/20-ldap.ini, /etc/php5/apache2/conf.d/20-mcrypt.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini, /etc/php5/apache2/conf.d/20-pdo_pgsql.ini, /etc/php5/apache2/conf.d/20-pdo_sqlite.ini, /etc/php5/apache2/conf.d/20-pgsql.ini, /etc/php5/apache2/conf.d/20-sqlite3.ini, /etc/php5/apache2/conf.d/20-xmlrpc.ini, /etc/php5/apache2/conf.d/sqlite.ini, /etc/php5/apache2/conf.d/suhosin.ini
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525
Zend Extension Build	API220100525,NTS
PHP Extension Build	API20100525,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory	enabled

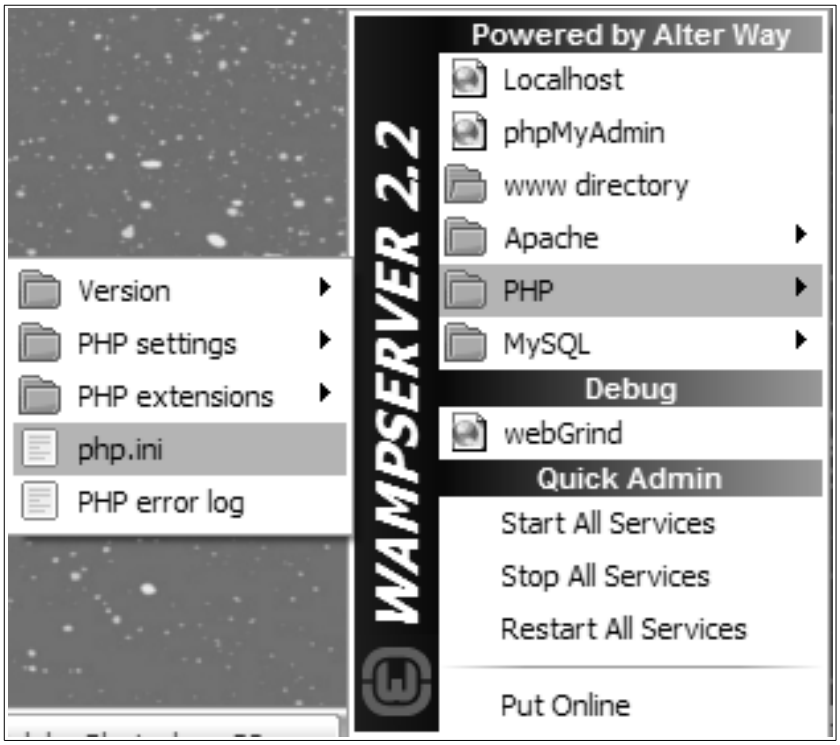
Nela poderemos ver a versão do PHP instalada e algumas informações do sistema operacional. Se continuarmos observando a página, perceberemos tudo o que está habilitado para ser usado no servidor PHP.

1.5.4 PHP.INI

Php.ini é o arquivo de configuração do PHP. Todo e qualquer módulo que se queira utilizar deve estar habilitado nele.

No **php.ini** poderemos habilitar o banco de dados que utilizaremos, a opção por trabalhar, ou não, com a geração de gráficos, se serão informados erros de sintaxe nos comandos dos programas feitos em PHP, e quais os tipos de erros serão mostrados.

Para acessá-lo, basta clicar em seu ícone no Wamp, que está localizado na bandeja ao lado do relógio: vá até a opção 'PHP' e, em seguida, clique em **php.ini**.



Ao clicar no **php.ini**, o arquivo de configuração será aberto no Bloco de Notas.

Vejamos como configurar a exibição dos erros gerados pelos programas.

	Nota: As linhas que contêm ponto e vírgula em seu início estão comentadas e não serão habilitadas pelo PHP.

1.5.4.1 display_errors

A diretiva **display_errors** é responsável pela exibição ou não dos erros.

Procuraremos no **php.ini** por 'display_errors'. A opção que não tiver ponto e vírgula no início da linha será aquela válida para o interpretador PHP. Para que seja feita a exibição de erros ela deve estar 'on'.

```
display_errors = on
```

1.5.4.2 error_reporting

A diretiva **error_reporting** é responsável por configurar quais erros serão exibidos.

Buscaremos essa diretiva no **php.ini**. Por padrão, vem configurada para exibir todos os erros, exceto os *notices*, que são gerados a partir de possíveis problemas que

podem acontecer no programa, como por exemplo, a tentativa de utilizar uma variável que não existe.

```
error_reporting = E_ALL & ~E_NOTICE
```

Essa é a configuração padrão e a mais utilizada, contudo existem várias outras opções. Para verificar quais seriam elas, ver a lista que se encontra um pouco acima da diretiva **error_reporting**.

1.6 Programas Cliente ou Servidor

Basicamente existem dois tipos de programas: o que funciona no cliente e o que funciona no servidor.

O PHP é um programa que funciona no servidor, ou seja, todo o processamento é feito em um servidor e enviado para quem solicitou a execução do programa que, no caso do PHP, é qualquer navegador de internet.

1.6.1 Lado cliente

Programas que funcionam do lado cliente também são conhecidos como *client-side*. Esses programas funcionam localmente, no computador de quem está utilizando o sistema. Um bom exemplo de um programa *client-side* são as nossas ferramentas de escritório. Elas estão instaladas em cada computador que precise utilizá-las.

Os programas *client-side* consomem recursos de *hardware* do computador que os está usando e, na maioria das vezes, não depende de outro computador para funcionar.

O exemplo mais fácil de entender é o navegador de internet. Ele é um cliente que faz solicitações ao servidor e aguarda a resposta; assim que a recebe, mostra o conteúdo solicitado.

1.6.2 Lado servidor

Programas que funcionam em um servidor também são conhecidos como *server-side*. Esses programas concentram toda a funcionalidade em um servidor, sendo assim, os recursos de *hardware* são consumidos no servidor e não mais em quem está utilizando o sistema.

Um bom exemplo de programas *server-side* são os websites. Eles sempre ficarão no servidor e qualquer pessoa que tenha acesso a ele poderá executar os programas. Porém, para acessar, deve ser utilizado um *software* que precisa estar instalado no computador de quem está acessando. No caso de um website, esse *software* é o navegador de internet, como por exemplo, o Firefox, o Chrome ou o Internet Explorer.

Então, para acessar um programa *server-side* sempre haverá um *software* fazendo o papel do cliente e um servidor.

Como funciona o acesso a um website:

Para acessar o site da Elaborata temos que abrir o navegador e digitar na barra de endereço: www.elaborata.com.br. Feito isso, o navegador irá disparar uma requisição na internet direcionada ao servidor que estiver configurado para o site da Elaborata. No servidor haverá um servidor Web instalado, no caso, o Apache, que receberá essa solicitação e irá procurar a página requerida. Se essa página for feita em PHP, o Apache acionará o interpretador PHP que processará o código-fonte e devolverá ao Apache somente códigos HTML. O Apache, então, se encarrega de enviar esse código HTML para o navegador que solicitou.

Nesse processo foi possível observar o servidor e o cliente (navegador).

1.7 Escrevendo o Primeiro Programa em PHP

Veremos agora um exemplo de código PHP.

Trata-se de um código bem simples, cuja função será escrever, “esse é o meu primeiro exemplo no curso de PHP”. A sua sintaxe será explicada mais adiante. No momento, esse exemplo é somente para entender como funciona o PHP.

```
<!DOCTYPE html>
<html lang=”pt-br”>
<head>
<meta charset=”utf-8" />
<title>Meu primeiro exemplo</title>
</head>
<body>
<?php
```

```
      echo “Esse é o meu primeiro exemplo no curso de PHP”;
?>
</body>
</html>
```

Obs.: *Veja que o código PHP está dentro do código HTML.*

1.8 Exercícios

1. Fazer a instalação do WAMP.
2. Usar como base o exemplo acima e escrever seu nome completo, idade e e-mail na tela.

Nota: *O PHP disponibiliza a sua documentação em português no site oficial, www.php.net. Nele você irá encontrar para* download *a documentação e um guia on-line de todas as suas funções.*

CAPÍTULO 2

CONHECENDO O PHP

2.1 Nomenclatura de Arquivos

Para que se possa desenvolver um bom programa deve existir um padrão a ser seguido. Esse padrão deverá contemplar o nome de arquivos, o nome de variáveis, o nome de funções, entre outros.

Veremos, então, um padrão para nomenclatura de arquivos, o que irá auxiliar muito no momento de localizar um, entre os vários arquivos, existentes dentro de um programa.

1. Nunca utilizar letras maiúsculas, nem espaços em branco e, muito menos, caracteres especiais, como alguma acentuação.
2. Sempre iniciar o nome do arquivo por uma palavra que faça menção ao assunto do qual ele trata e depois a ação que ele venha a ter. Por exemplo, vamos criar um arquivo que contenha o formulário de cadastro de um usuário. O nome do arquivo seria: usuarios_cadastrar.php ou usuarioscadastrar.php.
3. Identificar quando é um arquivo de classes. Exemplo: criar um arquivo que contenha a classe de conexão com o banco de dados: conexao.class.php.
4. Sempre terminar um arquivo com a extensão PHP, pois assim poderemos evitar falhas de segurança. Por exemplo, se criarmos o arquivo de conexão com o banco de dados chamado conexao.class e alguém souber o caminho para acessar esse arquivo através da URL do navegador, ele conseguirá visualizar todo o conteúdo do arquivo, tendo, então, acesso ao host do usuário e senha de acesso ao banco de dados. Isso aconteceria porque se o arquivo não está com a extensão .php não será interpretado pelo servidor e sim mostrado diretamente no navegador.

2.2 A Sintaxe do PHP

Assim como toda linguagem de programação, o PHP possui a sua sintaxe, que envolve, desde *tags* para indicar início e fim de códigos PHP, até variáveis, constantes, estruturas de controle, estruturas de repetição, etc.

2.2.1 Delimitador de códigos PHP

Para delimitar o código PHP usaremos a seguinte marcação:

```
<?php
      comandos PHP
?>
```

Em versões anteriores eram permitidos outros tipos de delimitadores, mas caíram em desuso. Abaixo poderá verificar alguns dos antigos delimitadores. É útil saber do que se trata, para que possa reconhecê-lo se encontrar um deles em algum código PHP antigo.

```
<?
      comandos PHP
?>
<%
      comandos PHP
%>
<script language="php">
      comandos PHP
</script>
```

2.2.2 Delimitador de comandos PHP

Para informar o final de um comando, é indispensável o uso do caractere ';' (ponto e vírgula).

Uma vez que o delimitador de comandos é esse, então poderemos encontrar vários comandos em uma mesma linha, mas isso não é recomendado, por causa da organização visual do seu código-fonte.

Exemplo:

```
<?php
      comando1;
      comando2; comando3;
      comando4
?>
```

Repare que o **comando1** está sozinho em uma linha e o ponto e vírgula no fim dessa linha, indicando o final do comando. O **comando2** e o **comando3** estão na mesma linha, mas mesmo assim, continuam tendo o ponto e vírgula. O **comando4** está

sozinho na última linha e sem o ponto e vírgula no final, isso porque ele é a última instrução no bloco de códigos.

Isto pode não representar um problema, à primeira vista, mas as chances do seu código crescer são grandes e, se não for colocado o ponto e vírgula, o indivíduo que estiver alterando o código provavelmente não irá perceber que esta linha está sem o delimitador , causando um erro e um bom tempo pode ser gasto para encontrá-lo e resolvê-lo.

2.2.3 Comentários

Comentários são partes de códigos não interpretadas pelo PHP. Sendo assim, poderemos descrever o nosso código sem que ocorra algum erro de sintaxe, ou simplesmente, é possível comentar um bloco de códigos para que ele não seja executado.

Existem dois tipos de comentário, o de linha e o de bloco.

2.2.3.1 Comentários de linha

Para comentarmos apenas uma linha ou parte dela poderemos utilizar os caracteres '#' ou '//'.

```
<?php
      //comentando uma linha
      echo “Olá, mundo”; //Comentando parte de uma linha

      #comentando uma linha
      echo “Olá, mundo”; #comentando parte de uma linha
?>
```

2.2.3.2 Comentários de bloco

Para comentarmos um bloco de código deveremos colocar no início do bloco a ser comentado os caracteres '/*' e, ao final do bloco, precisaremos colocar os caracteres '*/', conforme se verifica a seguir.

```
<?php
      /* comentando mais de
         uma linha /*
```

```
      echo “Olá, mundo”;
?>
```

Se quisermos um efeito mais organizado no seu comentário de bloco, experimente colocar os caracteres da seguinte forma:

```
<?php
      /*
       * comentando mais de uma linha
       */
      echo “Olá, mundo”;
?>
```

2.2.4 Variáveis

Antes de aprendermos como criar e empregar variáveis no PHP, devemos ter em mente que o PHP é uma linguagem fracamente tipada, ou seja, não é preciso informar o tipo de dado que a variável irá conter. O interpretador se encarrega disso, o que deixa mais fácil a utilização do PHP.

Toda variável do PHP é iniciada pelo caractere '$' (cifrão). O nome das variáveis é *case sensitive*, isto é, serão diferenciados nomes em letras maiúsculas de nomes em letras minúsculas.

Quando existe a necessidade de utilizar uma variável, ela não precisa ser declarada, basta incluí-la diretamente.

```
<?php
      $cidade = “Curitiba”;
?>
```

No exemplo acima criou-se uma variável, definindo o seu valor como sendo “Curitiba”.

Obs.: *O primeiro caractere depois do cifrão no nome da variável nunca poderá ser um número ou um espaço em branco. Espaços em branco e caracter es especiais não são permitidos em nenhum local no nome das variáveis.*

Existem, no PHP, variáveis especiais chamadas SUPERGLOBAIS.

Na maioria dos casos o PHP irá criá-las automaticamente, mas existem situações em que elas precisam ser estabelecidas pelo desenvolvedor.

É fácil identificar essas variáveis, pois sempre irão apresentar o seguinte padrão: o primeiro caractere sempre será o '$' (cifrão), seguido de um '_' (*underline)* e, depois, o nome da variável superglobal terá todas as letras maiúsculas.

```
<?php
      $_POST['nome'];
?>
```

No exemplo anterior foi empregada a variável superglobal para recuperar o valor de um campo enviado através de um formulário.

Lista das variáveis superglobais mais utilizadas:

Variável	Descrição
$_POST	A variável superglobal $_POST é criada sempre que existir o envio de dados através de formulários onde o método de envio é o POST. Ela é um vetor e para acessar seus valores é preciso informar o índice, que são os nomes dos campos do formulário que foi enviado.
$_GET	A variável superglobal $_GET é criada em duas situações: a primeira delas é quando existe o envio de dados através de formulários pelo método GET; a segunda é quando são enviados parâmetros via URL.
$_REQUEST	A variável superglobal $_REQUEST é criada sempre que existir algum tipo de dado, pode ser através de formulário, pelos métodos POST e GET, ou através da URL.
$_FILES	A variável superglobal $_FILES é criada sempre que é feito o *upload* de arquivos.

2.2.5 Tipos de dados suportados

O PHP, assim como todas as linguagens de programação, tem seus tipos de dados suportados. Veremos, então, as características de cada um.

- String:

É uma sequência ordenada de caracteres (símbolos) escolhidos a partir de um conjunto predeterminado.

```
<?php
      $treinamento = “PHP”;
?>
```

- Inteiro:

São constituídos de números naturais {0, 1, 2, ...} e dos seus opostos {0, -1, -2, ...}.

```
<?php
      $idade = 21;
      $temperatura = -3;
?>
```

Note que os valores não estão entre aspas, isso porque valores numéricos não necessitam da presença de aspas.

- Ponto flutuante:
- Objeto:

  Um objeto representa uma entidade que pode ser física, conceitual ou de *software*. É uma abstração de algo que possui fronteira definida e significado para a aplicação.

É um formato de representação digital de números reais.

```
<?php
      $valor = 127.90;
      $nota = 9.7;
?>
```

Observe que os valores não estão entre aspas, já que valores numéricos não necessitam da presença de aspas.

Repare também que o caractere que faz a divisão dos números inteiros com os decimais é o ponto (.).

- Array:

Um array, também conhecido como 'vetor' ou 'lista' (para arrays unidimensionais) ou 'matriz' (para arrays bidimensionais), é uma das mais simples estruturas de dados.

```
<?php
      $diasSemana = array("Domingo", "Segunda-feira", "Terça-feira");
      //ou
      $diaSemana[] = "Domingo";
      $diaSemana[] = "Segunda-feira";
      $diaSemana[] = "Terça-feira";
?>
```

- Objeto:

Um objeto representa uma entidade que pode ser física, conceitual ou de software. É uma abstração de algo que possui fronteira definida e significado para a aplicação.

```
<?php
      $banco = new conexao();
?>
```

2.2.6 Constantes

Uma constante, diferentemente de variáveis, não possui nenhum caractere inicial para identificá-la. Para ser criada teremos que utilizar a função **define** e, para usar a constante basta escrever o seu nome.

Sintaxe da função **define**.

```
define("<nome da constante>", "<valor>");
```

```
<?php
      //definindo a constante
      define("curso", "PHP");
      //utilizando a constante
      echo curso;
?>
```

2.2.7 Operadores

2.2.7.1 String

Para *strings* não existem muitas opções de operadores, na verdade são apenas duas: a concatenação e a atribuição de concatenação.

Para concatenar basta colocar um '.' (ponto) entre duas *strings*

```
<?php
      echo "Aluno"." ; ".$nome_aluno;
?>
```

No exemplo acima estamos concatenando o texto 'aluno' com um sinal de dois pontos e com o conteúdo da variável $nome_aluno.

Para fazer uma atribuição de concatenação basta colocar um '.' (ponto), seguido de um sinal de '=' (igual).

```
<?php
      $texto  = “Aluno”;
      $texto .= $nome_aluno;
?>
```

2.2.7.2 Aritméticos

Operadores aritméticos são empregados quando necessitarmos realizar cálculos.

Operador	Nome	Exemplo
+	Adição	<?php echo 1 + 1; ?>
-	Subtração	<?php echo 2 - 1; ?>
/	Divisão	<?php echo 15 / 3; ?>
*	Multiplicação	<?php echo 10 * 15; ?>
%	Módulo	<?php echo 10 % 2; ?>

2.2.7.3 Comparação

Operadores de comparação serão utilizados quando for essencial realizar a comparação entre dois valores.

Operador	Nome	Exemplo	Descrição

==	Igual	If($x == $y){
		#comandos
		}
			Compara se o valor de $x é igual ao valor de $y. Com esse operador só é comparado o valor da variável e não o tipo do dado.

===	Exatamente igual(idêntico)	if($x === $y){
		#comandos
		}
			Compara se o valor de $x é idêntico ao valor de $y. Com esse operador sempre serão comparados o tipo de

			dado e o valor.

!= ou <>	Diferente	if($x != $y)
		#comandos
		}
			Compara se o valor de $x é diferente de $y.
			Com esse tipo de operador apenas o valor será comparado.

!==	Não idêntico	if($x !=== $y){
		#comandos
		}
			Verifica se o valor e o tipo de dado de $x são diferentes de $y.

<	Menor que	if($x < $y){
		#comandos
		}
			Verifica se o valor de $x é menor que o valor de $y.

>	Maior que	if($x > $y){
		#comandos
		}
			Verifica se o valor de $x é maior que o valor de $y.

<=	Menor ou igual a	if($x <= $y){
		#comandos
		}
			Verifica se o valor de $x é menor ou igual ao valor de $y.

>=	Maior ou igual a	if($x >= $y){
		#comandos
		}
			Verifica se o valor de $x é menor ou igual ao valor de $y.

2.2.7.4 Controle de erro

O PHP possui um operador que faz o controle de erros das aplicações. Ao utilizá-lo poderemos ocultar erros que não irão interferir no desenvolvimento do programa. Ele só não funcionará quando se tratar de um erro de sintaxe.

Operador	Nome	Exemplo
@	Arroba	@session_start(); @unlink("../imagem.jpg");

2.2.7.5 Incremento e decremento

O PHP possui operadores que incrementam e decrementam valores.

Operador	Nome	Exemplo	Descrição
$x++	Pós-incremento	$x = 1; echo $x++; #1 echo $x; #2	Retorna o valor e depois faz o incremento. Por isso, quando usamos **echo $x++**

			ele imprime o valor **1**, já que o incremento ainda não aconteceu.
++$x	Pré-incremento	$x = 1; echo ++$x; #2 echo $x; #2	Faz o incremento e depois retorna o valor.
$x--	Pós-decremento	$x = 1; echo $x--; #1 echo $x; #0	Retorna o valor e depois faz o decremento. Quando usamos **echo $x--** ele imprime o valor **1**, pois o decremento ainda não aconteceu.
--$x	Pré-decremento	$x = 1; echo --$x; #0 echo $x; #0	Faz o decremento e depois retorna o valor.

2.2.7.6 Lógicos

Existem também os operadores lógicos. São eles:

Operador	Nome	Exemplo	Descrição

&& OU AND	E	if($x == $y && $a != $b){
		#comandos
		}
			Apenas executará os comandos caso $x seja igual a $y E $a seja diferente de $b.

|| OU OR	OU	if($x == $y || $a != $b){
		#comandos
		}
			Apenas executará os comandos caso $x seja igual a $y OU $a seja diferente de $b.

!	Não	if(!(empty($a))){ #comandos	A função **empty** é utilizada para verificar se a variável

		} 	está vazia. Se inserirmos um '!' na frente, estaremos negando a expressão e ele passa a assumir que $a não está vazia.

XOR	XOR	if($x == $y XOR $a != $b){
		#comandos
		}
			Executará os comandos se $x for igual a $y OU $a for diferente de $b, mas somente se uma ou outra for verdadeira. Caso ambas sejam verdadeiras não serão executados os comandos.

2.2.8 Conversão do tipo de variáveis

O PHP não é uma linguagem tipada, ou seja, não é necessário informar o tipo de dado que uma variável irá conter. Mesmo assim, é possível forçar o tipo de dado que uma variável deve armazenar.

Poderemos definir o tipo das variáveis como:

- boolean
- integer
- float
- string
- array
- object
- null

Para setar o tipo de dado de uma variável, utiliza-se a função **settype** com a seguinte sintaxe:

```
<?php
	$x = “5bar”;
	$y = true;
```

```
      settype($x, “integer”)
      settype($y, “string”);
      echo $x;
      echo $y
?>
```

Inicialmente, o tipo de dado da variável $x era uma *string* com o valor 5bar. E a variável $y era um *boolean* com valor *true*.

Porém, ao ser executada a função **settype**, mudaremos de *string* para *integer* e de *boolean* para *string*. Ao imprimir as variáveis poderá observar as mudanças.

Obs.: *Caso não seja possível fazer a mudança do tipo de dado da variável, ela continuará com o mesmo tipo inicial.*

2.2.9 Estruturas de controle e repetição

2.2.9.1 IF

O **if**, sem dúvida, é a estrutura de controle mais usada e mais importante não só no PHP, mas em praticamente todas as linguagens de programação. Com ele poderemos fazer com que blocos de códigos sejam ou não executados, dependendo da sua condição.

Para demarcar o que será executado dentro de um **if**, usaremos chaves '{}' e delimitaremos o seu escopo.

Sintaxe:

```
if(condição){
      comandos
}
```

Exemplo:

```
<?php
      $curso = “PHP“;
      if($curso == “PHP”)
      {
            echo “você está no curso de PHP”;
      }
```

```
?>
```

Nota: *Um único* ***if*** *pode ser utilizado com quantas condições forem necessárias, basta apenas aplicar os operadores lógicos.*

2.2.9.2 ELSE

O **else** sempre será empregado juntamente com o **if**. Ele se encarrega de executar em bloco de comandos, caso a condição do **if** não seja satisfeita.

```
<?php
      $curso = “PHP“;
      if($curso == “PHP”)
      {
            echo “você está no curso de PHP”;
      }
      else
      {
            echo “Você não está no curso de PHP”;
      }
?>
```

Obs.: *Quando utilizarmos o* ***if...else*** *só poderemos ter uma condição e uma alternativa para o caso de a condição ser falsa. Para fazer várias condições, verifique o* ***elseif****.*

2.2.9.3 ELSEIF

O **elseif**, da mesma forma que o **else**, sempre será utilizado como um complemento do **if**.

Poderão existir casos em que seja preciso realizar várias verificações para saber qual bloco de comandos será executado.

```
<?php
      $curso = “PHP“;
      if($curso == “PHP”)
      {
            echo “você está no curso de PHP”;
```

```
	}
	elseif($curso == “MySQL”)
	{
		echo “Você está no curso de MySQL”;
	}
	elseif($curso == “Ajax”)
	{
		echo “Você está no curso de Ajax”;
	}
	else
	{
		echo “Você não está no curso de PHP, nem no curso de MySQL e nem no curso de Ajax”;
	}
?>
```

Obs.: *Veja que utilizamos também o* ***else****. Ele poderá ser usado sem problemas junto com o* ***if*** *e o* ***elseif****, porém sempre será a última condição.*

2.2.9.4 SWITCH

A estrutura **switch** é semelhante a do **if/elseif/else**. Serve para realizar verificações no momento de saber qual bloco de comandos executar.

```
<?php
	$curso = “PHP”;
	switch($curso)
	{
		case “PHP”:
			echo “você está no curso de PHP”;
			break;
		case “MySQL”:
			echo “você está no curso de MySQL”;
			break;
		case “Ajax”:
			echo “você está no curso de Ajax”;
```

```
                break;
            default:
                echo “Você não está no curso de PHP, nem no curso de MySQL e
nem no curso de Ajax”;

    }
?>
```

switch($curso) recebe o valor que deverá ser comparado mais adiante. A verificação do valor é feita pelo **case “VALOR”:**. Se o valor da variável que foi informada no **switch** for igual ao valor do **case**, ele irá executar as opções dentro do **case**.

Observaremos também que dentro de cada **case** há o **break**; ele serve para parar a execução do **switch**, pois se ela não for interrompida, o PHP continuará a executar o conteúdo dos **cases** a seguir, contudo só executará o conteúdo sem fazer a verificação no **case**.

Portanto, nunca esquecer de inserir o **break** depois de executar tudo o que for necessário no **case** em questão.

2.2.9.5 FOR

Empregaremos o **for** para fazer laços com tamanho finito, isto é, quando se conhece o momento em que o laço irá parar.

Para demarcar o que será executado dentro de um **for** utilizam-se chaves '{}', delimitando-se o seu escopo.

A sintaxe é:

```
for(valor inicial; condição; incremento ou decremento)
{
    #comandos
}
```

Vamos fazer um laço que imprima na tela os valores de 0 até 10.

```
for($i = 0; $i <= 10; $i++)
{
    echo $i.“<br />”;
```

```
}
```

Obs.: *O* ***for*** *pode não ser executado nenhuma vez, dependendo do valor inicial e da condição utilizados. Por exemplo, se utilizássemos a variável $i iniciando em 500 ele nunca iria executar, pois 500 é maior do que 10.*

2.2.9.6 WHILE

Assim como o **for**, usamos o **while** para fazer um laço, contudo um laço do qual não sabemos o momento cuja execução cessará. Isso porque ele irá manter a execução do laço enquanto a condição for verdadeira.

A sintaxe é:

```
while(condição)
{
      #comandos
}
```

Vamos fazer um laço enquanto a variável $i for menor ou igual a 100.

```
$i = 0;
while($i <= 100)
{
      echo $i."<br />";
      $i++;
}
```

Obs.: *O* ***while*** *pode não ser executado nenhuma vez, dependendo da condição utilizada. Se utilizássemos a variável $i iniciando de 500 ele não executaria, pois 500 é maior do que 100.*

2.2.9.7 DO WHILE

O **do while**, assim como o **for** e o **while**, é empregado para executar laços, porém com uma grande diferença entre eles.

Com o **for** ou com o **while** pode ocorrer de o laço não ser executado nenhuma vez. Já com o **do while** ele sempre será executado pelo menos uma vez.

Sua sintaxe é:

```
do
```

```
{
      #comandos
}while(condição);
```

Vamos fazer um laço enquanto a variável $i for menor ou igual a 100.

```
$i = 128;
do
{
      echo $i;
      $i++;
}while($i <= 100);
```

2.2.9.8 FOREACH

O **foreach** é uma estrutura de repetição para uso exclusivo com vetores. Ele percorrerá, um a um, todos os índices do vetor que for informado.

Existem duas possibilidades para o uso do **foreach**: acessar somente os valores do vetor ou acessar os índices e os valores.

A sintaxe é:

```
// para buscar o índice e o valor
foreach(vetor AS indice => valor)
{
      #comandos
}
```

Ou:

```
// para buscar somente o valor
foreach(vetor AS valor)
{
      #comandos
}
```

Vamos fazer um laço para percorrer o vetor $frutas.

```
$frutas = array("Maçã", "Goiaba", "Pera");
foreach($frutas AS $indice => $valor)
{
```

```
	echo “Na posição ”.$indice.” temos a fruta ”.$valor;
}
```

Ou:

```
$frutas = array(“Maçã”, “Goiaba”, “Pera”);
foreach($frutas AS $valor)
{
	echo $valor;
}
```

2.2.9.9 BREAK

A instrução **break** tem como meta finalizar a estrutura de condição ou de repetição. Então, sempre que for executado o **break** no programa, ele interromperá o processamento da estrutura e irá para a próxima linha de comando após a estrutura.

```
<?php
	for($i = 0; $i < 10; $i++)
	{
		if($i == 5)
		{
			break;
		}
		echo $i.”<br />”;
	}
	echo “saiu do for”;
?>
```

Vamos procurar compreender o que ocorreu:

O *loop* foi iniciado com a variável $i tendo o valor '0' e assim foi até a variável chegar no valor '5'. Quando isso aconteceu foi executado o comando **break**, interrompendo o processamento. O programa, então, executou o **echo** “saiu do **for**”.

Caso o **break** não fosse executado, o processamento iria até a variável $i chegar no valor '10'.

2.2.9.10 CONTINUE

A instrução **continue** é similar ao **break**, sua função é interromper a estrutura de repetição atual, entretanto não interrompe a estrutura inteira, e sim, “pula” para o próximo *loop*.

Além disso, o **continue** só é usado com estruturas de repetição, diferentemente do **break**, que pode ser utilizado no **switch**.

```
<?php
      for($i = 0; $i < 10; $i++)
      {
            if($i == 5)
            {
                  continue;
            }
            echo $i.”<br />”;
      }
      echo “saiu do for”;
?>
```

Entendendo o que ocorreu:

O *loop* iniciou com a variável $i tendo o valor 0 e assim foi até a variável chegar ao valor 5. Quando isso aconteceu executou-se o comando **continue,** interrompendo o processamento do *loop* e fazendo com que o programa seguisse para o próximo *loop*. Consequentemente, não foi executado o **echo $i.”
”** quando a variável $i possuía o valor '5', mas continuou até que ela chegasse ao valor '10'.

2.3 Funções

Funções são trechos de códigos reutilizáveis. Utilizaremos funções para realizar tarefas rotineiras.

Para criar uma função usaremos a palavra reservada **function** antes do nome e, para executá-la, bastará escrever o seu nome e passar os argumentos, caso existam.

Por exemplo, temos um sistema que exibe a lista com preços de produtos de uma loja. Para não repetir o código que calcula o preço do produto, fazer uma função

que realize essa tarefa. Sempre que for preciso calcular o preço dos produtos, utilizaremos a função.

2.3.1 Escopo

Pode-se definir **escopo** como sendo o limite até onde o programa vai. Em outras palavras, o **escopo** de um programa em PHP vai da primeira linha de um arquivo até a sua última linha.

Funções também possuem seu escopo e, apesar de uma função estar definida dentro de um arquivo PHP, ela tem um escopo independente do resto programa, ou seja, a função não é afetada pelo programa e o programa não é afetado pela função, a não ser que optemos por isso acontecer.

Inicialmente, as únicas formas de uma função trocar informações com o programa são através de **argumentos** e **retorno de valores**.

O escopo de uma função é representado por chaves '{}'.

```
function nome()
{
      escopo
}
```

2.3.2 Argumentos

Argumentos são valores que o programa informa para a função. Podem ser passados quantos **argumentos** forem necessários para o programa. São separados por vírgula e podem possuir um valor padrão.

Para declarar um **argumento** basta informar o nome da variável nos parênteses, após o nome da função.

```
Function nome($argumento1, $argumento2, $argumento3)
```

Obs.: *As variáveis $argumento1, $argumento2 e $argumento3 só existirão dentro da função, então pode haver variáveis com os mesmos nomes fora dela, sem que os valores sejam alterados.*

Agora definiremos um valor padrão para um argumento.

```
Function nome($argumento1, $argumento2, $argumento3 = “Valor padrão”)
```

	***Obs.:*** *Sempre que uma função for utilizada será OBRIGATÓRIO informar todos os argumentos, menos aqueles que possuem valor padrão. Caso não sejam informados, será empregado o valor padrão do argumento.*

	***Dica:*** *Sempre deixe os valores padrão por último na declaração dos argumentos, pois um argumento com valor padrão no início da função não adianta em nada, já que para passarem os outros argumentos é necessário informar o primeiro.*

2.3.3 Retorno de valores

Uma função faz uso de argumentos para receber valores e utiliza retorno para informar valores ao programa.

Poderemos retornar qualquer tipo de dado de uma função, bastará informar a palavra reservada **return**. A execução da função será interrompida quando o **return** for executado.

```
Function nome($argumento1, $argumento2, $argumento3)
{
      $total = ($argumento1 * $argumento2) / $argumento3;
      return $total;
      echo "Esse echo não vai ser executado, pois antes dele foi executado o return e assim a execução da função foi interrompida.";
}
```

	***Obs.:*** *Não é obrigatório o retorno de algum valor, então é possível retornar um valor, não retornar nenhum valor ou imprimir algum valor na tela.*

Vamos fazer uma função que executará um cálculo e retornar um valor.

```
function soma($a, $b)
{
      $total = $a + $b;
      return $total;
}

$soma = soma(15,37);
echo $soma;
```

Nota: *O valor da soma executada foi enviado para a variável $soma, isso porque ela recebeu o retorno da função.*

Vamos fazer uma função que execute um cálculo e imprima o valor na tela.

```
Function soma($a + $b)
{
      $total = $a + $b;
      echo $total;
}
soma(15,37);
```

2.4 Vetor

Vetores (também conhecidos como *arrays)* são variáveis, porém com uma grande diferença: uma variável comum consegue armazenar somente um valor, enquanto um vetor pode armazenar vários valores.

2.4.1 Índice

Para trabalharmos com vetores faz-se necessário entender o que é **índice** de um vetor.

Um **índice** é a forma que teremos para acessar os valores do vetor, como se fossem chaves de acesso aos valores armazenados no array. Podem ser criados manual ou automaticamente, ou ainda, mesclando as duas formas em um mesmo vetor. Os índices podem ser números ou *strings*.

2.4.2 Criando um vetor

Basicamente são duas as formas de se criar um vetor.

Exemplo 1:

```
$vetor = array(“Fiat”, “Ford”, “BMW”);
```

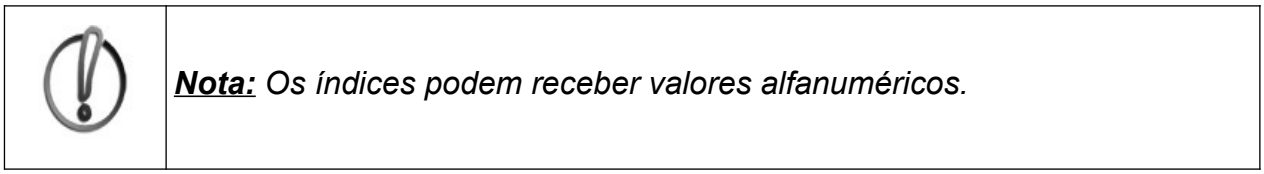
No exemplo criamos um vetor com os valores 'Fiat', 'Form' e 'BMW', entretanto quanto aos índices, o PHP gerou automaticamente. Sendo o índice '0' para 'Fiat', '1' para 'Ford' e '2' para 'BMW'.

Caso necessário você também pode definir os índices manualmente.

Exemplo 2:

```
$vetor = array(1 => “Fiat”, 12 => “Ford”, 57 => “BMW”, “Audi”);
```

Conforme pode-se perceber, foram definidos apenas três índices. Um deles o PHP criou automaticamente, nesse caso, o índice 58.



Nota: *Os índices podem receber valores alfanuméricos.*

Vejamos uma outra forma de criar um vetor.

```
$vetor[] = “Fiat”;
$vetor[] = “Ford”;
$vetor[] = “BMW”;
```

Nesse caso o PHP também definiu o índice automaticamente.

```
$vetor[1]  = “Fiat”;
$vetor[12] = “Ford”;
$vetor[57] = “BMW”;
$vetor[]   = “Audi”;
```

Aqui foram informados os índices de três valores e um deles foi criado automaticamente.

2.4.3 Acessando o conteúdo de um vetor

Para acessar o conteúdo de um índice torna-se essencial saber em qual deles está o valor que procuramos.

Conhecido o índice, bastará informá-lo entre os colchetes ([]);

```
$vetor = array(1 => “Fiat”, 12 => “Ford”, 57 => “BMW”, “Audi”);
echo $vetor[57];
```

Nesse exemplo será impresso o valor contido no índice 57.

O PHP possui várias funções internas que manipulam vetores. Para conhecê-las acesse www.php.net/array.

Para visualizar na tela o conteúdo de um array poderemos usar uma função que imprima suas chaves e valores na tela. Usando o exemplo acima.

```
print_r($vetor);
```

Que resultará em:

```
Array([1]=>Fiat [12]=> Ford [57]=> BMW [58] =>Audi
```

2.5 Manipulação de Arquivos

Com o PHP poderemos criar, ler e apagar arquivos texto e arquivos comprimidos (zip) no servidor, desde que tenham permissão no sistema operacional para que sejam executadas tais ações.

2.5.1 file

Lê um arquivo texto inteiro criando um vetor; cada linha do arquivo texto será um índice do vetor criado.

```
$linhas = file (“arquivo.txt”);
```

Para acessar o conteúdo, bastará utilizar a estrutura de repetição “foreach”.

2.5.2 file_get_contents

Lê um arquivo inteiro, jogando todo o seu conteúdo para uma única *string*.

```
$arquivo = file-get_contents (“arquivo.txt”);
```

Para acessar o conteúdo, basta utilizar a função “echo”.

2.5.3 file_put_contents

Escreve uma *string* em um arquivo. O parâmetro **arquivo** indica onde está o arquivo e o parâmetro **dados** é a *string* que será escrita no arquivo.

```
file_put_contents($arquivo, $string);
```

2.5.4 copy

Copia um arquivo.

```
copy($arquivo,$arquivo_backup);
```

2.5.4.1 unlink

Apaga um arquivo.

```
unlink ($arquivo);
```

2.5.5 fopen

Abre um arquivo.

Ao usar a fopen para abrir arquivos, é preciso informar de que modo o arquivo pode ser aberto, por exemplo, poderá abri-lo como somente leitura, conforme exemplificado abaixo.

```
$handle = fopen("arquivo.txt","r");
```

Na tabela a seguir, a relação dos modos com os quais se pode abrir um arquivo.

Modo	Descrição
r	Abre somente para leitura; posiciona o ponteiro no começo do arquivo.
r+	Abre para leitura e escrita; posiciona o ponteiro no começo do arquivo.
w	Abre somente para escrita; posiciona o ponteiro no começo do arquivo e reduz o comprimento do arquivo para zero. Se o arquivo não existir, tenta criá-lo.
w+	Abre para leitura e escrita; posiciona o ponteiro no começo do arquivo e reduz o comprimento do arquivo para zero. Se o arquivo não existir, tenta criá-lo.
a	Abre somente para escrita; posiciona o ponteiro no final do arquivo. Se o arquivo não existir, tenta criá-lo.
a+	Abre para leitura e escrita; posiciona o ponteiro no final do arquivo. Se o arquivo não existir, tenta criá-lo.
x	Cria e abre o arquivo somente para escrita; posiciona o ponteiro no começo do arquivo. Se o arquivo já existir, a chamada para **fopen()** falhará, retornando FALSE e gerando um erro de nível E_WARNING. Se o arquivo não existir, tenta criá-lo. Isso é equivalente a especificar as *flags* O_EXCL|O_CREAT para a chamada de sistema **open(2)**.
x+	Cria e abre o arquivo para leitura e escrita; posiciona o ponteiro no começo do arquivo. Se o arquivo já existir, a chamada para **fopen()** falhará, retornando FALSE e gerando um erro de nível E_WARNING. Se o arquivo não existir, tenta criá-lo. Isso é equivalente a especificar as *flags* O_EXCL|O_CREAT para a chamada de sistema **open(2)**.

2.5.6 fread

Lê os dados de um arquivo, fazendo a leitura de uma linha por vez.

```
fread ($handle);
```

	***Obs.:*** *O parâmetro* $handle *é o retorno da função* fopen*, então só poderemos utilizar o* fread *depois de ter sido executado o* fopen*.*

2.5.7 fwrite

Grava dados em um arquivo.

```
fwrite($handle,"algum texto");
```

	***Obs.:*** *O parâmetro* $handle *é o retorno da função* fopen*, então só poderemos utilizar o* fwrite *depois de ter sido executado o* fopen*.*

2.5.8 fclose

Fecha um arquivo aberto.

```
fclose($handle);
```

	***Obs.:*** *O parâmetro* $handle *é o retorno da função* fopen*, então só poderemos utilizar o* fclose *depois de ter sido executado o* fopen*.*

2.5.9 Criando e escrevendo em um arquivo

Para criar e escrever no arquivo deve-se, primeiramente, abri-lo. Abrir significa que teremos acesso a ele e ver seu conteúdo. Depois escreveremos e, por último, fecharemos o arquivo.

```
<?php
      /*
       *  abre o arquivo caso ele já exista e limpa todo o conteúdo
       *  Caso ele não exista será criado, se a pasta tiver
       *  permissão de escrita.
       */
      $handle = fopen("teste.txt", "w+");
      // escreve um texto no arquivo.
      fwrite($handle, "Primeira linha \n");
      fwrite($handle, "segunda linha \n");
      fwrite($handle, "\t terceira linha");

      // Fecha o arquivo.
```

```
        fclose($handle);


?>
```

Nota: *Repare que foram utilizados caracteres diferentes, como o* ***\n*** *e o* ***\t****. O primeiro deles é responsável por fazer a quebra de linha em arquivos texto e o segundo serve para tabular os textos que serão escritos no arquivo.*

2.5.10 Lendo um arquivo texto

Para que possamos ler as informações de um arquivo, necessitaremos abri-lo para leitura. Depois disso percorreremos todo o arquivo mostrando o seu conteúdo. Após a leitura, bastará fechá-lo.

```
<?php
        // abre o arquivo para leitura
        $handle = fopen("teste.txt", "r");


        // percorre todo o arquivo mostrando o conteúdo linha por linha
        while(!feof($handle))
        {
                $linha = fgets($handle);
                echo $linha."<br />";
        }
        // fecha o arquivo
        fclose($handle);
?>
```

2.6 Banco de Dados

O PHP possui suporte nativo a muitos bancos de dados, o que é um grande diferencial em relação a outras linguagens, nas quais torna-se imprescindível instalar extensões para ter acesso a determinados bancos.

Um dos bancos de dados mais utilizados com o HP é o MySQL. Assim verificar-se-á como fazer a interação com o MySQL através do PHP.

Para maiores informações sobre quais banco de dados o PHP suporta, acessar http://br.php.net/manual/pt_BR/refs.database.php.

2.6.1 Conectando no MySQL

Para realizar a conexão com o MySQL o PHP emprega-se a função **mysql_connect**.

```
$con = mysql_connect($host,$usuario,$senha);
```

- Host: Informa em qual computador está o banco de dados. Pode ser informado o IP ou DNS do servidor.
- Usuário: Usuário que irá se conectar ao MySQL.
- Senha: Senha de acesso do usuário informado para o servidor MySQL.

Vejamos um exemplo de como realizar uma conexão com o MySQL:

```
$conexao = mysql_connect("localhost", "root", "elaborata") or
die(mysql_error());
```

No exemplo acima estamos realizando uma conexão, onde o MySQL está instalado, no mesmo computador em que está o PHP, por esse motivo utilizaremos a palavra **localhost**. Foram usados usuário **root** e senha **elaborata** para realizar a conexão.

Nota: *Observamos que uma variável recebeu o retorno da função* ***mysql_connect****. Guardar esse valor retornado não é obrigatório, porém recomenda-se, pois poderemos utilizá-lo no momento de realizar algumas operações no MySQL. A razão disso é que na variável $conexão teremos um elemento chamado RESOURCE e nele estarão contidas as informações da conexão. Assim, o PHP consegue enviar e receber informações do MySQL como se houvesse um "túnel" ligando os dois.*

Notar que após a função **mysql_connect** foi usada a expressão **or die(mysql_error())**.Com ela, caso aconteça algum erro na hora de realizar a conexão, será retornado um erro informando o motivo da falha. Resumindo, ou o PHP executa a conexão com sucesso ou interrompe a execução do programa e mostra o que ocorreu. Se não for empregado o **or die(mysql_error())**, o tempo despendido poderá ser maior até que seja descoberto o motivo pelo qual não estará funcionando a sua aplicação.

2.6.2 Selecionando o banco de dados

Não basta apenas realizar a conexão com o MySQL, também é indispensável informar qual será o banco de dados selecionado.

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

Sendo que somente o primeiro parâmetro é obrigatório, que é exatamente o nome do banco de dados com o qual será realizada a conexão. O segundo parâmetro é o **resource** da conexão, caso não seja informado, será utilizado o **resource** da última conexão aberta com o banco de dados.

Nota: *Veja que diferentemente do* ***mysql_connect****, no* ***mysql_select_db*** *não se armazena nenhum valor em nenhuma variável.*

2.6.3 Executando comandos no MySQL

Depois de aberta a conexão, poderemos executar comandos no banco e sempre que um comando deva ser executado, utilizaremos a função **mysql_query**. O resultado da query ficará armazenado na variável $result, como observa-se a seguir.

```
$result = mysql_query("select * from tabela");
```

Basicamente quatro tipos de comando serão executados: INSERT, SELECT, UPDATE e DELETE. O segundo parâmetro que a função pede, que é opcional, é o retorno da função **mysql_connect** (*resource*) que, no exemplo dado, foi guardado na variável $conexao.

No exemplo a seguir, usaremos as últimas três funções mostradas.

```
// abrindo a conexão com o MySQL
$conexao = mysql_connect("localhost", "root", "elaborata") or
die(mysql_error());


// selecionando o banco de dados
mysql_select_db("curso", $conexao);


// executando um comando no MySQL
$result = mysql_query("select *from alunos", $conexao) or die(mysql_error());
```

Lembrando que dentre os quatro comandos comentados, apenas de um deles sempre iremos esperar um retorno da função **mysql_query**, é quando realizamos um **select**. Se executarmos um SELECT significa que queremos buscar alguma informação do banco de dados, por isso é preciso guardar o retorno da função **mysql_query** em uma variável.

O valor retornado pelo banco de dados recebe o nome de **resultset**.

No exemplo anterior, os valores retornados ainda estão inacessíveis. Para poder acessar os dados será necessário executar uma outra função, que veremos na sequência.

2.6.4 Acessando valores de uma consulta realizada

Depois de executar o **mysql_query** com um SELECT, a próxima etapa será acessar as informações que foram retornadas pelo MySQL. Para isso veremos a função **mysql_fetch_assoc**.

A função **mysql_fetch_assoc** torna acessíveis os dados que foram consultados pelo **mysql_query**.

Quando a query é executada, cria-se um vetor associativo pelo nome do campo da tabela no banco de dados, ou seja, o índice do vetor será o nome do campo da tabela selecionada.

```
mysql_fetch_assoc($result);
```

Obs.: *A função **mysql_fetch_assoc** poderá ser utilizada para selecionar apenas um registro ou vários. Caso sejam vários registros selecionados, você deverá fazer um laço com o **while**, por exemplo.*

Exemplo:

```
// abrindo a conexão com o MySQL
$conexao = mysql_connect("localhost", "root", "elaborata") or
die(mysql_error());


// seleciona o banco de dados
mysql_select_db("curso", $conexao);


// executando um comando no MySQL
$result = mysql_query("SELECT nome, email FROM alunos")
```

```
or die(mysql_error());


//recuperando os valores selecionados
while($row = mysql_fetch_assoc($result))
{
      echo $row['nome'];
      echo $row['email'];
}
```

Obs.: *Caso seja necessário contar as linhas encontradas no banco de dados, ou apenas testar se a query retornou algo antes de executar uma estrutura de repetição, como o while, poderemos usar a função mysql_num_rows.*

Para consultar todas as funções de manipulação do MySQL acesse: http://br.php.net/manual/pt_BR/ref.mysql.php

2.7 Cookies e Sessões

PHP e Apache, por padrão, não guardam nenhuma informação de estado, ou seja, é impossível saber quem, nem quantas vezes o acesso ocorreu. A função do Apache é receber uma requisição e respondê-la, porém ele não registra nenhuma informação internamente.

Em muitos casos haverá necessidade de guardar informações que serão acessadas em uma ou em várias outras páginas. Nesse ponto é que entram **cookies** e **sessões**.

2.7.1 Cookie

Cookie é um arquivo texto criptografado que fica armazenado no navegador de quem está acessando o site. Cada navegador guarda os seus *cookies*.

Hoje, a principal aplicação para *cookies* é guardar preferências de usuários, como o Google faz na sua página de buscas; nela você pode configurar a quantidade de registros que devem aparecer em cada consulta, por exemplo.

Observar que se isso foi realizado em um computador, caso o usuário mude para outra máquina, nela estarão as configurações padrão do Google.

Portanto, uma vez que os *cookies* são conservados nos computadores que estão realizando o acesso, não é interessante guardar neles informações relevantes, como senhas de acesso ao sistema da empresa ou valores de produtos; se elas estiverem no computador de quem está acessando o site, esse usuário, por sua vez, poderá obter e até alterar tais informações.

2.7.1.1 Criando um cookie

Para criar um *cookie* existe a função **setcookie**, responsável por enviar o arquivo texto ao navegador.

Exemplo de como se criar um *cookie* com o nome de 'curso' e valor 'PHP':

```
setcookie(“curso”, “PHP”);
```

Um outro recurso muito empregado no *cookie* é definir uma data de expiração, isto é, o prazo diário de validade. Para isso utilizaremos o parâmetro **expire**.

Exemplo de como criar um *cookie com expiração:*

```
setcookie(“dia”,”sábado”,time()+3600);
```

No exemplo acima, usamos a função time( ), que retorna a hora atual, e adicionamos 3600 segundos, o que fará com que o *cookie* expire em 1 hora, a partir da hora atual.

2.7.1.2 Lendo um cookie

Depois de criar um *cookie* poderemos ler o seu valor, para isso utilizaremos a variável superglobal $_COOKIE;

Exemplo:

```
echo $_COOKIE['curso'];
```

Obs.: *Um problema que existe com os* cookies *é que os navegadores podem ser bloqueados para recebê-los. Nesse caso, o seu sistema pode não funcionar corretamente.*

2.7.2 Sessão

Sessão é um recurso muito parecido com o *cookie*, entretanto ficará guardado no servidor e sua estimativa de vida é bem mais curta. Uma vez que seu armazenamento é no servidor já não há mais o problema de o usuário tentar ler ou alterar os valores, pois não terá acesso ao servidor onde está funcionando o site.

Uma outra vantagem em utilizar sessões é que o funcionamento delas não depende de nenhuma configuração do navegador, somente do servidor.

2.7.2.1 Iniciando uma sessão

Para iniciar uma sessão empregaremos a função **session_start**.

```
<?php
      session_start();
?>
```

Nota: *Não é possível iniciar duas vezes a sessão no mesmo arquivo, caso aconteça ele exibirá uma mensagem de erro, a não ser que seja utilizado o operador de controle de erro.*

2.7.2.2 Criando variáveis de sessão

Para criar variáveis de sessão bastará usar a superglobal $_SESSION. O nome da variável deverá ser informado como índice da superglobal.

Exemplo:

```
// iniciando a sessão
session_start();

// criando uma variável de sessão
$_SESSION['curso'] = “PHP”;
```

2.7.2.3 Lendo valores de variáveis de sessão

Para ler o valor de uma sessão bastará iniciá-la e fazer o acesso pela superglobal $_SESSION.

Exemplo:

```
// iniciando a sessão
session_start();
// criando uma variável de sessão
$_SESSION['curso'] = “PHP”;
// lendo o valor de uma variável de sessão
echo “Você está no curso de {$_SESSION['curso']}”;
```

Obs.: *Poderemos criar uma sessão em uma página e acessar esse valor em outras páginas. Para isso bastará iniciar a sessão.*

Dica: *Procure sempre utilizar o operador de controle de erro, pois isso evita que mensagens inúteis sejam exibidas. Por exemplo: @session_start( )*

2.7.2.4 Destruindo uma sessão

Depois de utilizarmos a sessão é importante destruí-la. Com isso todas as variáveis criadas serão igualmente destruídas. Para desfazer uma sessão usaremos a função **session_destroy**.

Vamos criar uma página onde iremos estabelecer uma variável de sessão e outra página que fará o acesso a essa sessão.

O arquivo cria_sessao.php terá o seguinte conteúdo:

```
<?php
      // inicia a sessão
      @session_start();

      // cria a variável de sessão
      $_SESSION['curso'] = “PHP”;
      $_SESSION['local'] = “Elaborata Informática”;
?>
```

O arquivo le_sessao.php terá o seguinte conteúdo:

```
<?php
      @session_start();
      echo “Você está no curso de ”.$_SESSION['curso'].” na ”.
$_SESSION['local'];
?>
```


ELABORATA
INFORMÁTICA
CAPÍTULO 3
ORIENTAÇÃO A OBJETOS
NO PHP

PHP

3.1 Introdução

A Programação Orientada a Objetos vem sendo costumeiramente utilizada em muitas linguagens, a partir de meados da década de 1960.

O PHP começou a implementar a Orientação a Objetos na sua versão 4, mas somente na versão 5, se tornou realmente popular e mais estruturada.

A Orientação a Objetos, também conhecida como Programação Orientada a Objetos (POO), ou ainda, em inglês, *Object-Oriented Programming* (OOP), é um paradigma de análise, projeto e programação de sistemas de *software* baseado na composição e interação entre diversas unidades de *software* chamadas de objetos.

Em alguns contextos, se prefere usar “modelagem orientada ao objeto”, em vez de “projeto”.

A análise e o projeto orientados a objetos têm como meta identificar o melhor conjunto de objetos para descrever um sistema de *software*. O funcionamento desse sistema se dá através do relacionamento e troca de mensagens entre os objetos.

Na Programação Orientada a Objetos implementa-se um conjunto de classes que definem os objetos presentes no sistema de *software*. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.

3.2 Modelagem de Sistemas

A modelagem de sistemas é uma etapa importantíssima no desenvolvimento de software, principalmente quando falamos de Orientação a Objetos.

Comumente, a maneira de definir modelagem de sistemas orientados a objetos é que basta desenvolver UML e classes e pronto, estará modelado. Mas na verdade não é apenas isso, pois muitos outros conceitos são envolvidos. Modelar sistemas é abstrair problemas da realidade para modelos computacionais, organizada e padronizadamente, onde a base de tudo isso é o que iremos tratar como objeto.

Objeto é onde serão definidos a estrutura e o comportamento do mesmo. O desenvolvimento orientado a objetos facilita na hora de entender o que deve ser desenvolvido.

Basicamente, modelagem de sistema é composta por três processos: a etapa de análise, a etapa do projeto e a etapa da implementação.

3.2.1 Etapa de análise

A etapa de análise deve ser a primeira a se realizar e nela não se deve pensar em como fazer o sistema funcionar, mas sim, no que o sistema terá que fazer. Ela envolverá, principalmente, conversas com o cliente e a procura pela melhor solução para o problema que ele possui.

3.2.2 Etapa do projeto

A etapa de análise deve se concentrar no que deve ser feito, já a etapa do projeto concentra-se em como fazer.

Nessa etapa em que começa a parte mais técnica é o que o *software* deve apresentar as funcionalidades, de acordo com o que foi levantado na fase anterior. Durante o projeto deve ser verificado se o resultado levantado possibilita desenvolver ou se haverá alguma limitação.

3.2.3 Etapa de implementação

É nessa etapa que haverá a escolha da linguagem a ser utilizada, a definição de como será o desenvolvimento e, por fim, a codificação. Normalmente esse estágio é o mais rápido, pois basta, basicamente, fazer a codificação.

3.3 Classe

Uma classe abstrai um conjunto de objetos com características similares. Ela define o comportamento de seus objetos através de métodos e os estados possíveis desses objetos por meio de atributos. Em outros termos, uma classe descreve os serviços providos por seus objetos e quais informações eles podem armazenar.

Classes não são diretamente suportadas em todas as linguagens e são necessárias para que uma linguagem seja orientada a objetos.

Uma classe é composta por atributos e métodos. Para criá-la usamos a palavra reservada **class** seguida do nome. Posteriormente, chaves ({}) indicam seu início e fim.

Exemplo:

```
class Teste
{
      // atributos
      // métodos
}
```

3.3.1 Variável $this

Os métodos e atributos de uma classe conseguem conversar entre si, mas para isso é necessário utilizar a variável $this, seguida do sinal de menos '-' e do sinal de maior '>', desta forma: **$this->**. Na sequência, o nome do método ou do atributo que você deseja acessar.

3.3.2 Atributo

Atributos são os elementos que formam a estrutura de uma classe, também conhecidos como variáveis de uma classe.

Para criar um atributo bastará escrever a palavra reservada **var** e o nome desse atributo. O seu valor pode ou não ser inicializado e não é preciso definir o tipo de dado que ela irá suportar.

Normalmente os atributos são os primeiros elementos a serem criados na classe.

Exemplo:

```
class Teste
{
      var $nome;
      var $idade;
}
```

3.3.3 Método

Os métodos definem as ações que uma classe terá. São igualmente conhecidos como procedimentos ou funções de uma classe.

Para declarar um método o procedimento é o mesmo que para criar uma função. Utiliza-se a palavra reservada **function** seguida do nome da função e parênteses contendo ou não parâmetros.

Iremos criar um método para alterar o valor do atributo **nome** e um método para mostrar na tela o valor desse atributo.

```
class Teste
{
	var $nome;
	var $idade;
	function setNome($novoNome)
	{
		$this->nome = $novoNome;
	}
	funtion mostraNome()
	{
		echo $this->nome;
	}
}
```

Nota: *Perceba que utilizamos a variável $this para alterar e acessar o valor do atributo $nome.*

Vejamos como fazer um método executar um outro método.

```
class Teste
{
	var $nome;
	var $idade;
	function setNome($novoNome)
	{
		$this->nome = $novoNome;
		$this->mostraNome();
	}
	function mostraNome()
	{
		echo $this->nome;
	}
}
```

Nesse exemplo, ao ser chamado o método **setNome**, ele mesmo irá executar o método **mostraNome**.

3.3.4 Encapsulamento

Encapsular é o modo que temos de controlar quem terá acesso aos métodos e atributos.

Existem três níveis de encapsulamento, conforme veremos a seguir.

3.3.4.1 Public

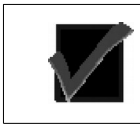
O modificador **public** informa que o método ou o atributo está aberto para qualquer tipo de acesso. Nesse acesso haverá o objeto (que ainda não é conhecido), a própria classe e outras classes provenientes da herança (a qual passará a ser conhecida também).

Exemplo:

```
class Teste
{
	public $nome;
	public $idade;

	public function setNome($novoNome)
	{
		$this->nome = $novoNome;
		$this->mostraNome();
	}

	public function mostraNome()
	{
		echo $this->nome;
	}
}
```



Obs.: *Veja que no atributo não há mais a palavra reservada* ***var*** *antes de seu nome, e sim, a palavra* ***public****.*

3.3.4.2 Protected

O modificador **protected** informa que o método ou atributo só poderá ser acessado pela mesma classe ou por outra classe através da herança. Dessa forma, o objeto não pode mais acessar o método ou o atributo diretamente.

Exemplo:

```
class Teste
{
	protected $nome;
	protected $idade;

	protected function setNome($novoNome)
	{
		$this->nome = $novoNome;
		$this->mostraNome();
	}

	protected function mostraNome()
	{
		echo $this->nome;
	}
}
```

3.3.4.3 Private

O modificador **private** informa que somente a classe-dona poderá acessar o método ou o atributo, sendo assim, nem o objeto, nem outras classes através da herança, conseguirão fazer o acesso.

Exemplo:

```
class Teste
{
	private $nome;
	private $idade;

	private function setNome($novoNome)
	{
		$this->nome = $novoNome;
		$this->mostraNome();
	}

	private function mostraNome()
	{
		echo $this->nome;
	}
}
```

3.4 Objeto

Objeto é uma instância de uma classe, ou seja, uma cópia da classe para memória. É através dele que serão acessados os métodos e atributos de uma classe.

Para criar um objeto basta definir uma variável que receberá a instância da classe. Após determinarmos qual será o nome do objeto (nome da variável), informaremos o sinal de igual (=) e a palavra reservada **new**, seguida do nome da classe e de parênteses.

Para acessar os métodos ou atributos, informamos o nome do objeto seguido do sinal de menos (-) e do sinal de maior (>), formando assim um seta (→), da mesma maneira que utilizamos com a variável $this em uma classe.

Em síntese, para acessar métodos e atributos por meio de um objeto, repetiremos o mesmo processo de quando acessamos os métodos e atributos de uma mesma classe, porém substituiremos a variável $this pelo objeto.

Exemplo:

```
class Teste
{
	var $nome;
	function setNome($novoNome)
	{
		$this->nome = $novoNome;
	}

	function mostraNome()
	{

		echo $this->nome;
	}
}

#instanciando um objeto
$teste = new Teste();
#acessando o método setNome
$teste->setNome("Diego");
#acessando o método mostraNome
$teste->mostraNome();
```

Note que ao executarmos o método **setNome** estaremos informando um parâmetro, isso porque o método obriga a passagem do mesmo. No método **mostraNome**, executaremos informando os parênteses, sem nenhum parâmetro, já que não necessita, da mesma maneira que ocorre quando criamos e executamos uma função.

3.5 Herança

Herança, em Orientação a Objetos, tem o mesmo conceito que na vida real, que é receber características do pai ou da mãe. Então, quando criamos uma herança, uma classe herda todos os métodos e atributos de outra classe, exceto os métodos e atributos que tiverem o modificador **private**.

Para realizar uma herança bastará colocar a palavra reservada **extends** logo após o nome da classe e, depois do **extends**, informar o nome da classe pai.

Uma classe só pode ter uma classe pai, mas pode ter várias classes filhas.

Exemplo:

```
class Pai
{
	public $nome;
	protected $idade;
	private $endereço;
	public function setEndereco($novoEndereco)
	{
		$this->endereco = $novoEndereco;
	}
}

class Filho EXTENDS Pai
{
	public function setNome($novoNome)
	{
		$this->nome = $novoNome;
	}

	public function mostraNome()
	{
		echo $this->nome;
	}
}
```

3.5.1 Polimorfismo

Polimorfismo é a capacidade de algo mudar de forma ou possuir várias formas. Em programação também pode haver polimorfismo.

Em linguagem de programação é representado por uma classe ou um método que tem um nome em comum, porém com ações diferentes executadas, por meio de

um mecanismo automático utilizado com herança. Isso quer dizer que poderemos sobrescrever métodos e atributos nas classes filhas. Para isso, criaremos um método de um atributo com o mesmo nome, assim prevalecerá o método ou atributo da classe filha.

Verificaremos isso no exemplo a seguir, quando o método setEndereco da classe filha sobrescreveu o método setEndereco da classe pai.

Exemplo:

```
class Pai
{
      public $nome;
      protected $idade;
      private $endereço;
      public function setEndereco($novoEndereco)
      {
            $this->endereco = $novoEndereco;
      }
}

class Filho EXTENDS Pai
{
      public function setEndereco()
      {
            $this->endereco = “O endereço foi alterado”;
      }

      public function setNome($novoNome)
      {
            $this->nome = $novoNome;
      }

      public function mostraNome()
      {
            echo $this->nome;
```

```
        }
}
```

No exemplo anterior criamos um método chamado **setEndereco** na classe filha, sem que ela recebesse nenhum parâmetro. Dessa forma sobrescrevemos o método da classe pai.

3.6 Interfaces

Interfaces têm um papel relevante no processo de desenvolvimento de um *software*, pois representam o esqueleto de toda uma classe e, consequentemente, de um projeto. São empregadas para informar o que deve ser feito, muito embora não mencione como isso funcionará.

Exemplo:

Temos uma interface que descreve a classe **Aluno**. Nela está especificado que devemos ter um método **Média** que recebe como parâmetro a **Nota** apenas, mas ela não informa como deve ser o processo de cálculo da média, nem onde obter a nota.

3.7 Abstract

Utiliza-se **abstração** quando necessitarmos que uma classe sirva de modelo para outras classes; criamos uma classe abstrata com métodos abstratos, isto é, que na classe pai não desempenhem nenhuma ação, mas cuja implementação seja obrigatória nas classes filhas.

Exemplo:

```
abstract class aluno
{
        abstract public function ativar();
        abstract public function inativar();
}
class aluno_PHP EXTENDS aluno
{
        public $estado;
        public function ativar()
        {
```

```
            $this->estado = “Ativo”;
      }

      public function inativar()
      {
            $this->estado = “Inativo”;
      }
}
```

No exemplo acima criamos uma classe chamada **aluno** que é dotada de alguns requisitos básicos para suas classes filhas. Ela definiu que todas as classes filhas terão que implementar obrigatoriamente métodos de ativar e inativar.

3.8 Final

Faz-se uso da palavra reservada **final** quando precisarmos que um método ou atributo não possa sofrer polimorfismo, ou seja, que ele não possa ser sobrescrito.

Exemplo:

```
class Pai
{
      public $nome;
      protected $idade;
      private $endereço;
      final public function setEndereco($novoEndereco)
      {
            $this->endereco = $novoEndereco;
      }
}
class Filho EXTENDS Pai
{
      public function setEndereco()
      {
            $this->endereco = “O endereço foi alterado”;
      }
      public function setNome($novoNome)
```

```
	{
		$this->nome = $novoNome;
	}
	public function mostraNome()
	{
		echo $this->nome;
	}
}
```

Nesse caso, um erro será retornado, informando que não é possível sobrescrever o método **setEndereco**.

3.9 Static

Pode-se definir um método ou um atributo como **static**. Fazendo isso, será possível acessar o método ou o atributo sem a necessidade de instanciar a classe.

Para realizar o acesso sem criar um objeto, deve-se informar o nome da classe, utilizar duas vezes seguidas dois pontos '::' e o nome do método ou do atributo.

Exemplo:

```
class Teste
{
	public static  $curso = "PHP";

	public static function mostraCurso()
	{
		echo self::$curso;
	}
}

echo Teste::mostraCurso();
```

Nesse exemplo, foram criados um atributo e um método com **static**, o que, posteriormente, permitiu o acesso do atributo de fora da classe, sem precisar fazer a criação do objeto.

Veja que para acessar o atributo de dentro do método empregou-se a palavra **self**, significando que ele está acessando um atributo da mesma classe, da mesma forma que é utilizada a variável $this.

Caso exista uma herança, deve ser usada a palavra **parent** no lugar de **self**.

	***Obs.:*** *Quando um atributo ou método de classe for* ***static****, ele será compartilhado por todas as instâncias da classe, isto é, apenas um será criado na memória.*

3.10 Associação OU Composição

Composição ou **associação** é uma espécie de relacionamento que existe entre classes, através do qual uma classe inclui, como se fossem seus, os métodos e atributos de um objeto de uma outra classe.

Em outras palavras, quando fazemos uma herança estaremos realizando uma associação de classe, e quando criamos um objeto de uma outra classe dentro de uma classe, estaremos fazendo uma composição.

A chave para que possamos compreender se é uma composição ou uma associação, deve-se realizar o seguinte questionamento: “é um ou tem um?”.

Vejamos este exemplo:Temos três classes; cliente, aluno e média. Vamos aplicar a pergunta para saber qual é o tipo de relacionamento existente.

Um aluno é um cliente ou tem um cliente?

Resposta: Um aluno é um cliente, então temos **associação**.

Um aluno é uma média ou tem uma média?

Resposta: Um aluno tem uma média, então temos **composição**.

3.11 Criando a Primeira Classe

Agora iremos criar a nossa primeira classe e verificar como utilizá-la.

Criaremos uma classe de recursos gerais. Haverá um método que converterá uma data no formato **dd/mm/aaaa** para o formato de datas do MySQL, que é **aaaa-mm-dd**, e um método que escreverá o dia atual + “de” + nome do mês + “de” + ano (ex: 1 de abril de 2008).

Os métodos serão **static**.

```
<?php
class Recursos
{
	/*
	 * Método que irá converter a data para o formato do MySQL
	 * @param string $data
	 * @return string novaData;
	 */
	public static function converteData($data)
	{
		$novaData = implode("-", array_reverse(explode("/", $data)));
		return $novaData;
	}

	/**
	 * Método que irá escrever a data no formato dia de mês de ano
	 *
	 */
	public static function escreveData()
	{
		$dia = date("d");
		$ano = date(Y);
		switch(date("m"))
		{
			case 01:
				$mes = "Janeiro";
				break;
			case 02:
				$mes = "Fevereiro";
				break;
			case 03:
				$mes = "Março";
				break;
			case 04:
```

```
                        $mes = “Abril”;
                        break;
                case 05:
                        $mes = “Maio”;
                        break;
                case 06:
                        $mes = “Junho”;
                        break;
                case 07:
                        $mes = “Julho”;
                        break;
                case 08:
                        $mes = “Agosto”;
                        break;
                case  09:
                        $mes = “Setembro”;
                        break;
                case 10:
                        $mes = “Outubro”;
                        break;
                case 11:
                        $mes = “Novembro”;
                        break;
                default:
                        $mes = “Dezembro”;
            }

            echo $dia.” de ”.$mes.” de ”.$ano;
      }
}

Recursos::escreveData();
echo Recursos::converteData(date("d/m/Y"));
?>
```

3.12 Métodos Mágicos

O PHP5 trouxe alguns conceitos avançados de orientação a objetos, como visibilidade de métodos e atributos, além de outras melhorias.

Neste capítulo vamos aproveitar essas funcionalidades e aprender como tirar proveito de tais recursos para melhorar a produtividade.

Os métodos mágicos normalmente começam com '__' (“underline underline”). Os mais conhecidos são **__construct**, **__destruct**, **__call** e o **__autoload**.

3.12.1 __construct

Também conhecido como método **construtor** de uma classe, o método **__construct** é executado toda vez que se instancia um objeto da classe. É muito usado para “setar” variáveis de inicialização e chamar algum método privado da classe e métodos construtores da classe pai.

O método construtor precisa ser obrigatoriamente **public** e pode ou não receber parâmetros.

Poderemos, da mesma maneira, formar um método construtor criando um método com o mesmo nome da classe a que ele pertence.

```
class Soma
{
	public function __construct($a, $b)
	{
		echo $a + $b;
	}

}
$soma = new Soma(“15”, “27”);
```

3.12.2_destruct

Conhecido também como método **destrutor**, o método **__destruct** é executado toda vez que um objeto da classe é destruído pelo fim do script ou através da função **unset**. Fundamentalmente, tem como objetivos zerar variáveis, limpar dados de sessões, fechar conexões com banco de dados, entre outros.

O método **destrutor** precisa ser obrigatoriamente **public** e não pode receber parâmetros.

```
class Calcula
{
	private $total
	private $a;
	private $b;
	public function __construct($a, $b)
	{
		$this->a = $a;
		$this->b = $b;
	}

	public function soma()
	{
		$this->total = $this->a + $this->b;
	}

	public function __destruct()
	{
		echo $this->total;
	}
}
$calc = new Calcula("15", "27");
$calc->soma();
```

3.12.3__call

O método **call** tem sua utilização pouco explorada, mas oferece um recurso muito avançado, podendo ser empregado no gerenciamento de mensagens de erro ou para simular outros métodos, os famosos **gets** e **sets**.

É executado toda vez que se chama um método inexistente da classe. Com ele fica fácil manipular o famigerado "Fatal error: Call to undefined method".

Esse método recebe dois parâmetros, um com o nome do método da classe chamada e o outro, um *array* com os parâmetros da chamada da função.

```
class Calcula
{
      private $total
      private $a;
      private $b;
      public function __construct($a, $b)
      {
            $this->a = $a;
            $this->b = $b;
      }

      public function soma()
      {
            $this->total = $this->a + $this->b;
      }

      public function __call($metodo, $argumentos)
      {
            echo “O método {$metodo} não existe, verifique o seu código.”;
      }
}
$calc = new Calcula(“15”, “27”);
$calc->total();
```

3.12.4 __autoload

O **autoload** sempre será executado quando for criada a instância para uma classe, em outras palavras, no momento da criação de um objeto.

Então, quando for executado, será verificado se existe o método **__autoload** no programa. Caso exista, será executado e sempre receberá um parâmetro, que é o nome da classe que está sendo instanciada.

Normalmente é utilizado para fazer a inclusão do arquivo de classes, de maneira automática, contudo faz-se necessária a criação de uma padronização para a nomenclatura dos arquivos.

Exemplo:

Temos um arquivo contendo uma classe de conexão no banco de dados. Esse arquivo encontra-se na pasta **classes** e seu nome é **mysql.class.php**. Ao invés de fazer o **require_once** do arquivo, criaremos o método mágico **__autoload**.

```
<?php
	function __autoload($classe)
	{
		require_once("classes/{$classe}.class.php");
	}

	$db = new mysql();
?>
```

Obs.: *Nesse caso, o nome da classe deve ser o mesmo do arquivo, caso contrário, ele não irá funcionar.*

3.13 Exercícios

1. Crie uma classe que terá a função de realizar cálculos. Ela deverá somar, subtrair, multiplicar e dividir. Para realizar todas as operações, utilize atributos para guardar os valores e o resultado da ação realizada. Cada atributo deverá ter um método, para que seja informado o seu valor. A classe deverá ter um método construtor.
2. Crie uma classe com um método que informe se um número é par ou ímpar. Esse método deverá ser static.
3. Crie uma classe chamada cliente. Ela conterá atributos e métodos para nome, e-mail, telefone, celular, RG e CPF. Crie uma classe aluno que deverá ser estendida da cliente. Ela deverá ter atributos e métodos para cursos e notas.

CAPÍTULO 4

UML

4.1 Introdução

UML significa Unified Modeling Language e foi criada a partir de várias pessoas, em meados da década de 1990. Até então haviam muitas metodologias de modelagem do sistema orientado a objetos e cada uma delas tentava ser o padrão mais utilizado.

Entretanto, exatamente em 1996, houve uma unificação de vários padrões existentes. O trabalho foi liderado por Grady Booch, James Rumbaugh e Ivar Jacobson. Cada um deles trabalhava em um modelo próprio e esse grupo ficou conhecido como “os três amigos”. Assim teve início a UML, que recebeu o melhor de cada modelo existente de seus três pais.

Em 1997, a UML foi aprovada como o padrão a ser seguido pela OMG (Object Management Group - www.omg.org), uma entidade internacional que define os padrões da programação orientada a objetos.

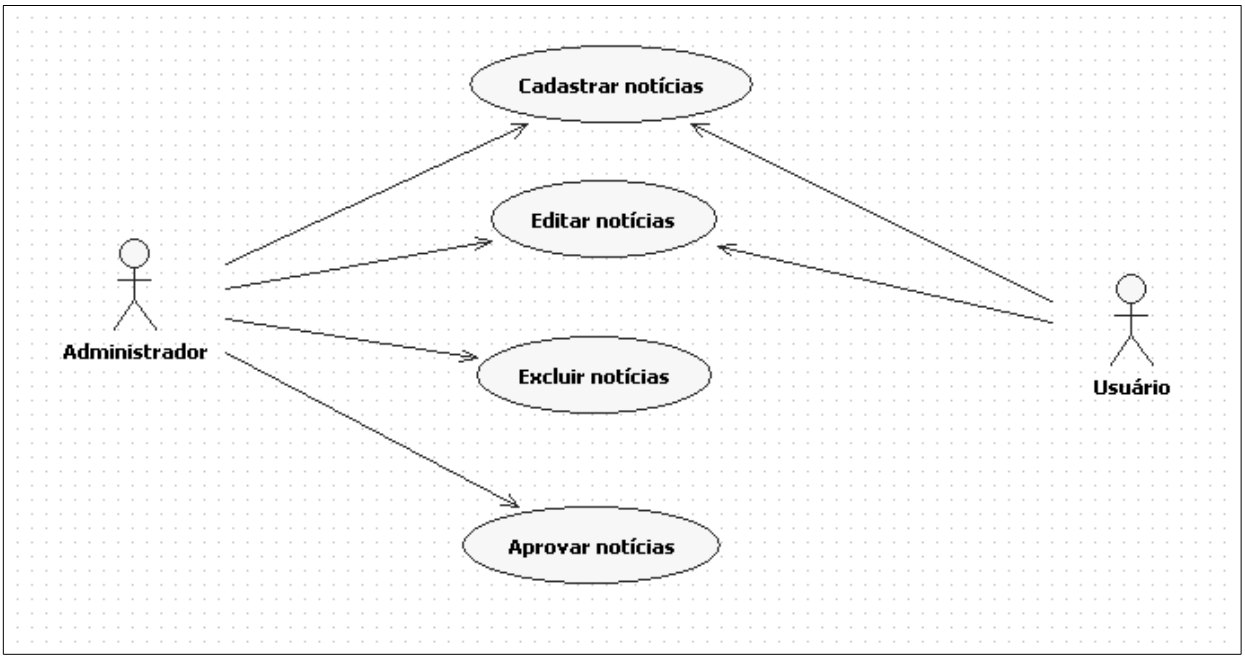
A UML, portanto, é uma linguagem de modelagem de dado visual, utilizando elementos gráficos para representar classes, heranças, composições, entre outros. Possui vários tipos de diagramas que podem ser criados. Veremos os mais utilizados.

4.1.1 Diagrama de caso de uso

O diagrama de **caso de uso**, também conhecido como *use case*, serve para descrever as interações que um usuário (pode ser uma pessoa, um *software* ou um computador) terá com o *software* que está sendo desenvolvido. Esse diagrama não tem a função de descrever rotinas específicas de cada interação do usuário, e sim, mostrar quais serão.

Exemplo:

Vamos demonstrar como seria o papel de dois usuários em um sistema. Um será o administrador e, o outro, simplesmente, o usuário. O sistema terá recursos de cadastrar, editar, excluir e aprovar notícias.

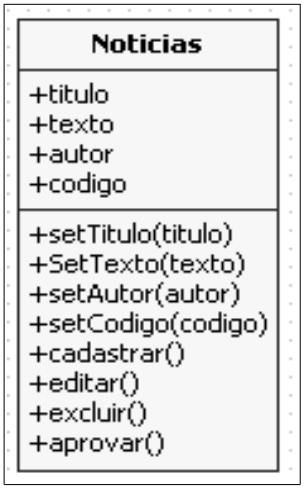


4.1.2 Diagrama de classes

O diagrama de classes tem como funções informar visualmente quais serão as classes que um determinado sistema irá conter, mostrar os atributos de cada classe e seus métodos, também deve informar os parâmetros que cada atributo receberá, além dos relacionamentos da classe, tanto associação quanto composição.

Exemplo:

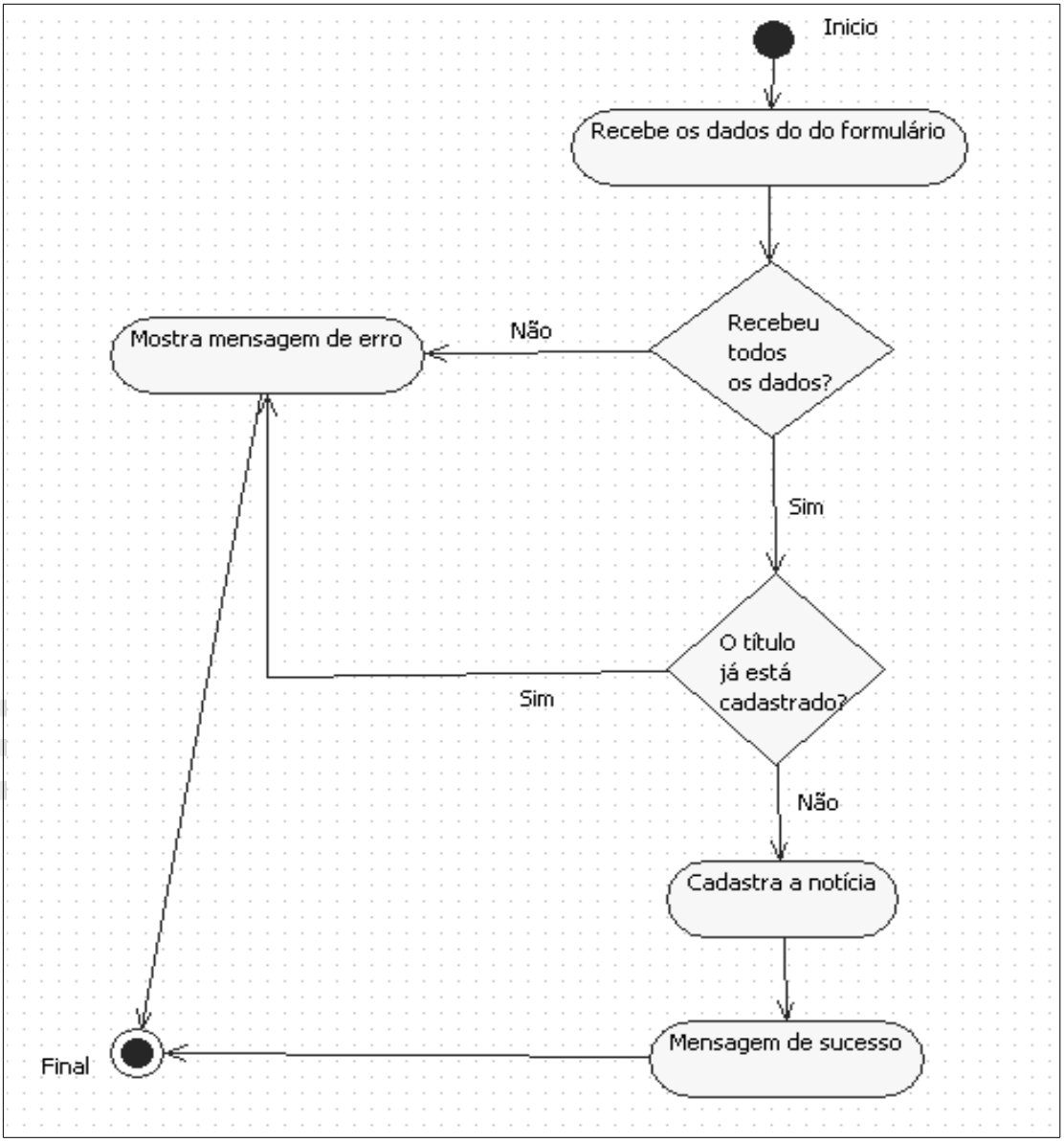
Vamos ver como ficaria o exemplo de uma classe de notícias.



4.1.3 Diagrama de atividades

O diagrama de atividades é empregado para descrever cada uma das ações apresentadas no diagrama de caso de uso.

Nesse diagrama estarão a descrição de como cada ação tomada funcionará e os passos para que o processo seja concluído.



4.1.4 Diagrama de sequência

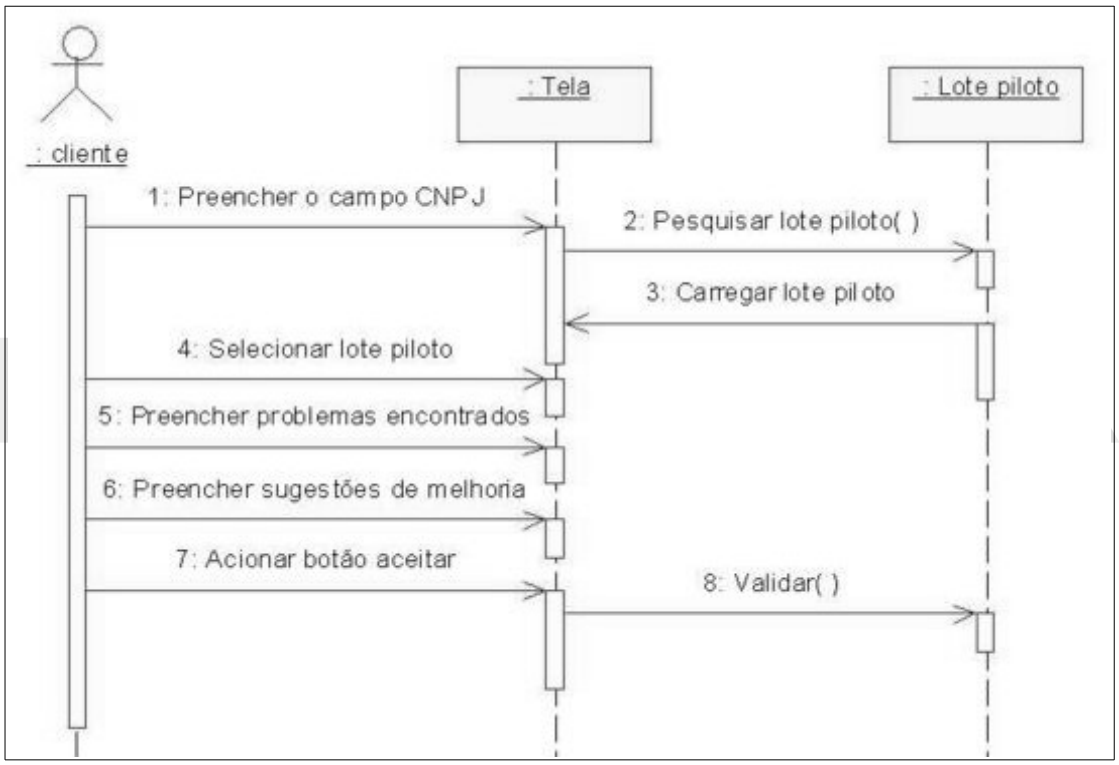
O diagrama de sequência é utilizado para conhecer a sequência dos processos realizados. Uma vez que um projeto pode possuir uma grande quantidade de métodos em classes diferentes, pode ser difícil determinar a sequência global do comportamento.

O diagrama de sequência representa essa informação de maneira simples e lógica, além de descrever o modo como os grupos de objetos colaboram em algum comportamento, ao longo do tempo. Ele registra o comportamento de um único caso de uso e exibe os objetos e as mensagens passadas entre esses objetos no caso de uso.

Em síntese: o diagrama de sequência é uma das ferramentas UML usadas para representar interações entre objetos de um cenário, realizadas através de operações ou métodos (procedimentos ou funções). Esse diagrama é construído a partir do diagrama de caso de uso. Primeiro se define qual o papel do sistema (*use case*), depois como o *software* realizará o seu papel (sequência de operações).

O diagrama de sequência dá ênfase à ordenação temporal em que as mensagens são trocadas entre os objetos de um sistema.

Entende-se por “mensagens” os serviços solicitados de um objeto a outro e as



respostas desenvolvidas para as solicitações.

4.2 Exercícios

Vamos pensar em um sistema que deverá ter três usuários interagindo com ele. O usuário **administrador** terá acesso completo ao sistema, ou seja, tudo que o gerente poderá fazer e, ainda, cadastrar usuários; o usuário **gerente** poderá cadastrar produtos, visualizar as vendas e realizar vendas; o usuário **vendedor** só poderá realizar vendas.

a) Faça o diagrama de caso de uso.

b) Faça o diagrama de atividades.

c) Faça o diagrama de classes.

CAPÍTULO 5

ENVIANDO E-MAILS

5.1 Enviando E-mails de Forma Simples e Fácil

O PHP dispõe do recurso de envio de e-mails. Esse envio pode ser realizado com textos simples, textos com formato HTML, vários destinatários e até com arquivos em anexo. Tudo isso é feito através da função **mail**, que é uma função interna do PHP.

```
mail($destinatario,$assunto,$mensagem);
```

Obs.: *Para que seja possível enviar e-mails é necessário que o servidor onde o PHP está instalado tenha um software SMTP, responsável pelo envio.*

Exemplo:

```
$destinatario = "email_destino@dominio.com.br";
$assunto = "Teste de envio de e-mail pelo PHP";
$cabecalho = "From: email_remetente@dominio.com.br \n";
$cabecalho .= "MIME-Version: 1.0\n";
$cabecalho .= "Content-type: text/html; charset=utf-8\n";
$mensagem  = "Esta é uma mensagem enviada pelo PHP <br />";
$mensagem .= "<b>Veja</b> <i>que tags funcionam aqui</i> HTML";
```

mail($destinatario, $assunto, $mensagem, $cabecalho) or die("Houve um problema com o envio de e-mail");

Dica: Você pode manipular todos os cabeçalhos do e-mail a serem enviados. Veja mais em br.php.net/manual/pt_br/function.mail.php

5.2 Enviando E-mails com a Classe Mail do PEAR

A classe Mail fornece diversas interfaces de envio de e-mail. A seguir veremos como mandar e-mails via SMTP (Simple Mail Transfer Protocol) autenticado.

Dica: *PEAR é um sistema de distribuição de componentes PHP reutilizáveis. Veja mais em http://pear.php.net.*

Exemplo utilizando a classe Mail do PEAR:

```
require_once (“Mail.php”); // inclui a classe Mail

// prepara os dados do email
$de = “remetente@dominio.com.br”;
$para = “destinatario@dominio.com.br”;
$assunto = “Teste de envio de e-mail via PHP”;
$mensagem = “Este e-mail foi enviado com a classe Mail do PEAR.”;

// prepara os dados do envio
$servidor = “smtp.dominio.com.br”;
$usuario = “site@dominio.com.br”;
$senha = “senha”;

$destinatarios = $para; // usa variável de apoio para destinatários

// monta o cabecalho da mensagem
$cabecalho = array(“From” => $de, “To” => $para, “Subject” => $assunto);

// passa os dados para o preparo do envio
$smtp = Mail::factory(“smtp”, array(“host”=>$servidor, “auth”=>true,
“username”=>$usuario, “password”=>$senha));

$mail = $smtp->send($destinatarios, $cabecalho, $mensagem); // envia o email

// testa o retorno, e dá mensagem de sucesso ou falha
if(PEAR::isError($mail)){
      echo $mail->getMessage();
}else{
      echo “Mensagem enviada”;
}
```

5.2.1 E-mails para vários destinatários

Para adicionar mais de um destinatário bastará informá-los ao setar a variável que tem os endereços de destino, conforme mostrado a seguir.

```
$para = “destinatario1@dominio.com.br, destinatario2@dominio.com.br,
destinatario3@dominio.com.br”;
```

5.2.2 E-mails com cópias

Para definir cópias do e-mail, bastará criar uma variável que contenha todos os endereços que deverão receber a cópia da mensagem e adicionar aos destinatários.

Exemplo:

```
require_once(“Mail.php”);


$de = “remetente@dominio.com.br”;
$para = “destinatario@dominio.com.br”;
$copia = “copia@dominio.com.br”;
$assunto = “Teste de envio de e-mail com cópia”;
$mensagem = “Este e-mail foi enviado com a classe Mail do PEAR.”;


$servidor = “smtp.dominio.com.br”;
$usuario = “site@dominio.com.br”;
$senha = “senha”;


$destinatarios = $para.”,”.$copia;


$cabecalho = array (“from” => $de, “To” => $para, “Cc” => $copia, “Subject”
=> $assunto);
$smtp = Mail::factory (“smtp”, array(“host” => $servidor,
“username”=>$usuario, “password” => $senha));


$mail = $smtp->send($destinatarios, $cabecalho, $mensagem);


if(PEAR::isError($mail)){
      echo  $mail->getMessage();
}else{
      echo ”Mensagem enviada”;
}
```

5.2.3 E-mails com cópias ocultas

Cópias ocultas do e-mail também podem ser enviadas a partir da classe Mail. É semelhante à cópia simples, a única diferença é que não é preciso colocar a cópia oculta no cabeçalho, somente adicionar a variável, que contém os endereços que devem recebê-la, aos destinatários, conforme exemplificado abaixo.

Exemplo:

```
require_once("Mail.php");


$de = "remetente@dominio.com.br";
$para = "destinatario@dominio.com.br";
$copiaoculta = "copiaoculta@dominio.com.br";
$assunto = "Teste de envio de e-mail com cópia";
$mensagem = "Este e-mail foi enviado com a classe Mail do PEAR.";


$servidor = "smtp.dominio.com.br";
$usuario = "site@dominio.com.br";
$senha = "senha";


$destinatarios = $para.",".$copiaoculta;


$cabecalho = array ("from" => $de, "To" => $para, "Subject" => $assunto);
$smtp = Mail::factory ("smtp", array("host" => $servidor,
"username"=>$usuario, "password" => $senha));


$mail = $smtp->send($destinatarios, $cabecalho, $mensagem);


if(PEAR::isError($mail)){
      echo  $mail->getMessage();
}else{
      echo "Mensagem enviada";
}
```

Obs.: *Não colocamos a variável com o endereço que deve receber a cópia oculta no cabeçalho, já que esse endereço deve permanecer em sigilo.*

5.2.4 E-mails com anexos

Poderemos enviar anexos com o e-mail, bastando para isso ter mais uma classe do PEAR instalada, a Mail_Mime, que permite a criação de e-mails complexos. Veremos a seguir como enviar um anexo por e-mail.

```
require_once("Mail.php");
require_once("Mail/mime.php");

$de = "remetente@dominio.com.br";
$para = "destinatario@dominio.com.br";
$assunto = "Teste de envio de e-mail com anexo";
$mensagem = "Este e-mail foi enviado com a classe Mail do PEAR, com anexo";

$servidor = "smtp.dominio.com.br";
$usuario = "site@dominio.com.br";
$senha = "senha";

$destinatarios = $para;

// adiciona mensagem e anexo
$corpo = new Mail_mime();
$corpo->setTXTBody($mensagem);
$corpo->addAttachment("imagem.png");

$mensagem = $corpo->get();

// monta cabeçalhos
$cabecalho_extra = array ("from" =>$de, "subject" => $assunto);
$cabecalho = $corpo->headers($cabecalho_extra);

$mail = Mail::factory("mail");
```

PHP

```
$mail->send($destinatarios, $cabecalho, $mensagem);


is (PEAR:: isError ($mail)){
      echo $mail->getMessage();
}else{
      echo “Mensagem enviada”;
}
```

CAPÍTULO 6

XML

6.1 O que é XML?

XML (Extensible Markup Language) é uma linguagem de marcação de dados (meta-markup language) que provê um formato para descrever dados estruturados. Com isso, fica muito mais fácil fazer declarações mais precisas dos conteúdos e resultados de busca, através de múltiplas plataformas.

No XML poderemos criar quantas tags forem necessárias, enquanto no HTML temos um número limitado delas.

Qualquer elemento XML pode conter dados declarados como: preços de venda, idade, até mesmo o endereço de um amigo, enfim, qualquer tipo de dados.

Já que a principal funcionalidade do XML é transportar dados de um sistema para outro, deve haver nele a habilidade para manipular e procurar dados. Porém, quando o registro é encontrado, ele pode ser importado para um banco de dados ou mostrado diretamente no navegador, mas isso depende da necessidade de cada sistema.

6.2 Comparação entre o HTML e o XML

HTML e XML são “primos”. Derivam da mesma inspiração, o SGML. Ambos identificam elementos em uma página e utilizam sintaxes similares.

Caso possua familiaridade com HTML, terá igualmente com o XML. O que diferencia os dois é que o HTML descreve a aparência e as ações em uma página na rede, enquanto o XML não descreve nem aparência e ações, mas sim o que cada trecho de dados é ou representa. Em outras palavras, o XML descreve o conteúdo do documento.

Assim como o HTML, o XML também faz uso de tags e atributos, mas enquanto o HTML especifica cada sentido para as tags e atributos, o XML emprega as tags somente para delimitar trechos de dados e deixa a interpretação do dado a ser realizada completamente para a aplicação que o está lendo. Sintetizando, enquanto em um documento HTML uma tag <p> indica um parágrafo, no XML essa tag pode indicar um preço, um parâmetro, uma pessoa ou qualquer outra coisa que se possa imaginar (inclusive algo que nada tem a ver com um p, como por exemplo, autores de livros).

Os arquivos XML são arquivos texto, mas não destinam-se à leitura por um ser humano como o HTML. Os documentos XML são arquivos texto, auxiliam os programadores ou desenvolvedores, fazendo com que "debuguem" mais facilmente as aplicações, de forma que um simples editor de textos possa ser usado para corrigir um erro em um desses arquivos. Mas as regras de formatação para documentos XML são muito mais rígidas do que para documentos HTML. Uma tag esquecida ou um atributo sem aspas torna o documento inutilizável, enquanto que no HTML isso é tolerado. As especificações oficiais do XML determinam que as aplicações não podem tentar adivinhar o que está errado em um arquivo (no HTML isso acontece), mas devem parar de interpretá-lo e reportar o erro.

6.3 Características do XML

O XML provê uma representação estruturada dos dados que mostrou ser amplamente implementável e fácil de ser desenvolvida.

Implementações industriais na linguagem SGML (Standard Generalized Markup Language) apresentaram a qualidade intrínseca e a força industrial do formato estruturado em árvore dos documentos XML.

O XML é um subconjunto do SGML, o qual é otimizado para distribuição através da web, e é definido pelo W3C, assegurando que os dados estruturados serão uniformes e independentes de aplicações e fornecedores. Além disso, o XML provê um padrão que pode codificar o conteúdo, as semânticas e as esquematizações para uma grande variedade de aplicações, desde as simples até as mais complexas, dentre elas:

- Um simples documento.
- Um registro estruturado, tal como uma ordem de compra de produtos.
- Um objeto com métodos e dados, como objetos Java ou controles ActiveX.
- Um registro de dados.
- Apresentação gráfica.
- Entidades e tipos de esquema padrão.
- Todos os links entre informações e pessoas na web.

Uma característica importante é que, uma vez tendo sido recebido o dado pelo cliente, tal dado pode ser manipulado, editado e visualizado, sem que seja preciso acionar novamente o servidor. Dessa forma, os servidores têm menor sobrecarga, reduzindo a necessidade de computação e reduzindo também a requisição de banda passante para as comunicações entre cliente e servidor.

O XML é considerado de grande relevância na Internet e em grandes intranets porque provê a capacidade de interoperação dos computadores, pois possui um padrão flexível, aberto e independente de dispositivo. As aplicações podem ser construídas e atualizadas mais rapidamente e, também, permitem múltiplas formas de visualização dos dados estruturados.

6.4 Separação entre Dados e Apresentação

A mais importante característica do XML se resume em separar a interface com o usuário (apresentação) dos dados estruturados.

O HTML especifica como o documento deverá ser apresentado na tela por um navegador. Já o XML define o conteúdo do documento. Por exemplo, em HTML são utilizadas tags para definir tamanho e cor de fonte, assim como formatação de parágrafo. No XML emprega-se as tags para descrever os dados, como por exemplo, tags de assunto, título, autor, conteúdo, referências, datas, entre outros.

O XML ainda conta com recursos como folhas de estilo definidas com Extensible Style Language (XSL) e Cascading Style Sheets (CSS) para a apresentação de dados em um navegador e separa os dados da apresentação e processo, o que permite visualizar e processar o dado como se queira, usando distintas folhas de estilo e aplicações.

Essa separação dos dados da apresentação possibilita a integração dos dados de diversas fontes. Informações de consumidores, compras, ordens de compra, resultados de busca, pagamentos, catálogos etc, podem ser convertidas para XML no middl-tier (espécie de servidor), permitindo que os dados sejam trocados online, tão facilmente como as páginas HTML mostram dados hoje em dia. Dessa forma, os dados em XML podem ser distribuídos através da rede para os clientes que desejarem.

Exemplo de um arquivo XML:

```
<?xml version=”1.0” encoding=”utf-8” ?>
<clientes>
      <cliente>
            <nome>José</nome>
            <estado>Paraná</estado>
            <cidade>Curitiba</cidade>
      </cliente>
      <cliente>
            <nome>João</nome>
            <estado>São Paulo</estado>
            <cidade>Santos</cidade>
      </cliente>
```

```
</clientes>
```

6.5 SimpleXML

A SimpleXML é uma biblioteca que está inclusa no PHP desde a versão 5 e, sem dúvida alguma é a maneira mais fácil de manipular arquivos XML.

A forma da SimpleXML manipular um arquivo XML é muito similar à maneira com que o DOM manipula, isto é, ela carrega todo o arquivo XML para a memória e monta uma árvore, o que torna a manipulação do arquivo muito mais simples e rápida.

Contudo, ao utilizá-la, deve-se tomar alguns cuidados, conforme listados a seguir.

- O arquivo XML deverá ter uma sintaxe perfeita. A SimpleXML não conseguirá manipular o arquivo caso tenha algum erro de sintaxe no XML, como por exemplo, esquecer de fechar uma tag.
- A SimpleXML só consegue trabalhar com a versão 1.0 do XML.
- O arquivo XML deverá ter uma sintaxe perfeita. A SimpleXML não conseguirá manipular o arquivo caso tenha algum erro de sintaxe no XML, como por exemplo, esquecer de fechar uma tag.
- A SimpleXML só consegue trabalhar com a versão 1.0 do XML.

Exemplo usando a SimpleXML:

Criaremos um arquivo XML:

```
<?xml version="1.0" encoding="utf-8"?>
<recado>
<para>José</para>
<de>João</de>
<assunto>Lembrete</assunto>
<mensagem>Não esqueça da reunião na Sexta-feira</mensagem>
</recado>
```

E agora usaremos a SimpleXML para ler o arquivo XML criado acima, conforme se verifica abaixo.

```
<?php
$xml = simplexml_load_file("arquivo.xml"); // carrega o arquivo
```

PHP

```
echo $xml->getName() . "<br />"; // imprime nome da tag, nesse caso: recado


foreach($xml->children() as $filho)
{
      // imprime o nome da tag e seu conteúdo
      echo $filho->getName() . ": " . $filho . "<br />";
}
?>
```


ELABORATA
INFORMÁTICA
CAPÍTULO 7
PDF

PHP

7.1 Introdução

O PHP, assim como muitas linguagens, tem o poder de gerar arquivos PDF.

Existem algumas alternativas para a geração dos arquivos, umas pagas e outras gratuitas. O PHP não consegue gerar arquivos PDF nativamente, ou seja, será necessária uma biblioteca externa para fazer isso.

7.2 FPDF

FPDF é uma biblioteca para a geração de arquivos PDF gratuita. Podemos fazer o *download* no seu site oficial, que é o www.fpdf.org.

FPDF significa *Free PDF* e, apesar de estar descontinuada desde 2005, ainda é a melhor alternativa para a geração de arquivos gratuitamente. Além disso, possui muitos recursos, entre os quais podemos citar:

- Possibilidade de selecionar o tipo e o tamanho da fonte que será utilizada.
- Implementação de cabeçalhos e rodapé automaticamente.
- Realiza quebra de página automaticamente.
- Realiza quebra de linha automaticamente.
- Realiza justificação de texto automaticamente.
- Suporta inserção de imagens.
- Seleção de cores para texto e fundo.
- Inserção de links.

7.2.1 Criando o primeiro exemplo com o uso do FPDF

Iremos gerar um arquivo PDF para ver o funcionamento da biblioteca.

Para poder executar o exemplo, faça o *download* da última versão no site oficial e descompacte o arquivo para uma pasta chamada **fpdf** dentro da pasta que está configurada para ser o servidor WEB.

Todo o texto que estiver contido em um arquivo PDF deve estar dentro de uma célula.

```
require_once("fpdf/fpdf.php"); // inclui a classe fpdf

$pdf = new FPDF(); // cria o objeto para a classe FDPF

$pdf->AddPage(); // cria uma nova página

$pdf->SetFont("Arial", "B", 20); // define a fonte que será utilizada

// cria uma célula e escreve um texto dentro da célula
$pdf->Cell("0", 15, "Curso de PHP - Elaborata Informática");

$pdf->Output(); // gera o arquivo
```

7.2.2 Implementando cabeçalho e rodapé

Conforme mencionado anteriormente, o FPDF consegue gerenciar cabeçalhos e rodapés automaticamente. Para que isso seja possível foram criados os métodos **Header** e **Footer** sem nenhum conteúdo, permitindo a implementação, de acordo com a necessidade de cada um.

Então, criaremos uma classe e faremos uma herança com a classe FPDF.

```
<?php
require_once('fpdf/fpdf.php'); // inclui a classe fpdf

class PDF extends FPDF
{
	// função que cria o cabeçalho
	function Header()
	{
		// inclui imagem
		$this->Image('logo.gif',10,6,30);
		// Arial bold 15
		$this->SetFont('Arial','B',15);
		// alinha è direita
		$this->Cell(80);
```

```
            // inclui um título
            $this->Cell(30,10,'Titulo',1,0,'C');
            // quebra a linha
            $this->Ln(20);
      }

      // função que cria o rodapé
      function Footer()
      {
            // posiciona a 1,5cm do fim da pagina
            $this->SetY(-15);
            // Arial italic 8
            $this->SetFont('Arial','I',8);
            // número da pagina
            $this->Cell(0,10,'Page '.$this->PageNo().'/{nb}',0,0,'C');
      }
}

// cria uma instância da classe
$pdf = new PDF();
$pdf->AliasNbPages();
$pdf->AddPage();
$pdf->SetFont('Times','',12);
for($i=1;$i<=40;$i++)
    $pdf->Cell(0,10,'Imprimindo o número da linha '.$i,0,1);
$pdf->Output();
?>
```

Dica: *O FPDF possui um guia de funções totalmente em português, portanto, faça o* download *e conheça todas as funcionalidades que a biblioteca possui.*

CAPÍTULO 8

GRÁFICOS

8.1 Geração de Gráficos

O PHP consegue gerar gráficos (imagens) bem interessantes, mas não o faz sozinho. É preciso ter instalada/habilitada a extensão GD.

Uma vez que estamos utilizando o WAMP, esse já vem habilitado por padrão. Caso esteja usando um sistema operacional Linux, pode instalar o pacote da biblioteca, como no exemplo:

```
aptitude install php5-gd
```

A biblioteca possui todos os recursos necessários para a geração de gráficos, porém, usar diretamente a GD para gerá-los é muito trabalhoso, por isso empregaremos a biblioteca **JPGraph**, que já possui inúmeros gráficos prontos. Bastará fazer o download, consultar o exemplo apresentado e utilizar. Vale lembrar que a **JPGraph** possui restrições de licença para uso comercial.

Vejamos um exemplo de imagem sendo gerada pela biblioteca **JPGraph**.

```
<?php
// inclui a biblioteca jpgraph
require('jpgraph/src/jpgraph.php');
require('jpgraph/src/jpgraph_line.php');

// monta vetor com dados aleatórios
$ydata = array(11,3,8,12,5,1,9,13,5,7);
// cria o gráfico
// estas duas linhas são obrigatórias
$graph = new Graph(350,250);
$graph->SetScale('textlin');
// coloca um título do gráfico
$graph->title->Set('Exemplo de gráfico');
// cria um gráfico do tipo linear
$lineplot=new LinePlot($ydata);
$lineplot->SetColor('blue');
// adiciona tipo de gráfico
$graph->Add($lineplot);
// exibe o gráfico
```

```
$graph->Stroke();

?>
```

CAPÍTULO 9

REFERÊNCIAS BIBLIOGRÁFICAS

9.1 Bibliografia

- MELO, Alexandre A.; NASCIMENTO, Mauricio G. F. PHP Profissional. 1. ed. São Paulo: Novatec, 2008.

9.2 Material Disponível Online

- http://en.wikipedia.org
- http://php.net/
- http://pear.php.nhttp://www.fpdf.org/
- http://jpgraph.net/
- http://www.omg.org/
- http://mysql.com/
- http://pear.php.net/manual/en/standards.php

Rua Monsenhor Celso, 256 - 1º andar - Curitiba – Paraná

41.3324.0015

41.99828.2468

Agradecimentos:

Equipe Elaborata Informática