Fábio Burch Salvador

Programando em PHP

Integração com MySQL

editora
VIENA
2ª Edição
Bauru/SP
Editora Viena
2012

Sumário

Lista de Siglas e Abreviaturas

⮌	*Indica que a linha de comando atual e a de cima são uma única linha de comando.*
CGI	*Common Gateway Interface.*
GIF	*Graphics Interchange Format.*
HTML	*Hypertext Markup Language.*
JPEG	*Joint Photographic Experts Group.*
PC	*Personal Computer.*
PHP	*Hypertext Preprocessor.*
PNG	*Portable Network Graphics.*
SGBD	*Sistemas Gerenciadores de Banco de Dados.*
SQL	*Structured Query Language.*
URL	*Uniform Resource Locator.*

1
Antes de Começar

1. Antes de Começar

Antes de falarmos sobre a linguagem **PHP**, vamos falar um pouco sobre programação. Este livro parte dos seguintes pressupostos:

- O leitor sabe pelo menos o básico sobre lógica da programação, é capaz de construir algoritmos funcionais e consegue entender o uso de laços de repetição, expressões condicionais e outros conceitos básicos do desenvolvimento em linguagens de programação.
- O leitor tem um conhecimento, pelo menos básico, sobre o que seja um banco de dados, como funcionam e para quê servem.

Dito isso, vamos a alguns lembretes, sempre úteis, que todo programador deveria anotar para nunca esquecer.

1.1. Lembre-se de Identar o Código

Em programação, usamos um termo, **"identar o código"**, que descreve uma prática essencial para qualquer programador de qualquer linguagem e que, a princípio, pode parecer artificial e difícil de acostumar, mas com o tempo, vira um hábito. Identar é, numa descrição prática, usar a tecla **<Tab>** para colocar as linhas de código que pertencem a um mesmo **"nível de profundidade"** dentro da rotina, no mesmo alinhamento horizontal, colocando os códigos que formam a parte interna de um bloco **"mais para a direita"**.

Parece complicado? Vamos um exemplo prático. Vou construir um trecho de código que faz a seguinte análise: o usuário digitou sua idade e ela foi carregada em uma variável X, então, se o sujeito for maior de idade, ele já pode dirigir. Se ele for menor, ainda haverá outra condicional dentro, para ver se ele tem 16 anos ou não.

O código deste trecho, sem identação:

```
IF ($X>=18) {
  Echo 'Parabéns, você já é maior de idade e pode dirigir!';
} else {
  IF ($X>=16) {
  Echo 'Você ainda não pode dirigir, mas já pode votar!';
} else {
  Echo 'Você é um pirralho!';
}
}
```

Note que o código, sem nenhuma formatação nas linhas, torna-se confuso para ler pois é complicado compreender a hierarquia dos blocos de expressão condicional. Na realidade, os **"ifs"** que têm relação com o fato de o usuário ter mais ou menos de 16 anos, estão **"dentro"** de um **"IF"** maior, que é aquele no qual descobrimos se ele é maior ou menor de idade. Mas, visualmente, não temos nenhuma evidência disso.

Assim, embora pareça simples de construir o código, a manutenção deste programa tornar-se-á muito difícil no futuro. Não especificamente no caso do exemplo dado, pois ele é muito simples. Mas em programas maiores, a falta de identação poderá se tornar um problema sério.

Agora, vejamos o mesmo exemplo, devidamente identado:

```
IF ($X>=18) {
    Echo 'Parabéns, você já é maior de idade e pode dirigir!';
} else {
  IF ($X>=16) {
    Echo 'Você ainda não pode dirigir, mas já pode votar!';
  } else {
    Echo 'Você é um pirralho!';
  }
}
```

Note que é possível enxergar, visualmente o quê está "dentro" do quê. A prática da identação de código é amplamente vista como um requisito básico da boa programação. Na maioria das linguagens, é uma boa prática basicamente pelo efeito de auxílio visual para a manutenção do código. Mas em algumas linguagens, como **COBOL**, para citarmos um exemplo clássico, a posição horizontal (de coluna) do início de uma determinada linha de código tem importância real na hora de o compilador interpretar como aquela parte do código deverá ser executada. Para citarmos exemplos mais modernos, temos linguagens, como **Python, Occam** e **Haskell**, nas quais as instruções de início de bloco, em alguns casos, é substituída pela identação dada ao código, tendo esta, portanto, uma função prática e real na maneira como o código será executado.

1.2. Use Comentários

Comentários dentro de um código são essenciais. Talvez não para o desenvolvedor original deste código, mas para os profissionais que darão a manutenção a ele no futuro. Aliás, o próprio programador original poderá se beneficiar dos comentários, uma vez que eles não servem apenas para fazer marcações sobre o conteúdo do programa, mas servem, também, para deixar ao longo deste uma série de dicas e lembretes que poderão ser essenciais diante da necessidade de fazer alterações ao código.

Mas, para os completamente leigos, permanece a dúvida: o que é um comentário, em programação? Um comentário é uma linha que, tendo uma determinada marcação como tal, não será executada pelo compilador ou interpretador na hora da execução do programa. É uma linha "morta".

Este recurso é muito utilizado, por exemplo, quando se desenvolve um determinado pedaço de programa que executa uma determinada função e, então um dia, resolve-se desativar aquela função. Embora o procedimento mais óbvio para eliminar o pedaço indesejado de código seja simplesmente deletá-lo, isso representa um novo problema, á medida que aquele trecho de programa muitas vezes representa horas de trabalho e complexos exercícios de raciocínio por parte do desenvolvedor. Perdê-lo significa perder a mesma quantidade de tempo e esforço no futuro, caso aquela função eliminada volte a ser necessária.

Neste caso, a melhor alternativa é transformar todo o trecho deletável de código em um imenso comentário, sem função ativa, mas pronto para ser ressuscitado em caso de necessidade futura.

A função primordial dos comentários, no entanto, não é colocar "no limbo" pedaços de programa. É possibilitar que o programador deixe anotações em meio ao código.

Existem comentários de uma linha só, demarcados com um sinal (// ou #) no início daquela linha. Um rápido exemplo funcional:

```
<?php
  echo 'teste 1 <br/>';
  # echo 'teste 2 <br/>';
  // echo 'teste 3 <br/>';
?>
```

No exemplo acima, o que veremos no navegador será, apenas a linha que diz "teste 1". As demais não valem nada, em termos de execução.

Temos também a possibilidade de criar blocos de comentários com várias linhas, usando símbolos de início (/*) e final (*/). Um exemplo:

```
<?php
  /*
  Programa exemplo do PHP
  Programador: Fábio Salvador
  Data: 16/07/2011
  Local: Viamão/RS
  */
  echo 'Olá mundo! <br />';
  $ano = date('Y');
  echo 'Estamos no ano de '.$ano;
?>
```

Neste programa, temos quatro linhas com os dados de identificação do programa-exemplo. Estas linhas, no entanto, estão entre os símbolos de início e fim do bloco de comentários. Portanto, não serão executadas. Mas poderão ser lidas, por exemplo, quando alguém for fazer manutenção no site. Como as linhas de texto com os dados da criação do programa estão "comentados", o que será visto na tela do navegador será:

```
Olá mundo!
Estamos no ano de 2012
```

O uso de comentários, no entanto, como eu já disse, não se restringe à função de colocar marcações de autoria do programa. Diante do uso de variáveis cuja utilidade não seja óbvia à primeira vista, é interessante colocar, junto ao seu primeiro uso, um comentário dizendo para quê servem. Assim como laços de repetição muito longos, que deverão ser comentados a cada passo para que, na fase de testes, a equipe não se perca com relação à pontos de travamento do programa. Ou, melhor, não crie novos pontos com problemas em manutenções futuras, pela simples desinformação a respeito da utilidade de certas linhas e pedaços do programa.

1.3. Lembre-se de Documentar

A prática de documentar sistemas, utilizando ou não as regras da **UML** (sim, pois sejamos francos: existem muitas empresas que não utilizam nenhum sistema de documentação universalmente aceito, e preferem criar suas próprias ideias mirabolantes), de qualquer forma, documentar o funcionamento dos sistemas é fundamental para que se possa trabalhar ao longo do tempo.

Embora muitas aplicações em **PHP** consistam em sites construídos por uma ou duas pessoas e, com isso, pareça evidente que uma documentação é inútil, o fato é que o próprio desenvolvedor original do sistema pode se perder num momento qualquer do futuro. Documentar o desenvolvimento de um sistema tem como vantagem, ainda, a possibilidade de delegar a tarefa de aperfeiçoamento dele para outras pessoas, e nunca se sabe o dia de amanhã. A empresa pequena, tocada pelo dono ou por uma dupla ou trio de amigos poderá, amanhã, ter empregados e estagiários. E aí, será preciso decifrar o funcionamento dos sistemas usando engenharia reversa, o que sempre é mais demorado e trabalhoso.

Em alguns casos, sistemas que sofrem alterações com certa frequência podem tornar-se, ao longo do tempo, tão complexos e intrincados que, na ausência de documentação, comentários e esclarecimentos, seja praticamente incompreensível. E aí, não tem jeito: já vi ocasiões em que a tarefa de decifrar o funcionamento de um código apresentou-se tão complicada que foi mais fácil reconstruir um programa do zero, usando o original apenas como inspiração.

Observação: *Fugindo um pouco do tema central do livro, fica uma dica aos programadores iniciantes: embora pareça que escrever um programa complicado e com documentação deficiente é uma tática muito "esperta", já que teoricamente garante a continuidade em um emprego (a empresa precisa de você para dar manutenção aos sistemas), a verdade é que programação complicada e não explicada é algo que irrita os colegas e superiores. E normalmente, para evitar que sistemas futuros tenham o mesmo "defeito", os analistas de sistemas optam por livrar-se do "espertinho", mesmo que isso venha a custar algumas horas ao resto da equipe, ou até a reconstrução, desde o início, de alguns bons programas. Ou seja: "amarrar" o código, e torná-lo deliberadamente um enigma, não trás benefício algum.*

1.4. História

Agora falemos de **PHP.** Antes da parte prática (e divertida), um pouco de teoria...

O nome da linguagem é uma abreviação de "**PHP: Hypertext Preprocessor**", que faz alusão à sua lógica de funcionamento: o programa, interpretado no servidor, pré-processa informação, gerando um código **HTML** (hipertexto), que será visualizado pelo cliente. **PHP** foi criada em 1994, por um programador groenlandês chamado Rasmus Lerdorf. Na verdade, inicialmente, era apenas uma série de scripts em **Perl,** que Lerdorf escreveu para usar em sua página pessoal na então ainda nascente Internet. Ele deu a este conjunto de scripts o nome de **Personal Home Page Tools**. Era algo simplório: o sistema tinha como funções, coisas hoje básicas, como mostrar seu curriculum vitae e contar o número de visitas na página.

Depois, ele reescreveu esses scripts em **C,** criando uma **CGI** básica. Só que esta nova versão, turbinada, era capaz de interagir com formulários **HTML** e com bancos de dados. Esta versão ganhou o nome de **Personal Home Page/Forms Interpreter** (**PHP/FI**). A versão 1.0 saiu em 1995 e, já em 1997, uma segunda versão ganhou o mundo.

Foi em 1997 que dois programadores israelenses, Zeev Suraski e Andi Gutmans fizeram algumas modernizações fundamentais às linguagens e mudaram seu nome para simplesmente **PHP**. Em 1998, foi oficialmente lançada a terceira versão da linguagem.

Em 2000, foi lançada a versão **PHP 4** e, em 2004, a quinta versão. Esta, com um suporte melhor quanto à orientação a objetos, melhoramentos na relação com bancos de dados e melhorias na performance de execução.

Hoje estamos já na era do **PHP 6**, e é curioso notar que esta linguagem, na metade da primeira década do século XXI, parecia fadada ao desaparecimento. Isso mesmo: as publicações especializadas falavam de um "fim" do **PHP**, ou na permanência desta como uma linguagem de "segunda classe" (aliás, também apostavam no desaparecimento do **JavaScript**), apostando que o futuro seria dos sites em **Flash**, ou de aplicações e sites desenvolvidos em outras linguagens, como **Java Enterprise Edition**. Mas as novas versões de **PHP** passaram a apresentar recursos que davam muito mais possibilidades aos programadores e o mundo virtual caminhou numa direção diferente daquela que se esperava, com tendências por buscar padronizações, linguagens e aplicações de domínio público. Assim, no final da mesma década, **PHP** e **JavaScript** passaram a ser as duas palavras mágicas no mundo da Web.

Existem razões para essa "volta por cima": sites em **Flash** são normalmente pesadas, o que atrapalha a navegação dependendo da conexão que o usuário tenha. Além disso, fazer rotinas em **Java2EE** é bem mais complicado e demorado do que construir as mesmas funções em **PHP**.

Agora, chega de história. Vamos ao trabalho.

1.5. É um Programa, é um Site

A primeira coisa que se precisa saber sobre a linguagem **PHP** é que as aplicações com ela desenvolvidas são processadas pelo servidor, e não pelo cliente.

Vamos conhecer melhor essa questão.

Para quem não sabe, um site é formado por um aglomerado de arquivos gravados em uma pasta específica de um computador que aqui chamaremos de **servidor de hospedagem**. Este computador possui instalado um programa servidor web (o mais famoso deles, hoje em dia, é o **Apache**). Este programa literalmente pega o conteúdo de uma pasta específica (que chamaremos de pasta **www**), e disponibiliza este conteúdo para acesso remoto.

Certo. Mas um simples montão de arquivos disponibilizados na rede não formam um site. Esses arquivos têm que ter características especiais. São arquivos contendo tags **HTML**, que dizem onde um trecho de texto deve ir na tela de um navegador, e onde vão aparecer as imagens, também hospedadas no servidor.

Vamos ao outro lado desta transação: o sujeito que está acessando estes arquivos remotos, não navega na pasta de arquivos em si. Porque o programa servidor disponibiliza o acesso apenas a um tipo de programa: o navegador ou browser **(Internet Explorer, Firefox**, **Chrome**, etc). Este programa, operando no computador do usuário, é capaz de ler e interpretar as tags **HTML** do código do site e exibir ao usuário a página de internet, como costumamos ver quando navegamos.

1.6. DNS e IP

Já vimos que há um computador hospedando o site, e que há um outro computador acessando ele. É óbvio que, para encontrar o site, o navegador do usuário deverá navegar até algum lugar, e este lugar é identificado pelo seu endereço IP.

Então, quer dizer que se quisermos abrir um site qualquer, vamos acessar o site sabendo o IP dele? Não. Ninguém digita o endereço IP dos servidores dos sites para acessá-los. As pessoas digitam um endereço, formado por palavras, normalmente começando por www. Como é que isso funciona?

Quando alguém cria um site, registra um domínio. Digamos que eu crie um site sobre Chevettes. Devo procurar o **Registro.BR**, e ver se está disponível o domínio que quero, no caso, **www.chevette.com.br.** Digamos que esteja, e que eu o registre. Quando passo as especificações técnicas para o **Registro.BR**, preciso dizer onde está o "guia" que vai conduzir o usuário até o meu site, e este guia é o servidor DNS, que sabe onde está hospedado o site.

Depois disso tudo pronto, o que acontece é o seguinte: alguém digita **www.chevette.com.br** no navegador e o programa faz uma requisição ao servidor DNS, perguntando "onde está esse site?". O servidor DNS responde com o endereço da hospedagem, permitindo que se navegue até ele.

1.7. Página Estática e Página Dinâmica

Nessa história toda, temos dois personagens: o cliente, que é o navegador, e o servidor, que é o programa rodando lá no computador que hospeda o site. O servidor envia um monte de código **HTML**, mais imagens, animações e o que mais o site tiver para exibir. O navegador mostra tudo organizado para o usuário.

Entendido isso, vamos olhar os sites, que podem ser estáticos ou dinâmicos. Um site estático é aquele que não há programa gerando código **HTML**, aquela página manjada que qualquer leigo faz em um programa como o **Dreamweaver**, e que exibe texto, foto e tudo como foi desenhado pelo desenvolvedor. Para alterar o conteúdo de uma página estática, o desenvolvedor dela precisa alterar o código **HTML** dela, abrir a pasta www, e sobrescrever estes arquivos velhos.

Uma página dinâmica é aquela cujo código **HTML** é, em parte, gerado pela execução de um programa. Para entender este conceito, vamos ver como funciona um blog: um blog é um site da Internet cujo mecanismo de funcionamento consiste em um programa e um banco de dados. Neste banco de dados, estão gravados textos, e o programa lê estes textos, posiciona tags **HTML** ao redor deles, e entrega a página montada ao navegador. Quando novos dados são adicionados à base de dados, o site muda, sem que o desenvolvedor precise mexer na pasta www, nos arquivos que fazem ele funcionar.

1.8. Programas Rodando na Web

Existem dois tipos de aplicações e de linguagens para fazer aplicações, as quais podemos usar para desenvolver tanto sites como programas que rodam via **Web** no navegador do usuário. Algumas linguagens geram programas que são interpretados

pelo navegador do cliente, e outras, programas que são interpretados pelo próprio programa servidor. As primeiras são chamadas de **client side** e as segundas, de **server side**.

1.8.1. Client Side

Códigos em linguagens como **JavaScript** são enviados "brutos" do servidor para o cliente, e são interpretados pelo navegador deste. Desta forma, o programa está rodando localmente na máquina do usuário e, portanto, pode "reagir" em tempo real às ações dele. Pequenos trechos deste tipo de linguagem fazem efeitos como, por exemplo, validar um campo de texto e verificar quantos caracteres um usuário já digitou nele, alertando o excesso. Ou menus interativos, que aparecem e somem quando o cursor do mouse passa por cima.

Os dois grandes problemas de programas interpretados pelo modo **client side** são:

- Como o código do programa é enviado bruto para o navegador, o usuário poderá ler este código, usando opções simple como, por exemplo, o **Exibir código-fonte**, do **Internet Explorer**. Isso impossibilita a colocação, por exemplo, de senhas seguras ao banco de dados, neste código.
- Quem desenvolve o site ou aplicação usa um navegador que interpreta bem a linguagem na qual ele está disponibilizando seus programas, mas nunca poderá ter certeza de que os clientes (navegadores), terão o mesmo interpretador. Assim, por exemplo, é possível escrever um código **JavaScript** que faz coisas incríveis no site, mas para um usuário (mal)equipado com um computador antigo, rodando um navegador pré-histórico, só o que aparecerá é uma imensa mensagem de erro.

1.8.2. Server Side

Este tipo de programa não tem o seu código-fonte enviado "bruto" para o navegador. O programa **server side** é sempre executado pelo servidor e, depois que roda e gera um código **HTML**, é apenas o **HTML** resultante que será com ótima aparência enviado para o cliente.

Um bom exemplo é o seguinte: um leitor entra no blog. O programa do blog, feito em **PHP**, é executado pelo servidor e esta execução consiste em ler os textos do banco de dados e montar um código **HTML** para exibir estes textos num layout de ótima aparência. Apenas o **HTML** pronto será enviado ao navegador e, não o código **PHP**. O usuário poderá ver o código-fonte da página à vontade, que não verá as linhas em **PHP**, apenas as tags **HTML** geradas pelo programa.

As duas grandes vantagens das linguagens **server side** são justamente o inverso dos defeitos das **client side**: é preciso que o servidor seja capaz de interpretar o código, sem se importar com os navegadores que estão do outro lado, e o usuário não consegue ler o código-fonte.

A desvantagem, claro, é que toda e qualquer interação com o usuário se dá numa transação cliente-servidor, sem aquele "dinamismo em tempo real" que temos com programas executando no navegador. Assim, por exemplo, para fazer um programa

que pede ao usuário que digite seu nome e dê uma resposta, é preciso fazer pelo menos quatro transações através da rede:

- O navegador requisita o conteúdo da página.
- O servidor devolve a página pedindo o nome do usuário.
- Após digitar, o usuário envia seu nome ao servidor.
- O servidor roda o programa em **PHP**, que pega o nome do sujeito e coloca a expressão "olá" antes, devolvendo a frase "Olá, Fulano" ao navegador, que exibirá a frase ao usuário.

1.9. Interpretação de Linguagens

Agora que entendemos o processo de criação, hospedagem e acesso a um site ou aplicação Web, devemos ter em mente que os programas-servidores podem receber a adição de "opcionais", como gerenciadores de bancos de dados. E interpretadores de linguagens de programação. Acontece que, com o tempo e o desenvolvimento das linguagens e sistemas de bancos de dados, e dos próprios servidores, estas coisas passam a vir agrupadas em "pacotes" prontos.

Assim, é de praxe que na instalação de qualquer versão minimamente moderna do **Apache**, por exemplo, tenhamos um interpretador de **PHP** e um gerenciador de base de dados **MySQL**, para dizer o mínimo.

Anotações

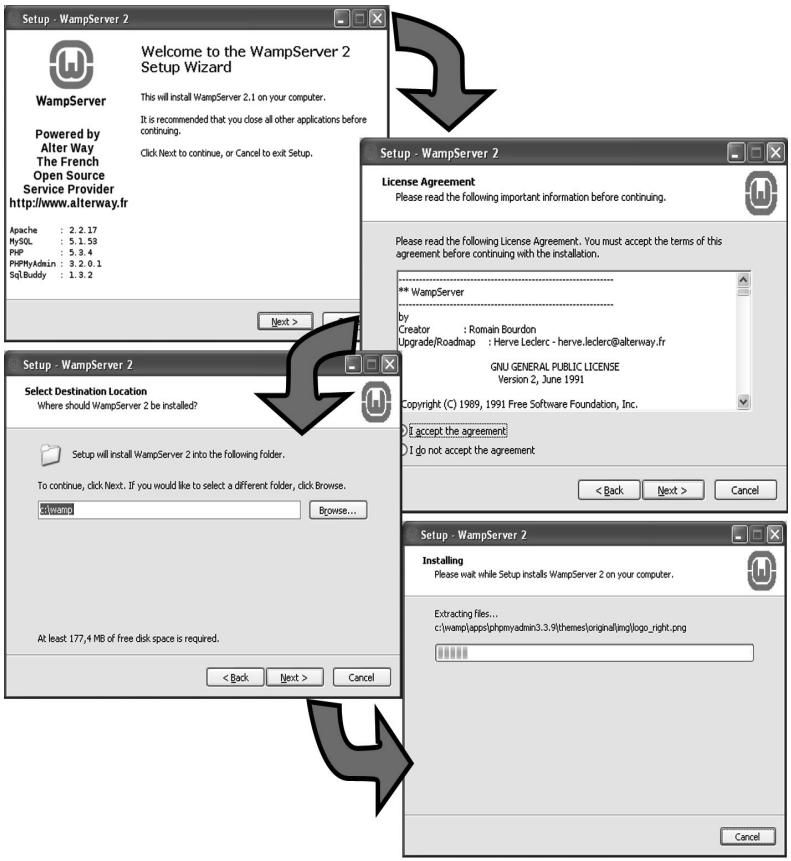
2
Instalação do WampServer

2. Instalação do WampServer

Já que vamos estudar **PHP**, precisaremos de um servidor capaz de rodar nossos programas, para que possamos testar as técnicas contidas neste livro e fazer experimentos. Existem diversos programas gratuitos, disponíveis na Internet, que instalam e rodam servidores Web, equipados com **PHP** e algum tipo de sistema gerenciador de banco de dados (normalmente, **MySQL**). Na primeira versão deste livro, usavamos o **EasyPHP** (que é ótimo), mas desta vez vamos instalar o **WampServer**, que é mais amplamente utilizado e parece ter um suporte mais constante. O nome **WampServer** é uma sigla para **Windows**, **Apache**, **MySQL** e **PHPServer**.

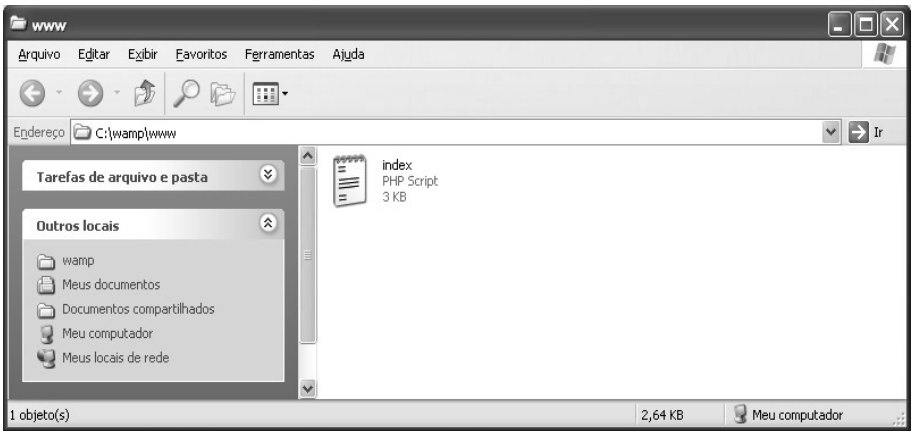
A primeira coisa a se fazer é o download do programa, no site **http://www.wampserver.com.** Obviamente pegaremos a última versão. O site é originalmente escrito em francês, mas já existem recursos de tradução de sites. Mesmo vendo o site no idioma original é fácil achar a área de downloads, logo abaixo do texto de apresentação. É muito importante pegar a versão correta do **Wamp**, ou seja, dizer aquela que foi feita para o seu sistema operacional.

Após baixar o programa, sua instalação segue um esquema bastante conhecido: é só aceitar e **OK** em todas as telas. Em uma delas, é preciso concordar com os termos de uso (é um freeware, não custa nada) clicar em, e seguir em frente.



O **WampServer** instala um servidor **Apache** com **MySQL** e **PHP** no **Windows**. Após ser ativado o **WampServer** exibe um ícone ( ) na barra de ferramentas perto do relógio/calendário. Clicando nele, temos algumas opções como o **PHPMyAdmin**, que nos permite construir e alterar de forma simples o banco de dados. Mas a opção mais interessante de todas, por hora, é **www directory**, que simplesmente abre a pasta na qual gravaremos o conteúdo do site que vamos disponibilizar/testar.

Vamos apagar tudo o que tem nesta pasta e depois criar, dentro dela, um arquivo chamado **index.php**. O arquivo **index** é sempre o primeiro a ser exibido no navegador quando se abre o site.



Dentro do arquivo **index.php**, escreva um pequeno código em **PHP** só para fazer uma demonstração do que significa uma aplicação **server side**. Segue o código:

```
<?php
  $a = 2;
  echo '<div style="color:#f00;">Exibir numero '.$a.'</div>';
?>
```