INTRODUÇÃO AO DESENVOLVIMENTO WEB COM PHP e MySQL

Dênis Leonardo Zaniro
Edilson José Davoglio Candido
Fernando Vieira Duarte
José Arnaldo Mascagni de Holanda

Ficha Catalográfica elaborada por Marcel Pereira Santos (Bibliotecário-Documentalista: CRB8-8060) / Biblioteca do IFSP – *Campus* Araraquara

Z31i

Zaniro, Dênis Leonardo; Candido, Edilson José Davoglio; Duarte, Fernando Vieira; Holanda, José Arnaldo Mascagni de

Introdução ao desenvolvimento web com PHP e MySQL / Dênis Leonardo Zaniro, Edilson José Davoglio Candido, Fernando Vieira Duarte, José Arnaldo Mascagni de Holanda. – 1. ed. – São Paulo: EDIFSP, 2025.

ISBN: 978-65-5823-048-9

1. PHP. 2. MySQL 3. Linguagem de programação de computador. 4. Desenvolvimento de sistemas. I. Título.

CDD: 005.276

Prefácio

Este livro surgiu como resultado da experiência profissional e das vivências em sala de aula dos professores da área de informática do Câmpus Araraquara do Instituto Federal de São Paulo (IFSP). Para atender às demandas que normalmente surgiam no processo de ensino e aprendizagem, formou-se uma equipe de professores com experiência em desenvolvimento Web para trabalhar na concepção e elaboração de um material que fornecesse apoio às diferentes práticas pedagógicas adotadas em sala de aula. A primeira versão deste material foi elaborada e disponibilizada no ano de 2018, e passou a ser utilizada por professores e estudantes nas aulas e nos estudos. A partir da experiência obtida, o material foi inteiramente revisado, atualizado e ampliado. A segunda versão, que é a base deste livro, foi disponibilizada aos professores e estudantes do Câmpus Araraquara no ano de 2021.

A proposta é capacitar estudantes e demais interessados a compreender, analisar e implementar sites e aplicações na Web utilizando a linguagem PHP. Fundamentos sobre acesso a bancos de dados também são abordados pelo livro, a partir da utilização do SGBD MySQL. Muitos conceitos, técnicas e boas práticas apresentados neste material são aplicáveis ao desenvolvimento Web de forma geral, assim, com pouca ou nenhuma adaptação, poderiam ser utilizados em outras linguagens e tecnologias de desenvolvimento.

O conteúdo e a estrutura do livro foram cuidadosamente pensados para beneficiar públicos diversificados, contemplando desde estudantes que desejem iniciar seus estudos na área até professores e profissionais que precisem se aprofundar em algum tema específico. Sabemos que há muitos outros livros sobre PHP e desenvolvimento para Web disponíveis na literatura, e este não necessariamente substitui ou complementa os demais.

A partir de necessidades observadas em diferentes momentos e ações de ensino e aprendizagem, buscou-se reunir, no mesmo material, tópicos fundamentais em praticamente qualquer disciplina de desenvolvimento para Web. Os capítulos foram organizados de tal modo que o leitor possa ampliar gradativamente seu conhecimento e desenvolver habilidades de programação conforme avança pelo conteúdo. Ao criar um caminho didático interligando vários assuntos que consideramos importantes, este livro pode servir como uma referência completa em certos contextos ou servir de ponto de partida direcionando a consulta a outros materiais.

Sumário

Capítulo 1. Introdução ao desenvolvimento de sistemas para web

Com a popularização da Internet e da World Wide Web (WWW), novos hábitos de consumo, comportamento, entretenimento e comunicação foram introduzidos no cotidiano dos seres humanos. Com o uso de linguagens, métodos e ferramentas específicas, o desenvolvimento de sistemas para Web deve acompanhar essas mudanças na rotina das pessoas e a complexidade dessa plataforma. Um sistema para Web envolve alguns elementos: conteúdo, interação, navegação e armazenamento de dados. Tais elementos são o tema central deste livro.

1.1. Programação desktop vs. programação web

As aplicações desktop são escritas usando uma linguagem de programação (C, Java, C#, entre outras) e geram um programa (software). Este software é instalado e executado em um computador realizando tarefas específicas, o que torna a aplicação dependente, ou seja, tem que ser compatível com a máquina (hardware e sistema operacional). Também podem ser usados por vários usuários em um ambiente de rede e podem funcionar com baixo ou quase nulo uso de Internet.

As linguagens de programação web, por sua vez, permitem desenvolver websites ou sites acessados via internet por meio de programas chamados navegadores ou browsers. Assim, as aplicações web possuem menos problemas de compatibilidade de hardware e software. Os sites podem ser classificados quanto à natureza de seu conteúdo como: sites de conteúdo estático, que não sofrem grandes alterações ao longo do tempo, e sites de conteúdo dinâmico, que utilizam aplicações web para responder às requisições dos usuários.

1.2. Sites de conteúdo estático

Os sites estáticos não sofrem grandes alterações em seu conteúdo ao longo do tempo, pois contêm páginas escritas diretamente usando a linguagem de marcação HTML (HyperText Markup Language), sem o apoio de aplicações que automatizam a geração do seu conteúdo. Os sites estáticos são similares às páginas de uma revista ou de um jornal impresso, cujo conteúdo não se modifica com a interação do leitor.

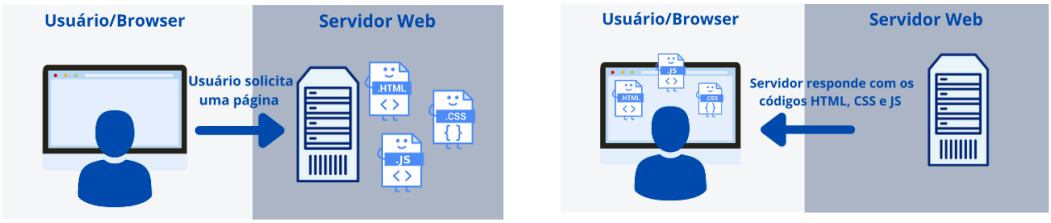


Figura 1.1: Funcionamento de um site de conteúdo estático.

O funcionamento geral de um site de conteúdo estático é ilustrado na Figura 1.1. Primeiramente, os desenvolvedores do site escrevem o conteúdo como páginas HTML, podendo utilizar também folhas de estilo Cascading Style Sheets (CSS) e scripts na linguagem JavaScript (JS). Este conteúdo é então disponibilizado por meio de um servidor web. Para acessá-lo, os usuários devem utilizar um navegador ou browser para solicitar uma página web ao servidor digitando um endereço URL (por exemplo, http://sitedeinteresse.com). Por fim, o servidor web responde à solicitação enviando os códigos HTML, CSS e JS originalmente escritos pelos desenvolvedores, que são processados e exibidos pelo navegador aos usuários.

1.3. Sites de conteúdo dinâmico e aplicações web

Os sites dinâmicos, por sua vez, permitem que o conteúdo seja gerado por aplicações web que realizam o processamento de informações enviadas pelo usuário. Assim, os sites tornam-se interativos e seu conteúdo pode ser modificado com facilidade. Exemplos de sites de conteúdo dinâmico são portais de notícias, sites de comércio eletrônico, redes sociais, entre outros.

As aplicações web, que geram o conteúdo dos sites dinâmicos, são escritas usando uma combinação de linguagem de marcação (por exemplo, HTML) com linguagem de programação web (por exemplo, PHP). As aplicações web são executadas em um servidor web, cuja função é receber uma solicitação (requisição), geralmente via formulário web, e devolver uma resposta para o cliente, como uma página HTML, por exemplo. A Figura 2 ilustra esse processo.

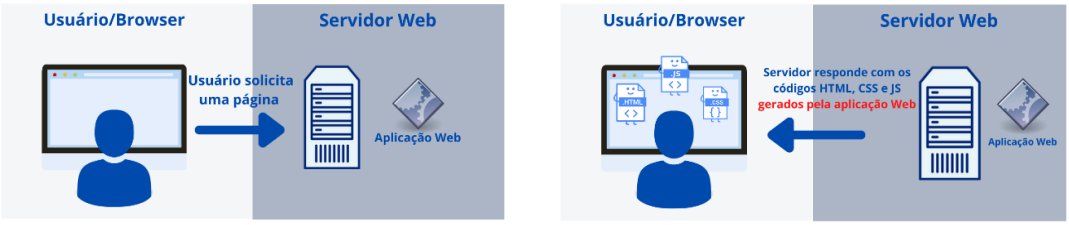


Figura 1.2: Funcionamento de um site de conteúdo dinâmico com aplicação web.

Na Figura 1.2 é ilustrado o funcionamento geral de um site de conteúdo dinâmico. Primeiramente, os desenvolvedores do site programam uma aplicação web combinando linguagem de marcação (por ex.: HTML) com linguagem de programação web (por ex.: PHP). Esta aplicação é então carregada em um servidor web. Os usuários devem utilizar um navegador ou browser para solicitar ao servidor o processamento de uma página web, por exemplo um formulário preenchido e submetido via clique de botão. O servidor recebe essa solicitação e a direciona para a aplicação web realizar o processamento dos valores preenchidos e gerar uma resposta com os códigos HTML, CSS e JS. Por fim, o servidor web envia a resposta gerada pela aplicação web para o navegador, que a exibe para os usuários.

Existem várias linguagens de programação para desenvolver aplicações web, sendo que a linguagem PHP é uma das mais populares, com a vantagem de ser gratuita e de código aberto.

1.4. Arquitetura Cliente/Servidor

Na arquitetura cliente/servidor, o computador cliente envia uma solicitação para o servidor através da conexão de rede. A solicitação é processada pelo servidor que retorna a resposta para o cliente. A internet é baseada na arquitetura cliente/servidor, onde o servidor web processa solicitações de clientes simultaneamente.

A arquitetura cliente/servidor funciona, geralmente, conforme a Figura 1.3.

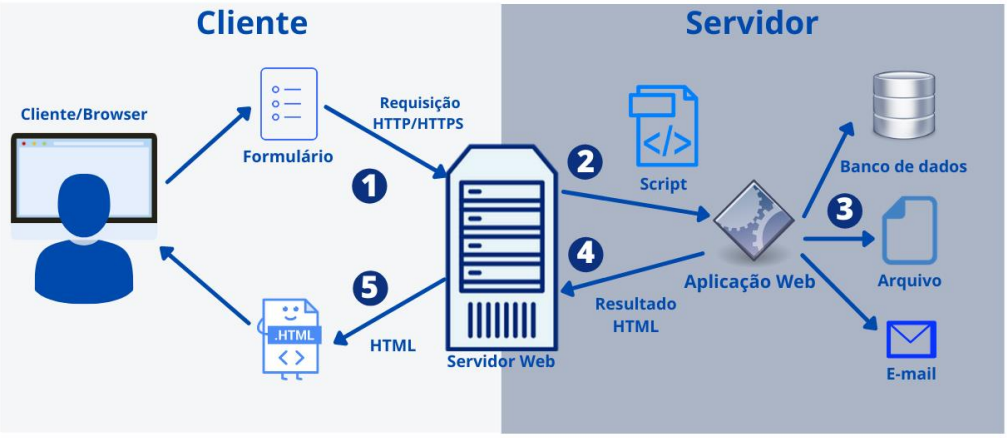


Figura 1.3: Processamento de um script PHP na arquitetura cliente/servidor.

Primeiramente, o usuário cliente preenche e envia o formulário ao servidor web por meio de uma requisição HTTP ❶. No servidor, uma aplicação web deve responder à sua requisição. Quando recebe seus dados, a aplicação web processa o código (um script) ❷. Dependendo das instruções, o um banco de dados pode ser acessado (consultado, atualizado etc.), um e-mail é enviado etc. ❸. Uma página HTML é montada pela aplicação web contendo os resultados desta requisição ❹. Finalmente, a página web gerada no formato HTML é enviada e carregada pelo navegador do cliente ❺.

Capítulo 2. Introdução ao PHP

O PHP (acrônimo para **P**HP: **H**ypertext **P**reProcessor) é uma linguagem de script de código aberto interpretada em um servidor Web (Nginx, Cloudflare Server e Apache, entre outros) que contenha o módulo PHP instalado. Um arquivo PHP possui a extensão ".php" e pode incluir texto, *tags* HTML, código CSS, JavaScript ou PHP. Após a interpretação dos scripts pelo módulo PHP no servidor Web, o resultado é geralmente retornado para o navegador em formato HTML. Contudo, o retorno também pode ser uma imagem, arquivos PDF ou textos em formato XML/JSON.

Saiba mais:

Um script (também conhecido como *scripting*) é um conjunto de instruções capazes de serem executadas sem a necessidade do processo de compilação. Para isso, as linguagens de script utilizam um programa, chamado de interpretador, que é responsável pela tradução e execução dos comandos a partir do código fonte. PHP, Python, Perl e JavaScript são exemplos de linguagens de script.

A linguagem de script PHP é utilizada para o desenvolvimento de aplicações Web e pode gerar conteúdos de páginas dinamicamente, manipular arquivos no servidor, coletar os dados de formulários, gerenciar *cookies* e sessões, manipular bancos de dados, realizar o controle de acesso e criptografar dados. O PHP, criado em 1994, evoluiu bastante desde então, e possui as seguintes características:

- Compatibilidade com bancos de dados e servidores Web: PHP é compatível com os principais bancos de dados (Db2, Firebird/Interbase, MySQL, Oracle, PostgreSQL, SQLite e SQL Server) e com a maioria dos servidores Web, incluindo Nginx, Cloudflare Server e Apache, que juntos, são utilizados por mais de 85% dos *websites* existentes.
- Código aberto: As versões PHP são de código aberto, livres e distribuídas de acordo com a "licença PHP", criada pelo "PHP Group". O código fonte pode ser adaptado e redistribuído, o que também permitiu a disponibilização de *frameworks* PHP, como Laravel e Symfony.

- Roda em várias plataformas: aplicações PHP podem ser desenvolvidas em diversos sistemas operacionais (Linux, MacOS, Windows), instaladas em plataformas de computação em nuvem e acessadas por diferentes navegadores.
- Suporte da comunidade: PHP possui uma comunidade que oferece suporte online e elabora documentos, guias e tutoriais para auxiliar os novos desenvolvedores e mostrar como novas características e funcionalidades podem ser utilizadas.

O código PHP é delimitado pelas instruções de processamento (*tags*) de início e fim <*?php* e *?*>, que permitem a escrita de códigos PHP, inclusive embutidos em páginas HTML. A instalação pode ser realizada a partir do *site* http://www.php.net. Nele, também são disponibilizadas a documentação, comentários, exemplos e correções para os defeitos (*bugs*) encontrados em versões anteriores.

Segundo o *site* W3Techs (https://w3techs.com/technologies/details/pl-php), em 2021, o PHP é utilizado por 78,1% dos *websites* que possuem uma linguagem de programação do lado servidor (*server side*). Dentre eles, podemos citar Facebook, Slack, Tesla, Tumblr, Wikipedia e WordPress.

2.1. Histórico

O PHP foi criado por Rasmus Lerdorf em 1994, mas seu anúncio ocorreu somente em 1995. Inicialmente descrito como Personal Home Page Tools (PHP Tools), tratava-se de um conjunto de programas CGI (Common Gateway Interface, ou Interface Comum de Ligação) escritos em linguagem C e utilizados para o monitoramento de visitas ao currículo online de Rasmus.

A segunda versão foi lançada por ele em 1996 e chamada de PHP/FI (Forms Interpreter). Ela facilitava a criação de formulários, a manipulação dos dados desses formulários em páginas subsequentes, assim como o controle de acesso nessas páginas. Também oferecia suporte aos bancos de dados DBM, MySQL e Postgres95, cookies e funções de apoio definidas pelo usuário.

Em 1997, os estudantes israelenses Andi Gutmans e Zeev Suraski, do *Technion* (Instituto de Tecnologia de Israel), se voluntariaram para reescrever o interpretador. Eles também criaram os primeiros recursos de orientação a objetos e um conjunto de bibliotecas para facilitar o desenvolvimento de novas extensões. O processo de desenvolvimento do

PHP deixou de ser um projeto pessoal de Rasmus para se tornar um projeto de código aberto, contando com o auxílio de Andi Gutmans, Zeev Suraski e milhares de outros programadores mundo afora.

Como resultado do trabalho dessa esquipe, em 1998, foi lançado o PHP 3 e a linguagem foi renomeada para PHP: *Hypertext PreProcessor*. Essa versão se assemelha com o PHP utilizado atualmente e oferecia suporte a maioria dos sistemas operacionais (Windows 95/NT, versões Unix e Macintosh), servidores Web (Apache, Netscape, WebSite Pro e Microsoft Internet Information Server) e bancos de dados (Oracle, Sybase, Solid, MySQL, PostgreSQL e ODBC).

Em 2000, Andi e Zeev lançaram o PHP 4.0. Essa versão foi baseada no núcleo "*Zend Engine*" (o termo *Zend* baseou-se nos primeiros nomes de Andi e Zeev), desenvolvido por eles em 1999, e apresentou melhorias de performance, suporte para mais servidores Web, sessões, formas mais seguras de manipular a entrada de dados, *buffer* de saída e novos recursos de orientação a objetos.

O PHP 5.0 foi lançado em 2004 e utilizava o núcleo *Zend Engine* 2, com um novo modelo de objetos. Entre 2004 e 2013, foram lançadas as versões de 5.0 até 5.6, disponibilizando um novo conjunto de extensões (SimpleXML, SOAP, MySQLi e SQLite), a biblioteca PHP *Data Objects* (PDO) como uma nova interface de acesso aos bancos de dados, suporte a JSON e fusos horários, blocos *finally* para tratamento de exceções, rastreamento de progresso do upload de arquivos, melhorias significativas de desempenho e correção de bugs. Durante esse período, também houve o lançamento do *Facebook.com* (2004) como um *site* PHP. A empresa também desenvolveu a linguagem *Hack* (2014), que estendia o PHP e contava com recursos que foram integrados a ele posteriormente. Além disso, houve também a tentativa de lançamento, sem sucesso, do PHP 6.0, cujo projeto foi abandonado oficialmente em 2010.

Em 2015 é disponibilizado o PHP 7.0, com o núcleo *Zend Engine* 3 e mais uma renovação da linguagem, adicionando melhorias e novos recursos, possibilitando uma versão duas vezes mais rápida do que a anterior (PHP 5.6). Novas características e funcionalidades foram adicionadas nas versões 7.1 (2016 – modificadores de visibilidade de constante de classe, tipos anuláveis e de retorno vazio), 7.2 (2017 – inclusão da biblioteca *libsodium* para criptografia e ampliação de tipos de parâmetro), 7.3 (2018 –

atualizações para *strings heredoc* e *nowdoc* e aprimoramento do coletor de lixo) e 7.4 (2019 – funções seta, propriedades tipadas e separador literal numérico).

O PHP 8.0 foi lançado em 2020. Dentre as principais novidades estão a possibilidade de iniciar *arrays* com índice negativo, expressão *match*, argumentos nomeados, atributos (termo conhecido como anotações em outras linguagens), tipos de união (pode-se definir mais de um tipo para uma variável), cláusula *throw*, motores de compilação JIT (*Just in Time*, que promete melhorias de performance nas aplicações PHP), *weak maps* (permitem a criação de objetos com referências "fracas", que podem ser removidas pelo coletor de lixo quando o objeto deixar de ser utilizado), operador *nullsafe* e vírgula de rastreamento na lista de parâmetros.

Na tabela a seguir são listados os principais eventos e lançamentos relacionados com o PHP, desde sua criação em 1994 até o ano de 2021.

Ano	Eventos
1994	- Criação do PHP, por Rasmus Lerdorf.
1995	- Disponibilização do PHP Tools. - Lançamento do MySQL.
1996	- Lançamento do PHP/FI (*Forms Interpreter*).
1997	- Reescrita do interpretador PHP, por Andi Gutmans e Zeev Suraski.
1998	- Criação do logotipo elePHPant, por Vincent Pontier. - Lançamento do phpMyAdmin, por Tobias Ratschiller.
1999	- Fundação da empresa Zend.
2000	- Disponibilização do PHP 4.0.
2001	- Criação do *framework* para testes PHPUnit, por Sebastian Bergmann.
2003	- Lançamento do WordPress.
2004	- Disponibilização do PHP 5.0. - Lançamento do Facebook como um *site* PHP.
2005	- Lançamento do *framework* CakePHP.
2006	- Início do projeto Symfony. - Lançamento do *framework* CodeIgniter.
2007	- Disponibilização do PHP 5.2, cuja versão permaneceu como a mais popular até 2013.
2011	- Lançamento do *framework* Laravel. - Lançamento do gerenciador de conteúdos Composer.
2014	- Criação da linguagem de programação Hack, pelo Facebook.
2015	- Disponibilização do PHP 7.0.
2020	- Disponibilização do PHP 8.0.

2.2. Instalação e configuração do PHP

Para o desenvolvimento de aplicações Web com PHP são necessárias ferramentas tais como editor de textos, servidor Web com suporte a PHP e servidor de banco de dados. Além disso, é preciso realizar a configuração e gerenciamento (ativação/desativação)

desses recursos. A seguir, são apresentados os detalhes e passo a passo para instalação do Visual Studio Code, um editor de textos, e do XAMPP, um ambiente de desenvolvimento PHP.

- Visual Studio Code: editor de textos multiplataforma criado pela Microsoft para a edição do código fonte. É Open Source, pode ser customizado, permite a instalação de extensões para inclusão de novas funcionalidades e possui um grande conjunto de atalhos para auxiliar no aumento da produtividade.
- XAMPP: ambiente de desenvolvimento PHP gratuito contendo Apache, MySQL, PHP e Perl.

2.2.1. Instalação do Visual Studio Code (VS Code)

1) Acesse a página (https://code.visualstudio.com/) e baixe a versão para seu sistema operacional. Para este exemplo, utilizaremos o Windows 10.

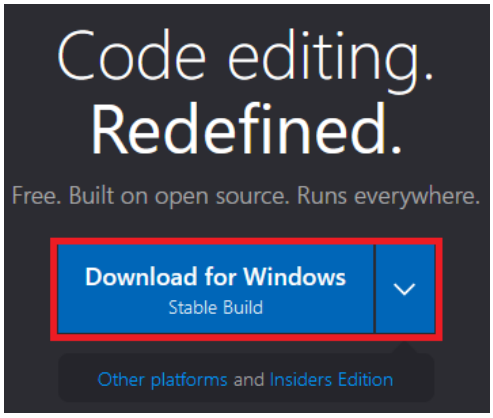


Figura 2.1 – *Download* da versão para Windows 10

2) Clique em “Salvar arquivo” e especifique a pasta na qual o arquivo de instalação será armazenado. Se essa opção não for exibida, seu navegador está configurado para *download* automático. Neste caso, o arquivo será armazenado, por padrão, na pasta “Downloads”.

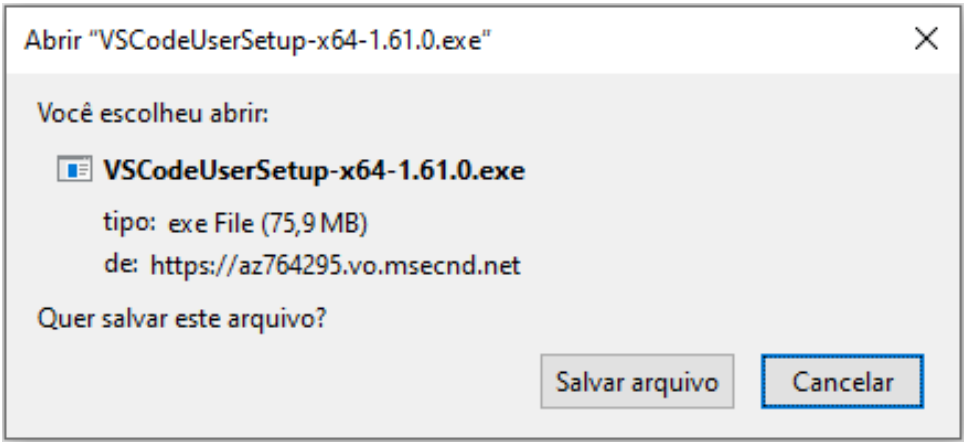


Figura 2.2 – Caixa de confirmação de *download* para Windows 10

3) Verifique o acordo de licença. É preciso aceitá-lo antes de continuar com a instalação.

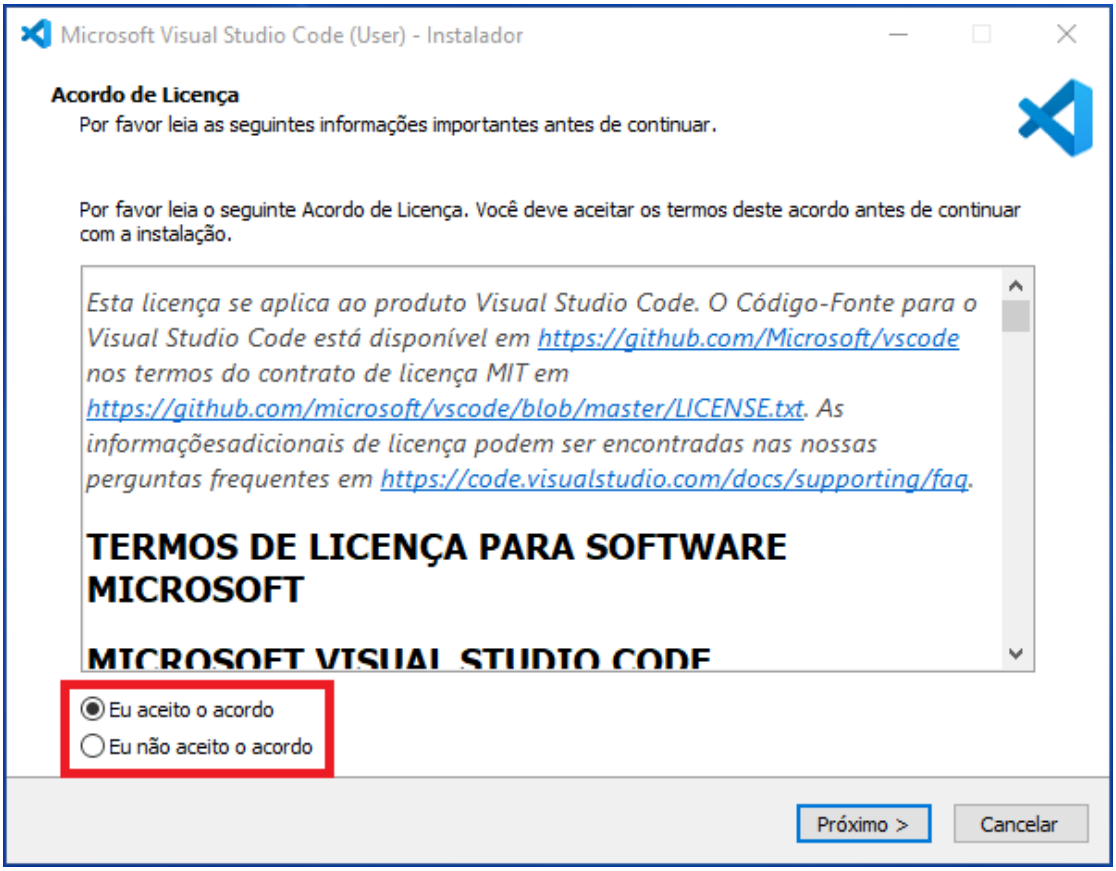


Figura 2.3 – Acordo de licença do VS Code para Windows 10

4) Selecione o local de destino para a instalação. Recomendamos manter a pasta sugerida por padrão.

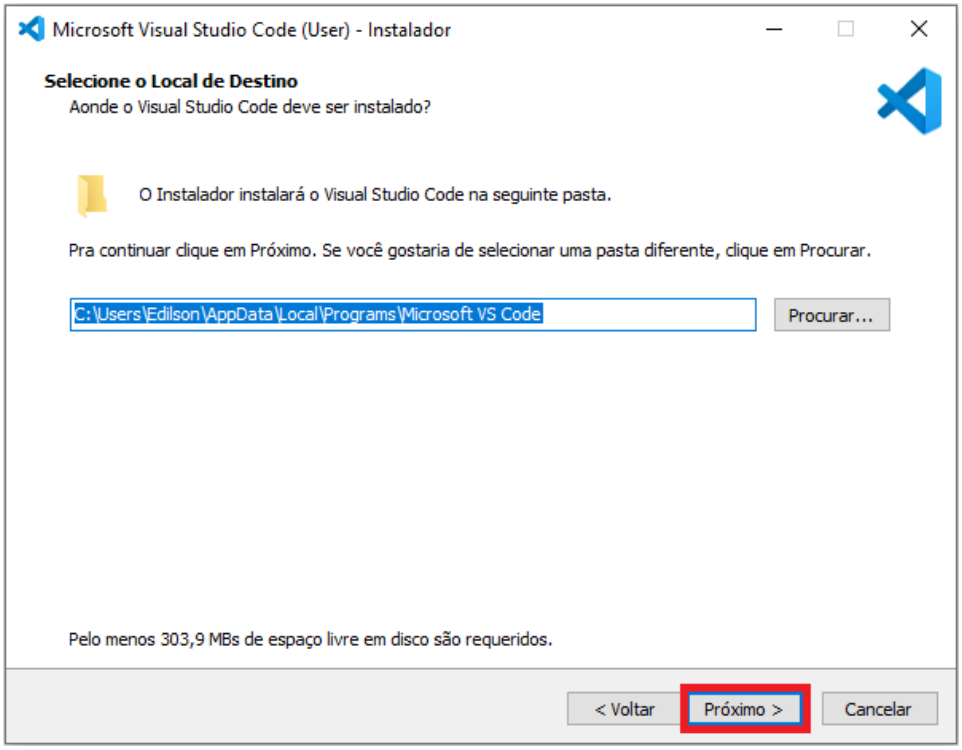


Figura 2.4 – Seleção de local de destino para a instalação

5) Especifique a pasta do menu Iniciar na qual o instalador irá criar os atalhos do programa.

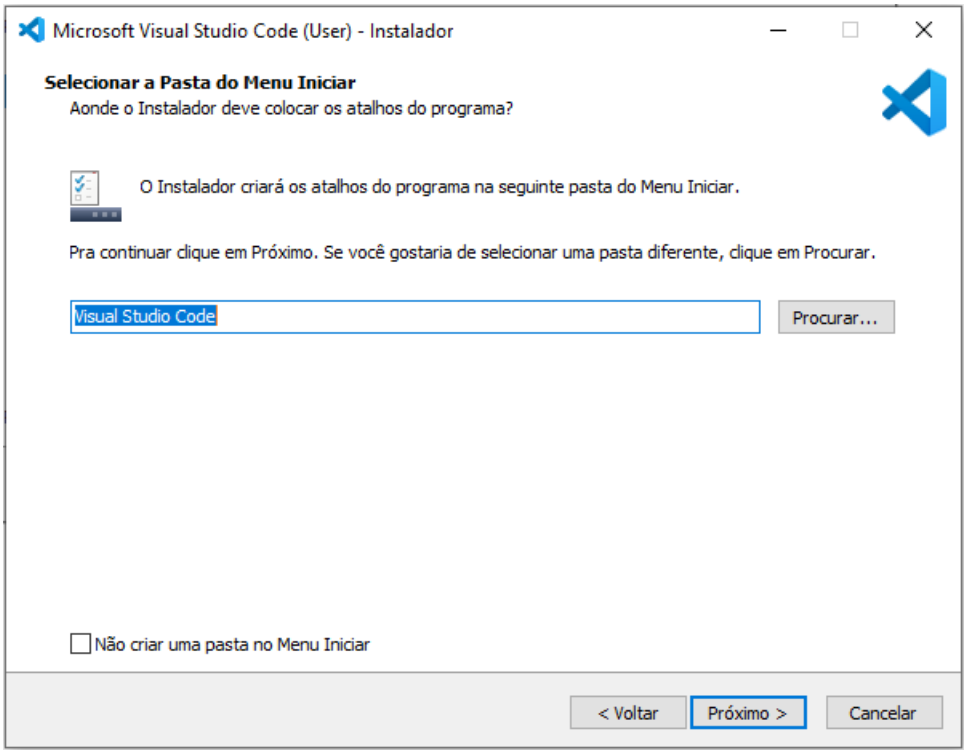


Figura 2.5 – Especificação de local para criação de atalhos do menu iniciar

6) Escolha as tarefas adicionais. Sugere-se incluir a tarefa “Criar um atalho na área de trabalho”.

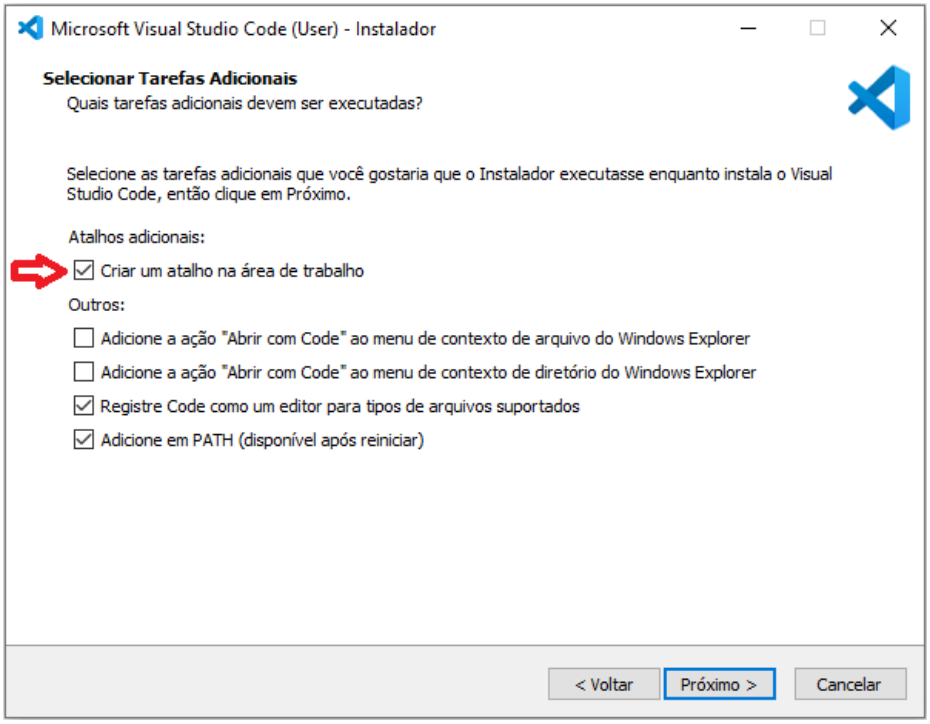


Figura 2.6 – Seleção de tarefas adicionais

7) Verifique as configurações de instalação. Se estiver de acordo, clique em “Instalar” para iniciar o processo.

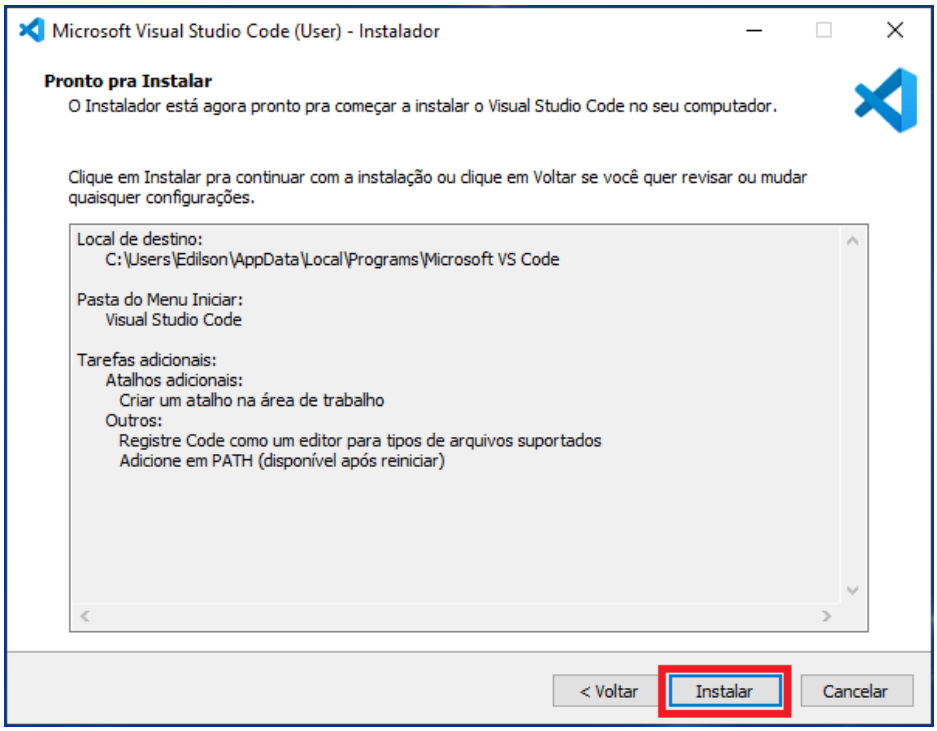


Figura 2.7 – Verificação das configurações de instalação

8) Após o término do processo de instalação, clique em “Concluir”.

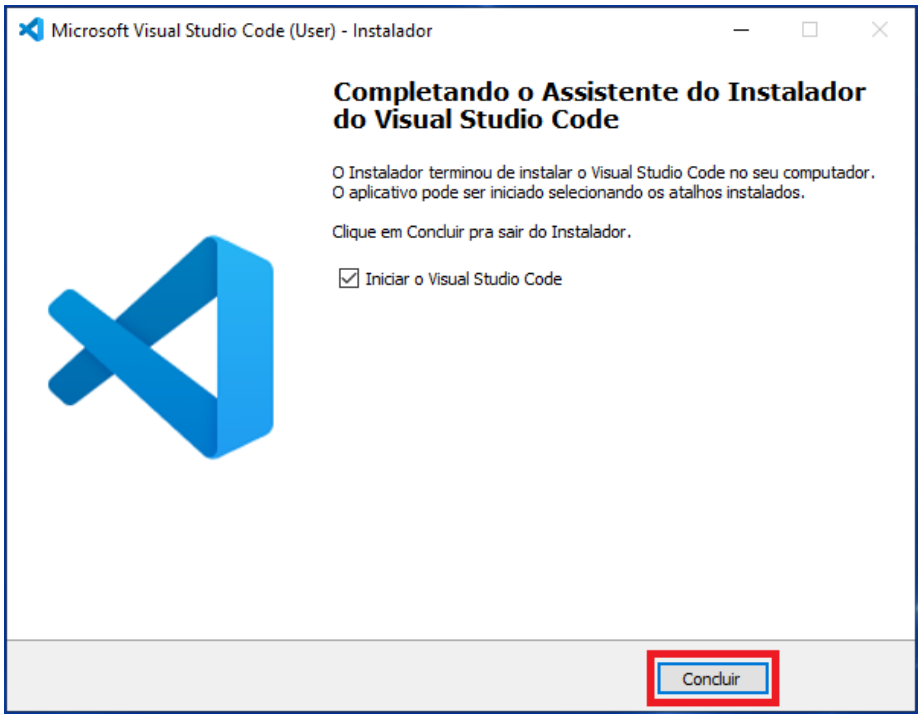


Figura 2.8 – Janela de conclusão do processo de instalação

O VS Code possui uma grande variedade de extensões que auxiliam no processo de desenvolvimento de software. A seguir, são sugeridas duas delas para auxiliar na criação de aplicações Web utilizando PHP: PHP Intelephense (possui características como auto complemento de código, rápido acesso a documentação e dicas) e vscode-icons (ícones customizados para arquivos e pastas).

Para instalar as extensões, inicialize o VS Code e, na barra lateral esquerda, selecione o ícone “Extensões”, conforme a figura 2.9, ou utilize as teclas de atalho “Ctrl + Shift + X”.

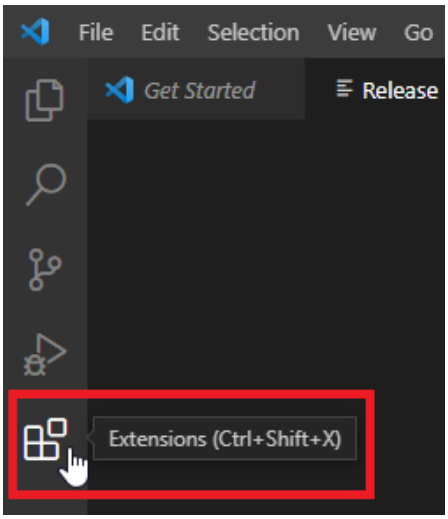


Figura 2.9 – Acesso às extensões do VS Code

Inicie a digitação da extensão e o próprio VS Code irá sugerir opções. Clique na sugestão “PHP Intelephense”.

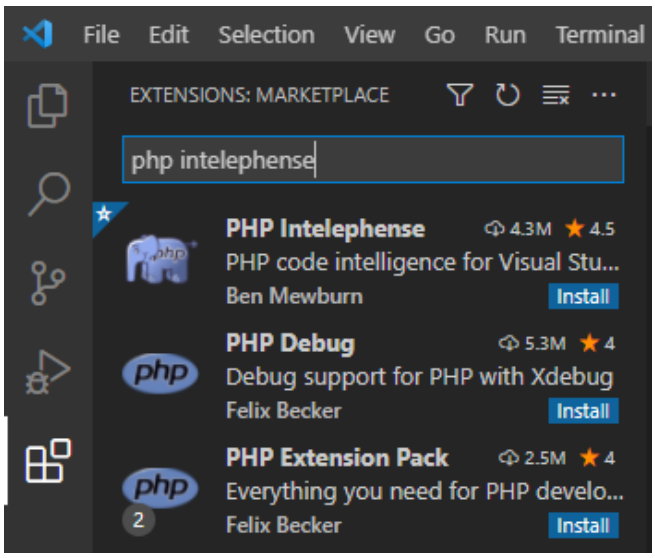


Figura 2.10 – Busca pela extensão PHP Intelephense

Uma página com detalhes da extensão é aberta, exibindo o autor, versão, quantidade de downloads e detalhes. Clique em “Install” para adicionar a extensão a seu Ambiente de Desenvolvimento Integrado (IDE – Integrated Development Environment) VS Code.

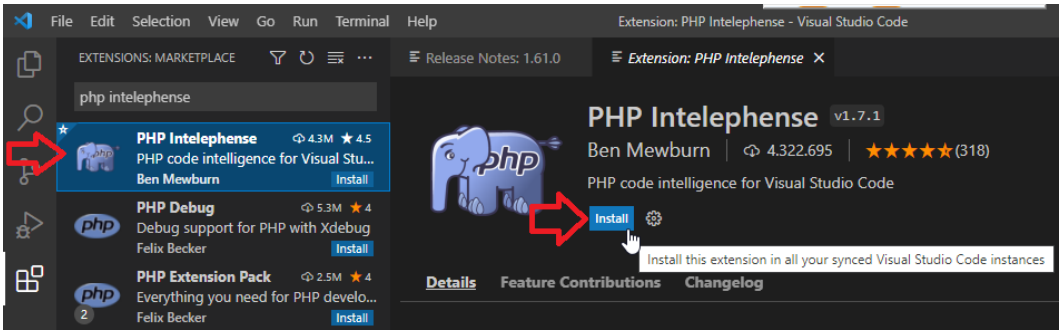


Figura 2.11 – Detalhes sobre a extensão PHP Intelephense

Após a instalação, é possível desabilitar ou até mesmo remover a extensão de seu IDE.

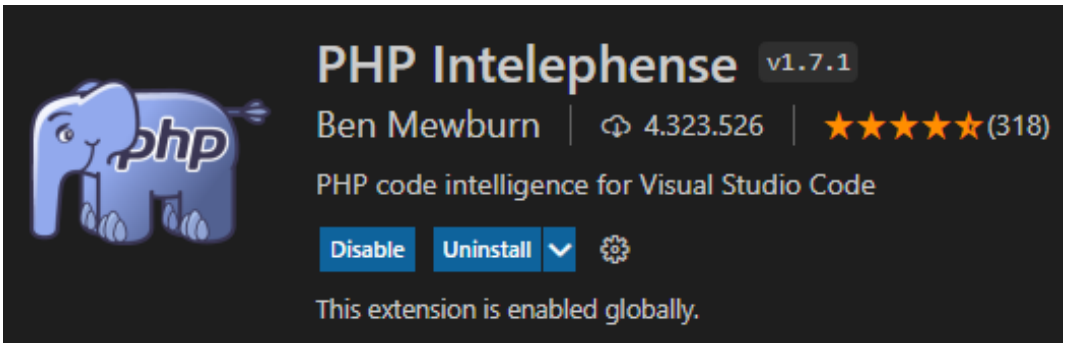


Figura 2.12 – Opções para desabilitar e desinstalar a extensão PHP Intelephense

O processo para instalação da extensão “vscode-icons” é similar ao apresentado anteriormente. Após a digitação de “*icons*” na caixa de pesquisa, ela é exibida como primeira sugestão. Clique na extensão para que a página com os detalhes seja exibida e a seguir, clique em “Install”.

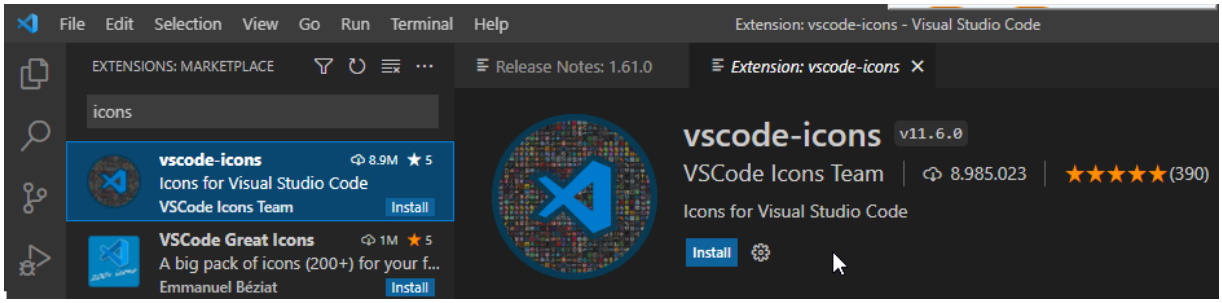


Figura 2.13 – Opções para desabilitar e desinstalar a extensão PHP Intelephense

Após a instalação, é possível desabilitar ou até mesmo remover as extensões de seu IDE. Elas podem ser facilmente acessadas por meio do ícone “Extensões”, na barra de atividades (barra lateral do lado esquerdo).

2.2.2. Instalação do XAMPP

Para a instalação do XAMPP é preciso realizar o download do arquivo de instalação e, assim como foi realizado para o VS Code, especificar a pasta na qual esse arquivo será armazenado ou aguardar o download automático, que por padrão, será baixado na pasta “Downloads”. Veja o passo a passo do processo nas etapas a seguir:

1) Acesse a página (https://www.apachefriends.org/pt_br/index.html) e baixe a versão para seu sistema operacional. Para este exemplo, utilizaremos o Windows 10.

Figura 2.14 – Escolha da versão de instalação do XAMPP

2) Clique com botão direito do mouse sobre o arquivo armazenado em seu computador e escolha a opção “Executar como administrador”. Na caixa de confirmação, responda “Sim”.

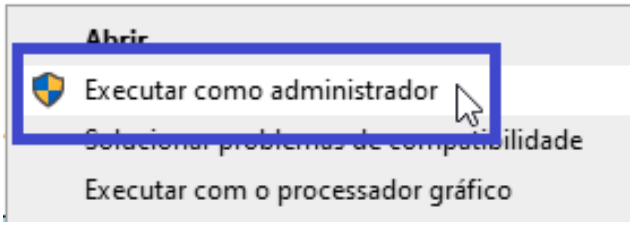


Figura 2.15 – Janela para realizar a instalação como administrador

3) Na tela inicial do assistente de instalação do XAMPP, clique em “Next” (próximo).

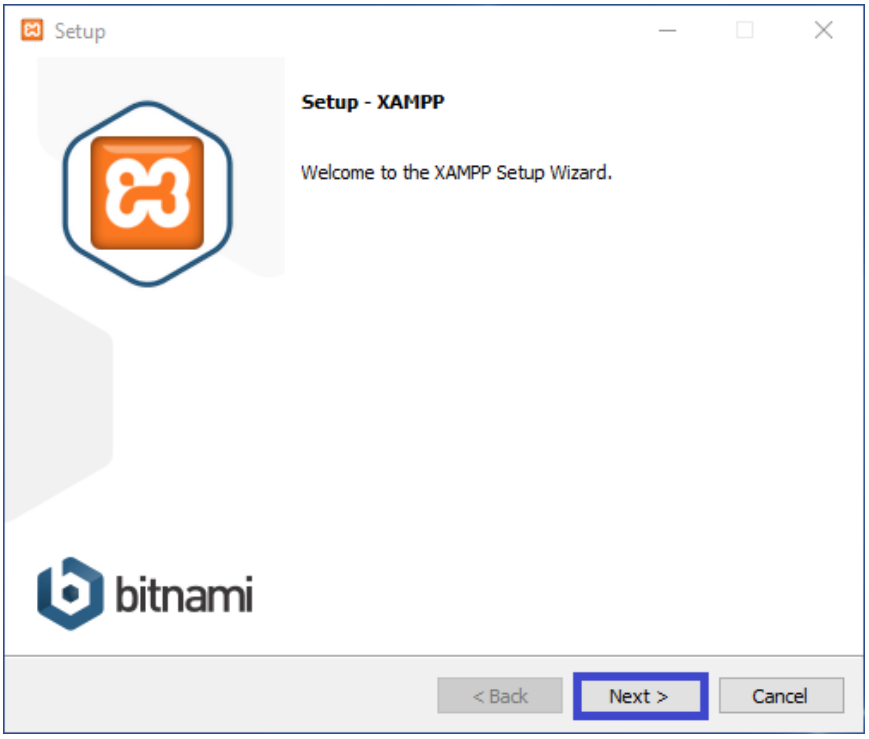


Figura 2.16 – Tela inicial do assistente para instalação

4) Selecione os componentes que deseja instalar. Neste exemplo, a configuração padrão foi mantida.

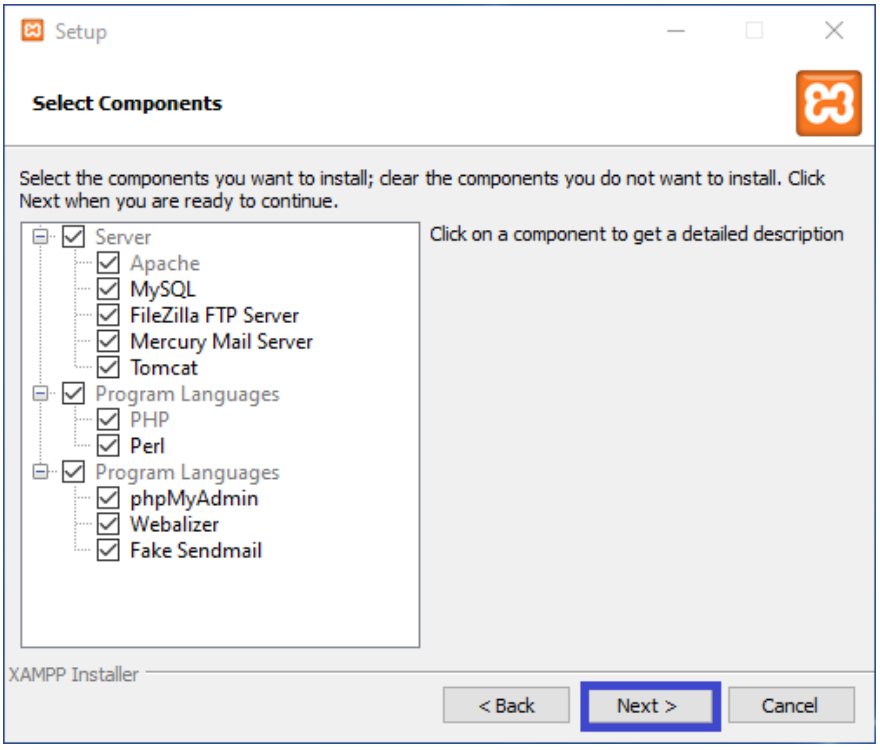


Figura 2.17 – Seleção dos componentes que serão instalados

5) Especifique a pasta para instalação. Sugerimos “C:\xampp”. Contudo, é possível definir outro local de sua preferência.

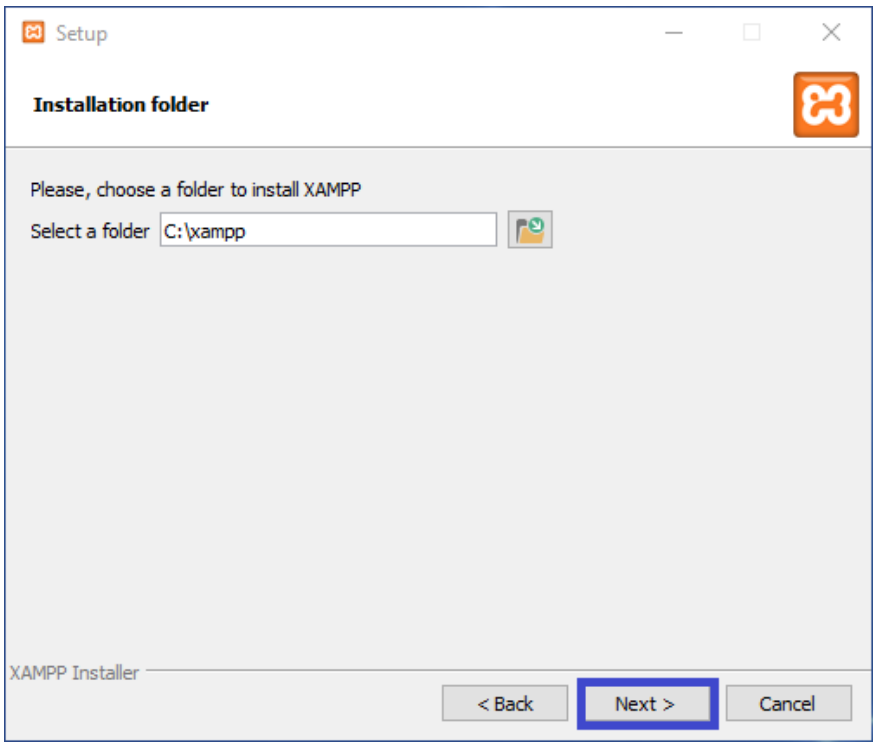


Figura 2.18 – Seleção da pasta para instalação

6) Especifique o idioma.

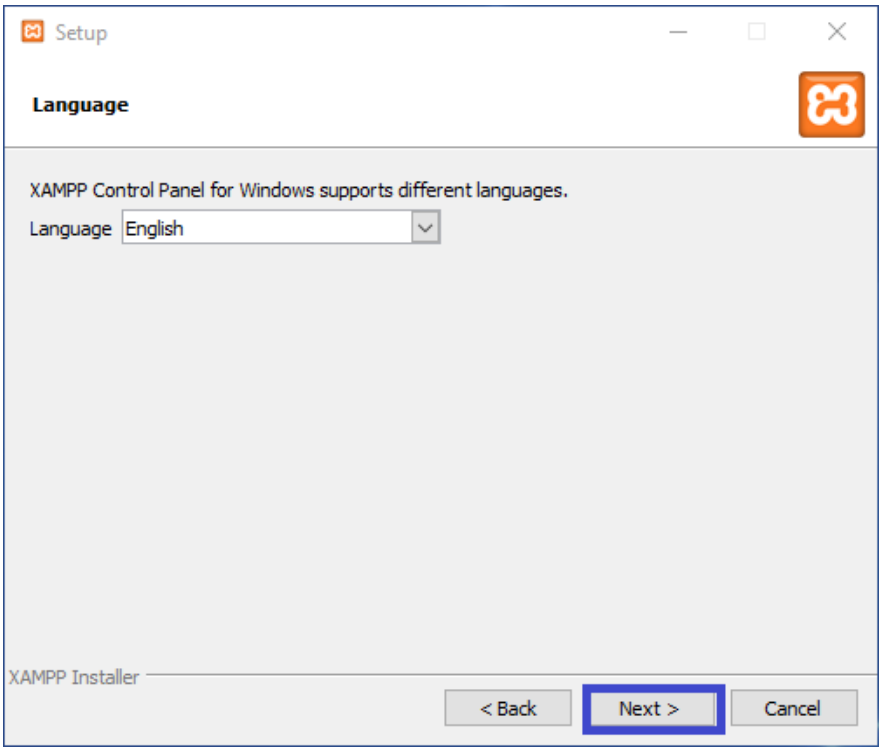


Figura 2.19 – Seleção do idioma

7) Na tela seguinte, é possível requisitar mais informações sobre o Bitnami por meio da caixa de seleção. Não houve a escolha desta opção na instalação.

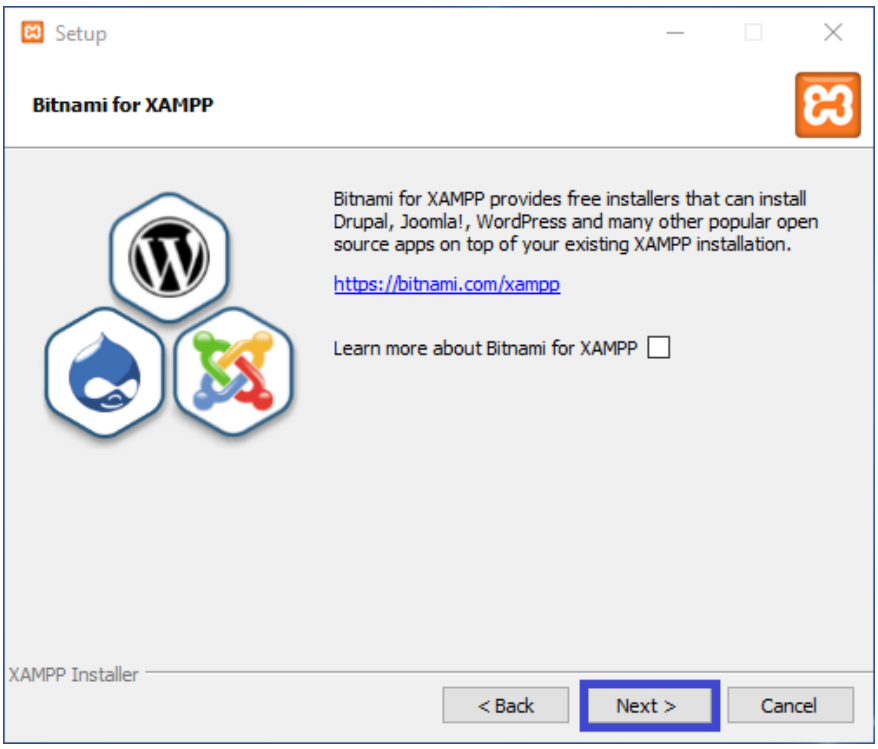


Figura 2.20 – Possibilidade de solicitação de mais informações sobre Bitnami

8) Clique em “Next” (próximo) para iniciar a instalação do XAMPP em seu computador.

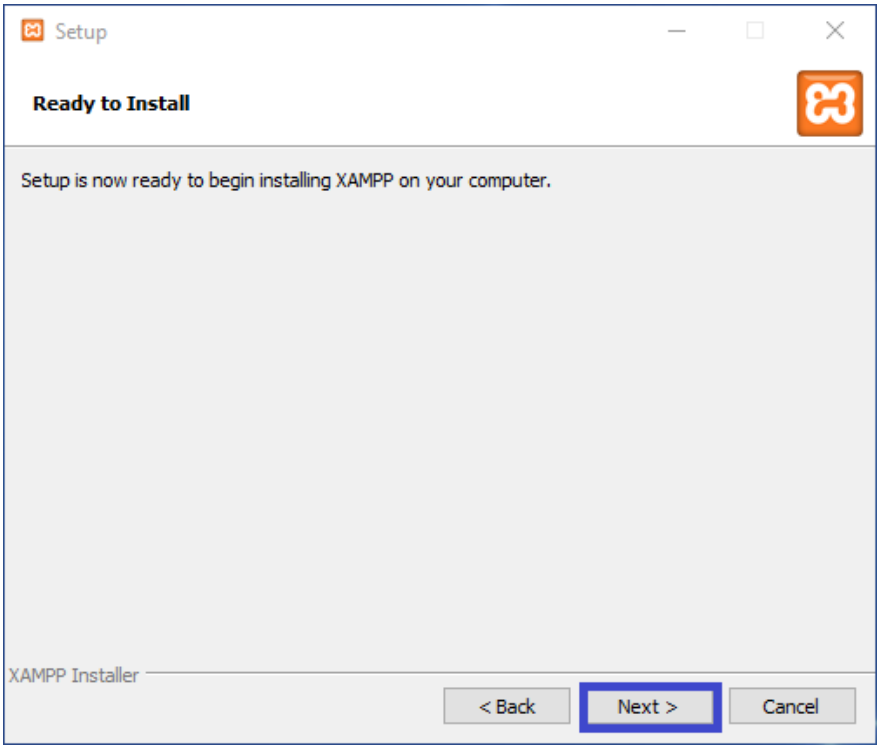


Figura 2.21 – Processo de instalação pronto para ser iniciado

9) Acompanhe o processo de instalação do XAMPP até que ele seja concluído.

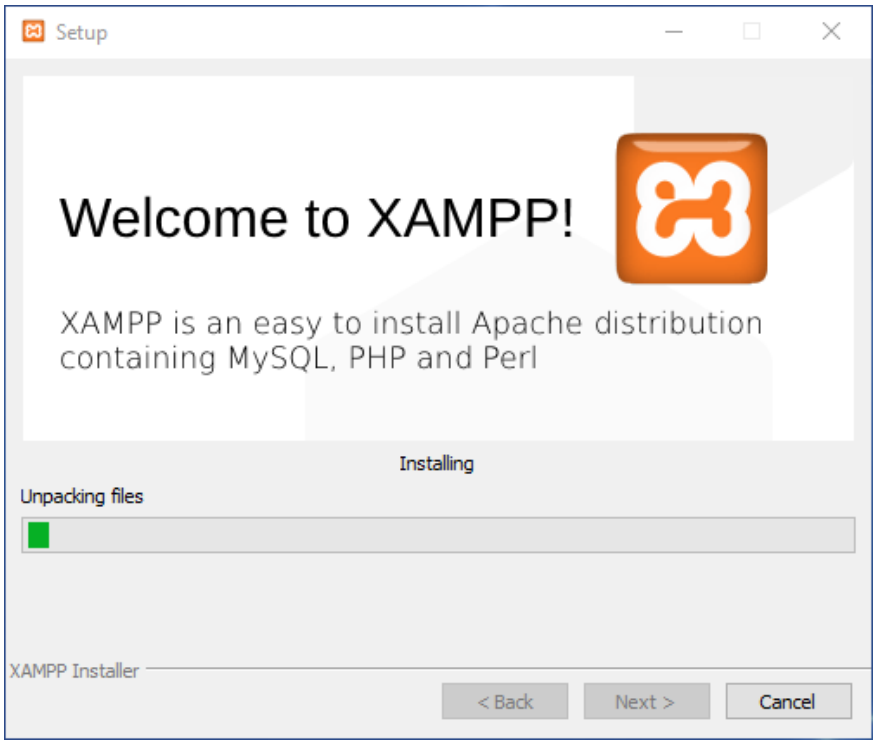


Figura 2.22 – Processo de instalação em andamento

10)O *firewall* instalado (Windows Defender) irá solicitar a liberação de acesso do servidor Apache às redes de comunicação especificadas. Selecione conforme a Figura 2.23.

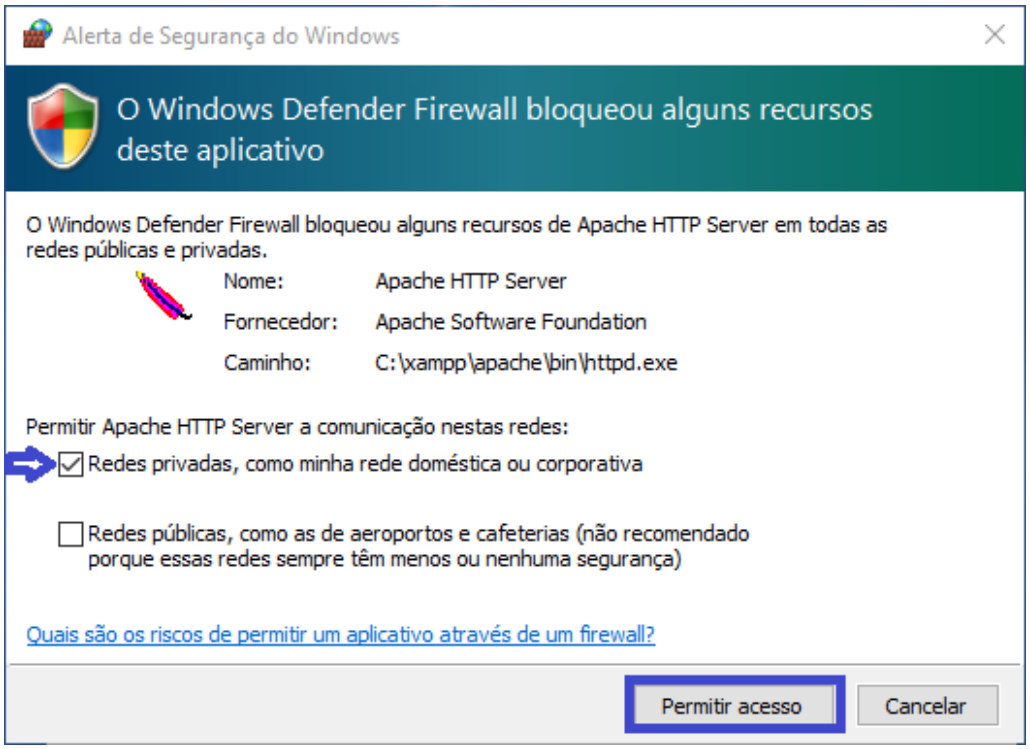


Figura 2.23 – Confirmação de permissão para o servidor Apache comunicar-se nas redes selecionadas

11) Caso sejam solicitadas liberações de acesso de outros recursos, proceda de forma similar ao exemplo apresentado na Figura 2.23 (passo anterior). Após as liberações de acesso junto ao *firewall*, o processo de instalação é concluído. Selecione a caixa de seleção que inicia o painel de controle do XAMPP e clique em “Finish” (Concluir).

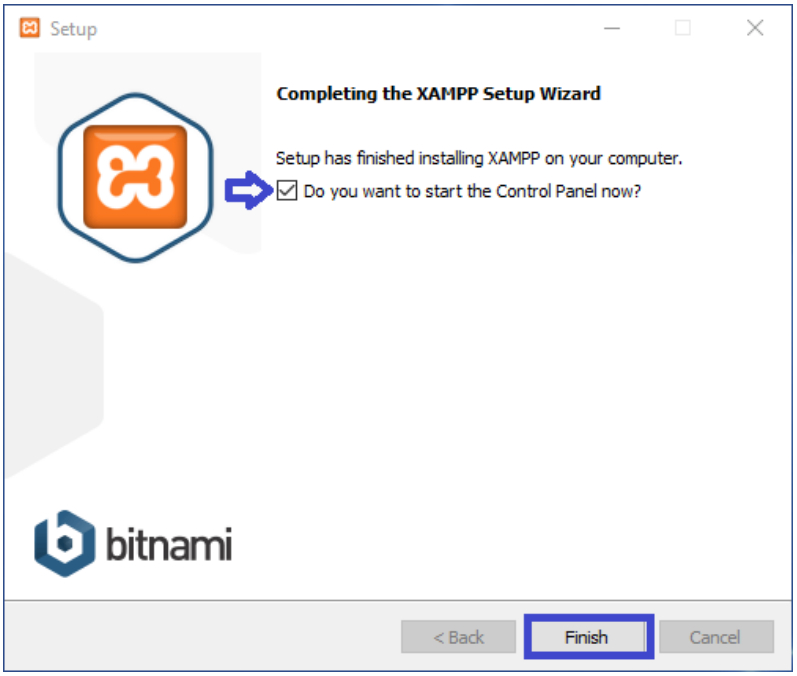


Figura 2.24 – Conclusão do assistente de instalação

Veja a seguir o painel de controle do XAMPP:

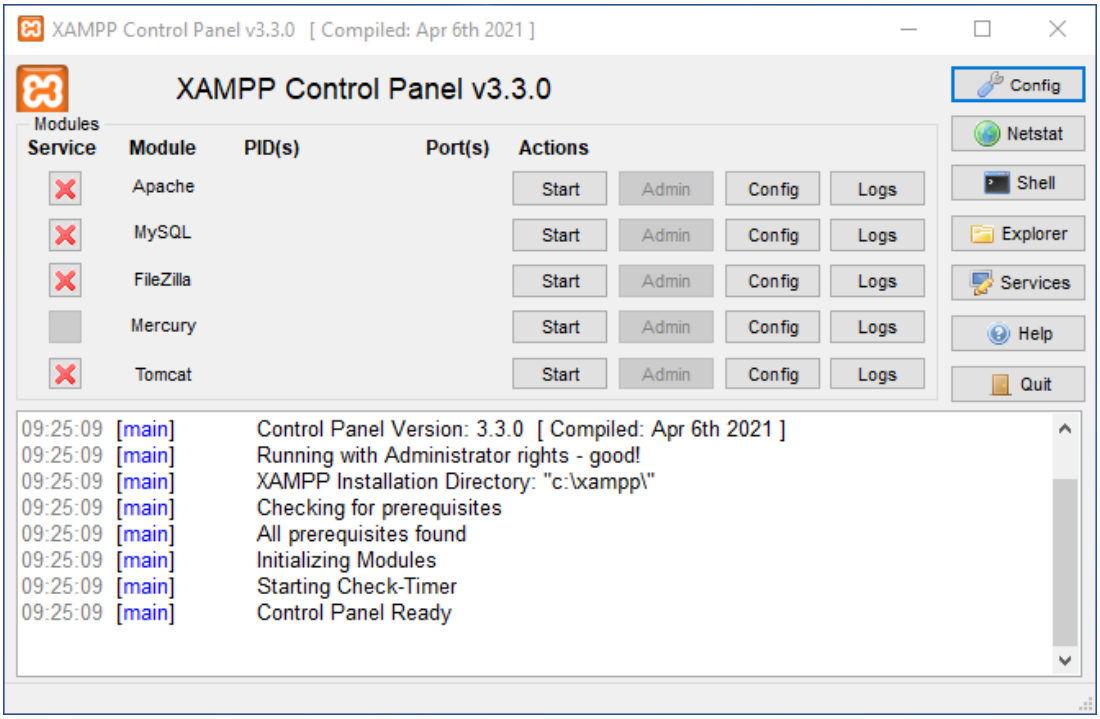


Figura 2.25 – Painel de controle do XAMPP

Uma funcionalidade cujo ajuste de configuração pode ser necessário é a porta utilizada pelo servidor Web (Apache). Para isso, clique em “Config”, conforme exibido na Figura 2.26.

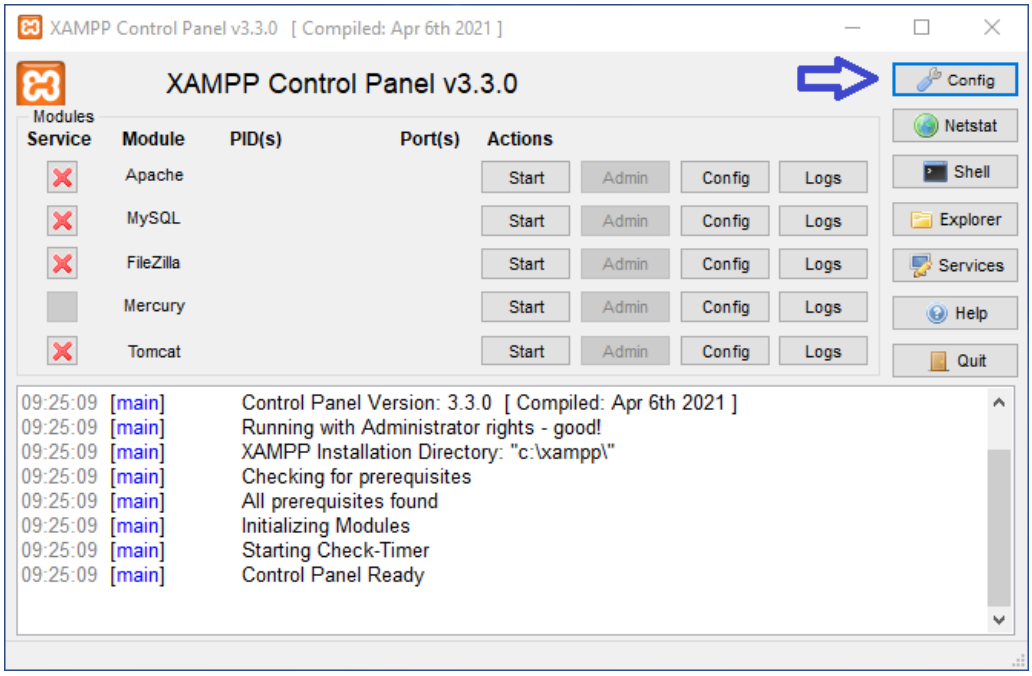


Figura 2.26 – Painel de controle do XAMPP – Configurações

A seguir, clique em “*Service and Port Settings*” (Configurações de portas e serviços).

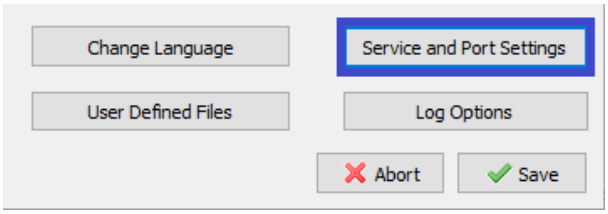


Figura 2.27 – Opções de Configuração

Por fim, ajuste a configuração de porta conforme sua necessidade.

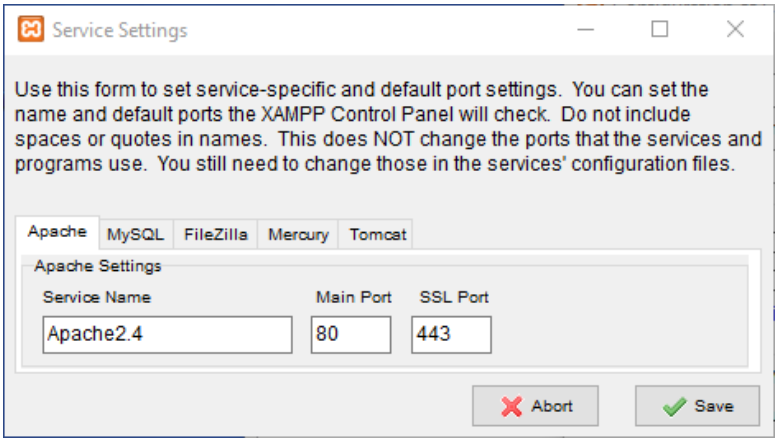


Figura 2.28 – Configurações de serviço

Outra funcionalidade importante é a inicialização do servidor Web Apache (ou outro servidor escolhido). Para isso, clique em “Start” (Iniciar), conforme a Figura 2.29.

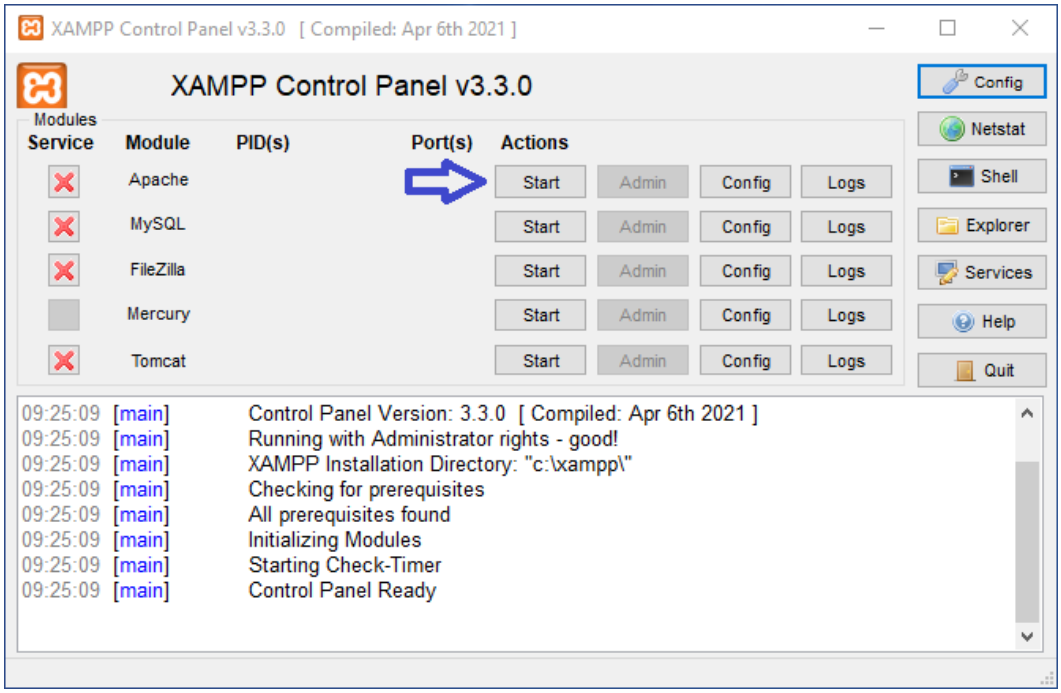


Figura 2.29 – Inicialização do Apache

Após a inicialização do servidor Web Apache, sua descrição fica destacada em fundo verde e são exibidas as portas HTTP (80) e HTTPS (443), que estão configuradas.

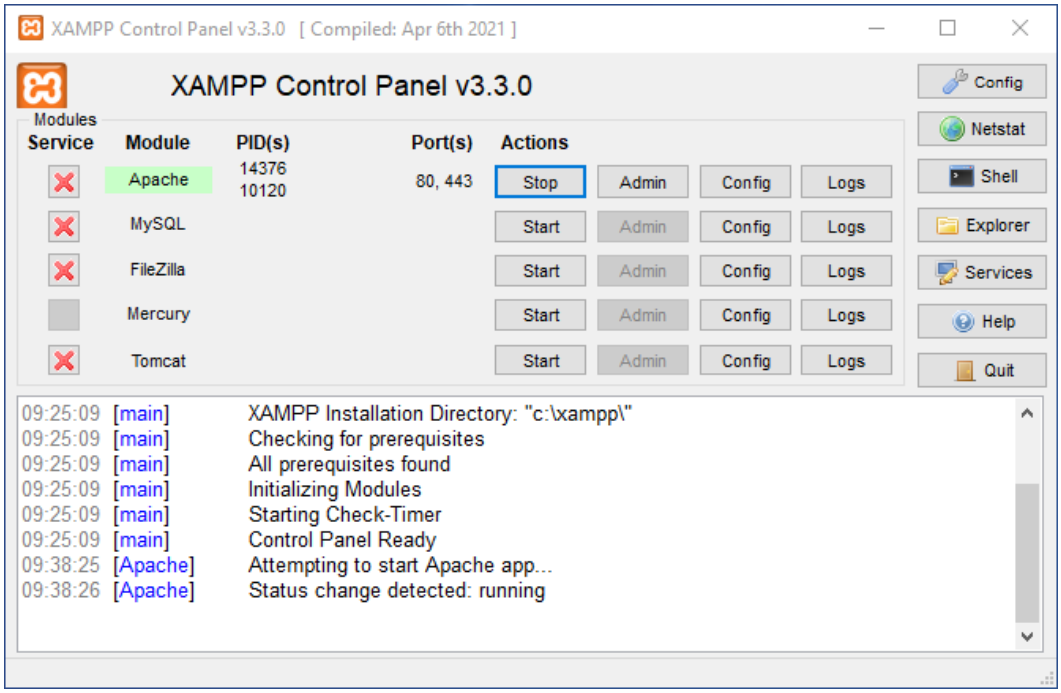


Figura 2.30 – Módulo Apache iniciado

Saiba mais:

A porta, também chamada de porta TCP (Transmission Control Protocol) ou UDP (User Datagram Protocol), é um software que permite a comunicação em rede. Ela está associada com o endereço IP (Internet Protocol) e o tipo de protocolo utilizado para troca de dados. A porta também auxilia em questões de segurança, pois permite o bloqueio de protocolos específicos. No processo de desenvolvimento de aplicações Web utiliza-se a porta HTTP 80, para a transferência de páginas WWW, e a porta HTTPS 443, para transmissão HTTP segura.

Ao clicar no botão “Admin”, que está na linha do servidor Web Apache, a página inicial do XAMPP será exibida em seu navegador.

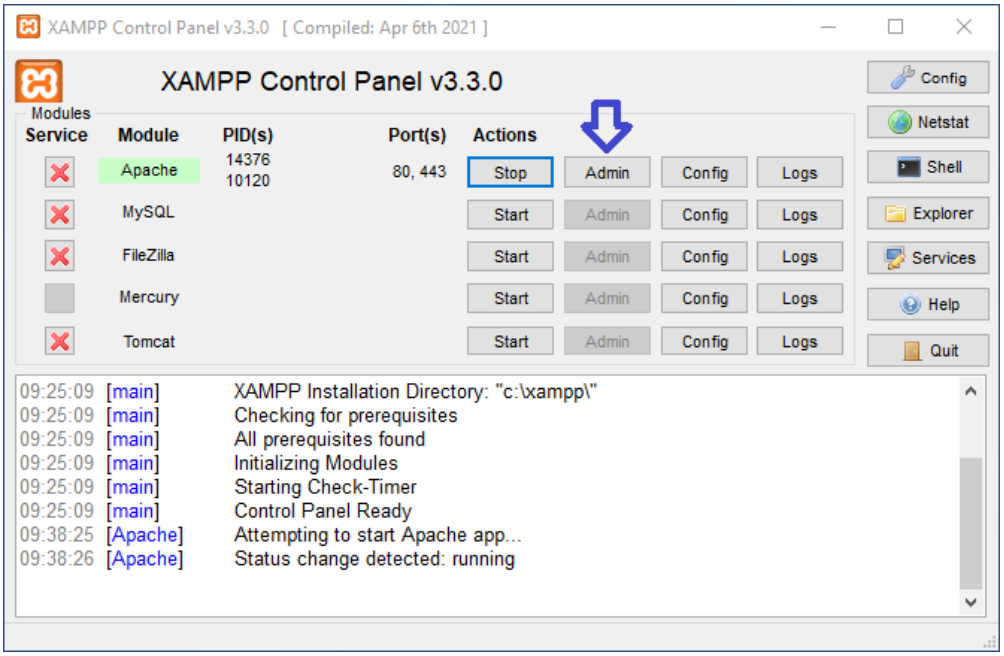


Figura 2.31 – Botão Admin do Módulo Apache

É possível obter o mesmo resultado do clique no botão “Admin” digitando o endereço (http://localhost) diretamente em seu navegador. **Importante**: Neste exemplo, a porta 80 está configurada para o protocolo HTTP. Por ser a porta padrão, ela não precisa ser incluída no endereço. Se a porta 80 for alterada, por exemplo para 8080, essa nova porta deve fazer parte do endereço (http://localhost:8080).

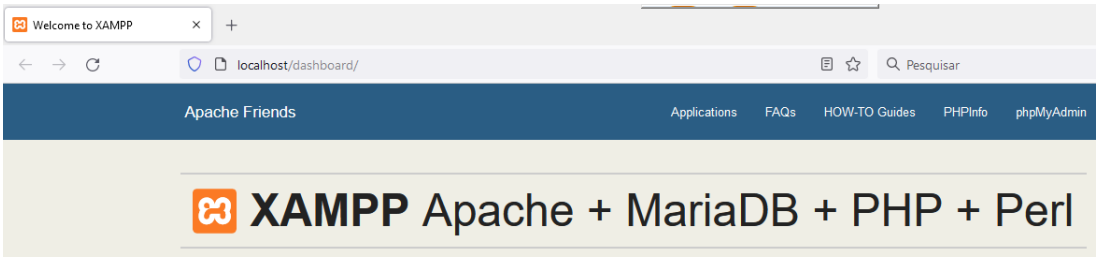


Figura 2.32 – Página inicial do XAMPP

Por meio do painel de controle também é possível acessar a estrutura de pastas do XAMPP. Para isso, utilize o botão “Explorer” (Explorar).

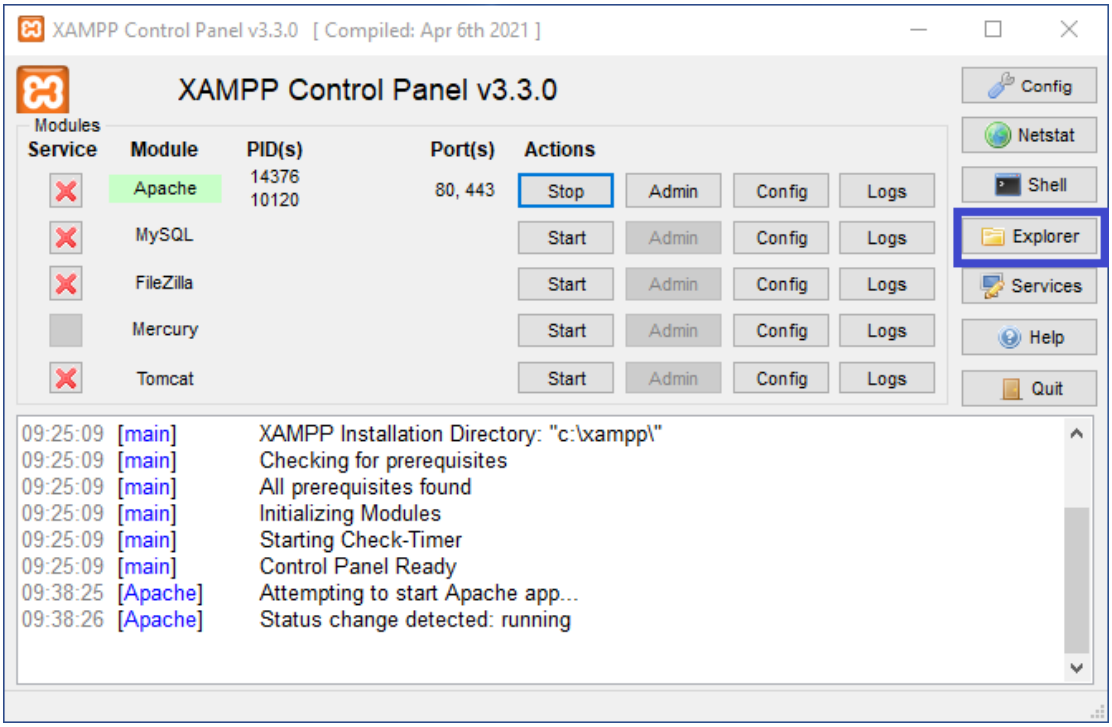


Figura 2.33 – Acesso a estrutura de pastas do XAMPP

Nessa estrutura, uma pasta bastante relevante é a “htdocs”. Nela são armazenadas as aplicações desenvolvidas, organizadas em pastas, e contendo os códigos fontes, arquivos de configuração e imagens.

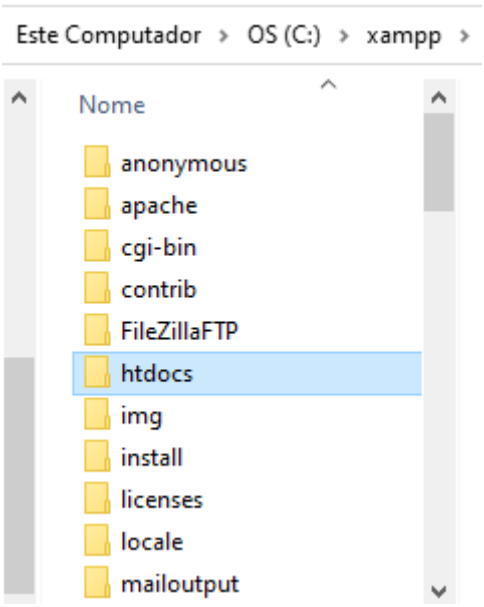


Figura 2.34 – Estrutura de pastas do XAMPP

2.3. Exercícios propostos

Exercício 1. O PHP é:

a) Uma linguagem de script executada no lado cliente.
b) Uma linguagem de script executada no lado servidor.
c) Uma linguagem de script executada tanto no lado cliente quanto no lado servidor.
d) Uma linguagem de script executada pelo próprio navegador do usuário.
e) Nenhuma das alternativas anteriores.

Exercício 2. As aplicações desenvolvidas utilizando-se PHP são interpretadas:

a) Pelo navegador do usuário
b) Pelo servidor Web.
c) Pelo módulo PHP instalado no servidor Web.
d) Pelo módulo CGI instalado no servidor Web.
e) Nenhuma das alternativas anteriores.

Exercício 3. Os scripts PHP são delimitados por:

a) </php e ?>
b) (/php e ?)
c) <!php e !>
d) <#php e #>
e) <?php e ?>

Capítulo 3. Princípios Básicos de PHP

Agora que você já sabe instalar e configurar um servidor Web, iremos aprender os princípios básicos da linguagem PHP. Para isto, inicialmente, mostraremos a estrutura básica de um script PHP e como visualizar o resultado de sua execução. Em seguida, exploraremos alguns dos fundamentos da linguagem, tais como comandos de entrada e saída de dados, manipulação de dados, variáveis, interpolação de variáveis, operadores e constantes. Ao final, você encontrará uma lista de exercícios para aplicar e praticar os conceitos vistos neste capítulo.

3.1. Estrutura de um Programa PHP

Normalmente, o início da jornada de aprendizagem de uma linguagem de programação inicia-se com um programa que imprime a mensagem “Hello world!” ou “Olá mundo!”, no português. Aqui, daremos continuidade a essa tradição, criando nosso primeiro programa em PHP para aprendermos sua estrutura básica.

Comece criando um arquivo em seu IDE. Salve esse arquivo com o nome ***olamundo.php*** no diretório ***htdocs***, localizado dentro do diretório de instalação do XAMPP. Depois, escreva o código do Exemplo 3.1 dentro do arquivo:

```
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3    <head>
4      <meta charset="utf-8" />
5      <title>PHP</title>
6    </head>
7    <body>
8      <?php
9        // Primeiro programa em PHP
10       echo "Olá mundo!";
11     ?>
12   </body>
13 </html>
```

Exemplo 3.1 – Primeiro programa em PHP.

O próximo passo é obter o resultado da execução desse *script*. Antes, contudo, verifique se o seu servidor web está ativo, pois é necessário que esteja para que esse programa funcione. Abra o seu navegador e digite o seguinte endereço: ***localhost/olamundo.php***. Você deverá observar a frase ***“Olá mundo!”*** sendo exibida em seu navegador, assim como na Figura 3.1.

Figura 3.1 - Saída do código descrito no Exemplo 3.1 no navegador Web.

Vamos ver, brevemente, como cada linha de código PHP do script funciona. Em primeiro lugar, nota-se que um programa escrito em PHP pode possuir comandos em HTML e códigos em PHP. Os comandos HTML devem aparecer fora das tags ***<?php*** e ***?>***, pois essas tags delimitam um trecho de programa PHP. Isso indica ao servidor Web o que deve ser processado nesta página.

Dentro das tags ***<?php*** e ***?>***, na linha 9, há um texto com os símbolos ***//*** no início. Estes símbolos indicam que o texto que vem em seguida, na mesma linha, é apenas um comentário e será ignorado durante o processamento do código PHP. Comentários em PHP também podem compreender várias linhas. Para isto, pode-se utilizar o ***/**** para começar os comentários e depois finalizá-los com o ****/***. É uma boa prática de programação utilizar comentários para documentar o seu programa.

Na linha 10, utilizamos o comando ***echo***. Esse é um dos comandos mais utilizados em scripts PHP e serve para exibir dados na tela. Mais precisamente, o comando ***echo*** irá imprimir os dados que estão à sua frente no código HTML gerado pelo servidor Web. Lembre-se que este HTML gerado é depois enviado ao navegador do cliente, que irá processá-lo e exibi-lo. Em nosso exemplo, os dados impressos pelo comando ***echo*** consistem no texto ***"Olá mundo!"***, que é exibido no navegador.

As linhas de programação dentro das tags ***<?php*** e ***?>*** devem sempre terminar com ***;*** (ponto e vírgula). Caso isso não aconteça, ocorrerão erros no momento de execução da página. Entre as tags delimitadoras do script PHP, é permitido escrever programas utilizando todos os recursos que o PHP oferece, tais como definição e chamada de funções,

acesso a banco de dados, atribuição de valores a variáveis, fórmulas matemáticas, entre outros.

Dica:

Para visualizar o código HTML provindo do servidor Web, basta clicar com o botão direito do mouse sobre a página web e escolher a opção de exibir o código-fonte da página. Você notará que o navegador não recebe nenhuma linha de código PHP, mas apenas HTML puro. Isto porque todo o script PHP é processado no servidor Web, que retorna somente o resultado para o navegador.

Dica:

Páginas salvas com a extensão .html, são tratadas pelo servidor web como se tivessem apenas código HTML, ou seja, não são processadas no servidor. Quando uma página possuir a extensão .php, o servidor web ativará o processador de hipertexto do PHP para verificar linha a linha em busca de códigos de programação. Como consequência do processamento, o retorno dessa página é mais lento. Por isso, só coloque a extensão .php nas páginas que realmente possuem códigos PHP, senão será gasto um tempo desnecessário, procurando, em cada linha, códigos que não existem na página.

É possível também escrever um programa concatenando vários scripts PHP com comandos HTML. Essa mistura entre HTML e scripts PHP é muito útil, pois pode-se utilizar o PHP para gerar os dados dinamicamente, enquanto o HTML é usado para formatar e exibir esses dados nas páginas apresentadas no browser. No exemplo mostrado no Exemplo 3.2, misturamos HTML e PHP para exibir uma mensagem com o nome de uma pessoa.

```
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3    <head>
4      <meta charset="utf-8" />
5      <title>PHP</title>
6    </head>
7    <body>
8      <?php
9        $nome = "Mega Games";
10     ?>
11     Seja bem-vindo à
12     <p align="center">
13       <?php echo $nome; ?>
14     </p>
15   </body>
16 </html>
```

Exemplo 3.2 – Uso do HTML e do PHP para exibir uma mensagem.

Note, nesse exemplo, que combinamos comandos HTML com códigos PHP. Na linha 9, atribuímos o valor ***"Mega Games"*** à variável **$nome**. Em PHP, variáveis começam sempre pelo símbolo de cifrão (**$**). Depois, continuamos com código HTML, imprimindo o texto ***"Seja bem-vindo à"***. Em seguida, utilizamos comandos HTML para gerar um parágrafo, onde, na linha 13, abrimos um trecho de código PHP para imprimirmos o valor da variável **$nome**, por meio do comando ***echo***. Para o interpretador PHP, todos os scripts PHP dentro de uma mesma página formam um único programa.

3.2. Comandos de Saída de Dados

Considere o programa em PHP descrito na Exemplo 3.3, que calcula o preço final de um produto.

```
1  <!DOCTYPE html>
2  <html lang="pt-BR">
3    <head>
4      <meta charset="utf-8" />
5      <title>Calcula frete</title>
6    </head>
7    <body>
8      <?php
9        $precoproduto = 105.5;
10       $frete = 10.0;
11       $precofinal = $precoproduto + $frete;
12     ?>
13   </body>
14 </html>
```

Exemplo 3.3 – Programa que calcula o preço final de um produto.

O que será exibido pelo navegador ao solicitar essa página? Se você respondeu nada, acertou! Um script PHP apenas consegue exibir uma informação na página se houver um comando de saída de dados, tal como o ***echo***.

Comandos para exibição de dados são fundamentais em qualquer linguagem de programação. Em PHP, veremos três funções capazes de imprimir dados, cada uma com suas próprias características.

3.2.1 Comando *echo*

O comando ***echo*** é o mais frequentemente utilizado em linguagem PHP. Ele é capaz de exibir qualquer tipo de informação, tais como textos, números ou variáveis. No Exemplo 3.4 estão alguns exemplos de seu uso.

```
1  <?php
2    $nome = "Mega Games";
3    echo $nome;
4    echo "<br/>";
5    echo "A $nome";
6    echo " é a sua loja de jogos";
7  ?>
```

Exemplo 3.4 – Uso do comando echo.

Note as diferentes utilizações do comando ***echo***. Na linha 3, o comando ***echo*** imprime o valor da variável **$linguagem**. Na linha 4, o comando imprime a tag ***
*** no código HTML, gerando assim, uma mudança de linha na página. Nas linhas 5 e 6, o valor da variável **$linguagem** é exibido em meio a um a uma frase.

3.2.2 Comando *print*

O comando ***print*** é utilizado da mesma maneira que o comando ***echo***. Contudo, o comando ***print*** possui a capacidade de retornar um valor, que pode ser usado para verificar se a mensagem foi realmente exibida. Devido à essa característica, o comando ***print*** torna-se um pouco mais lento que o comando ***echo***. O Exemplo 3.5 mostra um exemplo de uso do ***print***.

```php
<?php
  if (print "Mega"){
    print " Games";
  }
?>
```

Exemplo 3.5 – Uso do comando print.

Neste exemplo, caso o script seja bem-sucedido ao imprimir a mensagem ***"Mega"***, será impresso também a mensagem ***"Games"***.

3.2.3 Comando *printf*

A função ***printf*** tem uso semelhante à função de mesmo nome da linguagem C. Seu propósito é exibir no navegador uma cadeia de caracteres (*string*) formatada. O Exemplo 3.6 mostra um exemplo de uso do ***printf***.

```php
<?php
  $texto = "Games";
  printf("Mega %s", $texto);
?>
```

Exemplo 3.6 – Uso do comando printf.

No exemplo, o valor da variável **$linguagem** será inserido no lugar do especificador ***%s*** na *string* (entre aspas). De forma mais geral, os valores passados como argumentos na função são inseridos na *string* por meio de especificadores com sinais de porcentagem (***%***). Os especificadores mais utilizados para formatar uma *string* no comando ***printf*** são: ***%c*** para caracteres; ***%d*** para números decimais; ***%f*** para números em ponto flutuante; e ***%s*** para *strings*.

3.3. Tipos de dados

Para avançarmos no aprendizado de PHP, vamos agora entender como manipular apropriadamente alguns dos tipos de dados utilizados na linguagem. Aqui, iremos considerar os tipos mais básicos, incluindo os numéricos, literais e booleanos. *Arrays* serão explicados no Capítulo 6. Outros tipos, tais como objetos e iteráveis, não serão abordados.

3.3.1. Dados numéricos

Dados do tipo numérico são dados que expressam quantidades e permitem efetuar diversos tipos de cálculos aritméticos. Eles são representados por números em diferentes notações, dependendo da grandeza e da base numérica utilizadas.

Os dados numéricos mais utilizados são aqueles nas bases decimal, octal e hexadecimal, além dos números reais. A Tabela 3.1 exemplifica a representação de cada um dos tipos de dados numéricos.

Dado numérico	**Descrição**
192	Valor inteiro representado na base decimal. Nesta base, o número deve utilizar os algarismos 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9.
034	Valor inteiro representado na base octal. Nesta base, o número deve iniciar com 0 (zero) e utilizar apenas os algarismos 0, 1, 2, 3, 4 ,5, 6 e 7.
0xFE	Valor inteiro representado na base hexadecimal. Nesta base, o número deve iniciar com 0x e utilizar os algarismos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F.
3.1415	Valor real (ponto flutuante) com quatro casas decimais.
.386	Valor real com três casas decimais. É equivalente ao número 0.386.
43000000	Valor real grande que pode ser expresso em notação científica: 4.3E+7.

Tabela 3.1 - Representação dos dados numéricos.

3.3.2. Dados literais

Dados literais são sequências de caracteres delimitadas por aspas simples ou aspas duplas, sendo também conhecidas como dados alfanuméricos ou *strings*. No Exemplo 3.7, a sequência de caracteres entre aspas será exibida da mesma forma em ambos os comandos ***echo***.

```
1  <?php
2    echo 'Bem-vindo à nossa loja on-line!';
3    echo "Bem-vindo à nossa loja on-line!";
4  ?>
```

Exemplo 3.7 - Exibição de dados literais.

A escolha do tipo de delimitador, contudo, impacta na forma como o interpretador PHP trata as *strings*. Veremos agora as especificidades do uso de cada tipo de aspas em sequências de caracteres.

3.3.2.1 Aspas simples

Ao utilizar uma sequência de caracteres delimitadas por aspas simples, é preciso ter cuidado com textos que possuam o caractere **‘**, indicando a presença de expressões entre aspas simples ou que faça uso do sinal apóstrofo. O motivo do problema é que o interpretador PHP irá interpretar o caractere **‘** como um delimitador, causando um erro de execução. Veja o Exemplo 3.8.

```
1  <?php
2    echo 'Garrafa d'água';
3  ?>
```

Exemplo 3.8 – Uso incorreto do apóstrofo com as aspas simples.

Nesse exemplo, a expressão ***“d'água”*** faz uso do apóstrofo para realizar supressão na expressão ***“de água”***. O interpretador PHP entende o apóstrofo como um delimitador da *string* ***“Garrafa d”***, deixando a palavra ***“água”*** fora da sequência de caracteres. Como a palavra ***“água”*** é um elemento desconhecido para o interpretador PHP, o programa irá apresentará um erro ao ser executado.

O problema pode ser resolvido pela inserção do caractere de controle **** (barra invertida) antes das aspas ou do apóstrofo. A **** indica ao interpretador PHP que o caractere **‘** deve ser interpretado como um texto comum e não como um delimitador. O caractere **“** pode ser usado livremente dentro de uma *string* delimitada por aspas simples. Veja o Exemplo 3.9.

```
1  <?php
2    echo 'Garrafa d\'água';
3  ?>
```

Exemplo 3.9 – Uso do caractere de controle com aspas simples.

Dica:

Para fazer quebras de linha em uma *string*, use a tag ***
*** do HTML. O uso do caractere especial ***\n*** não terá efeito na exibição do conteúdo da página Web.

3.3.2.2 Aspas duplas

Assim como acontece com as aspas simples, é preciso ter cuidado ao utilizar o caractere **“** dentro de uma *string* delimitada por aspas duplas. Esse uso também poderá confundir o interpretador PHP e causar um erro de execução. Para evitar esse problema, basta utilizar o caractere de controle **** antes do sinal de aspas duplas para que o resultado desejado seja obtido. Veja o Exemplo 3.10.

```
1  <?php
2    echo "\"Penso, logo existo\" (René Descartes)";
3  ?>
```

Exemplo 3.10 – Uso do caractere de controle com as aspas duplas.

Uma outra característica das aspas duplas como delimitadores é a possibilidade do uso da interpolação de variáveis, A interpolação de variáveis consiste na inclusão de valores de uma variável dentro de uma *string*. No Exemplo 3.11 é ilustrado esse conceito.

```
1  <?php
2    $nomeloja = "Mega Games"
3    echo "Bem-vindo ao site da $nomeloja";
4  ?>
```

Exemplo 3.11 - Interpolação de variáveis utilizando aspas duplas.

Nesse exemplo, a variável **$nomeloja** possui o valor ***“Mega Games”***. Ao executar o comando ***echo*** da linha 3, será exibida a mensagem ***“Bem-vindo ao site da Mega Games”***, com a variável **$nomeloja** sendo substituída pelo seu valor.

Cabe observar que a interpolação de variáveis ocorre apenas quando os delimitadores utilizados são as aspas duplas. Para obter o mesmo efeito com o uso de aspas simples, é preciso utilizar o operador **.** (concatenação), que será estudado posteriormente. Além disso, não é possível utilizar a interpolação para *arrays* e funções.

Dica:

Para utilizar o caractere ***$*** dentro de uma *string* delimitada por aspas duplas, use a sequência de controle **\$**. Caso queira utilizar o caractere ****, use ****.

3.3.3. Dados lógicos

O tipo de dado lógico ou booleano permite representar os valores verdadeiro e falso. Este tipo de dado é bastante utilizado em estruturas de controle condicional, estruturas de repetição e retorno de funções.

Os possíveis valores de um dado booleano são ***true*** (verdadeiro) ou ***false*** (falso). Em PHP, esses valores são insensíveis ao caso, podendo ser escritos de diferentes maneiras, como, por exemplo, ***False***, ***FALSE***, ***false***, ***true*** ou ***TRUE***.

3.4. Variáveis

As variáveis são responsáveis por armazenarem valores de dados que podem ser utilizados ao longo do programa. As variáveis são armazenadas na memória do computador, sendo que cada uma delas possui um endereço de memória associado. Em PHP, não é necessário realizar a declaração de variáveis, como ocorre em outras linguagens de programação, como C, Pascal e Java. Para criar uma variável, basta atribuí-la um valor. O tipo do valor atribuído (numérico, literal, lógico, entre outros) irá determinar também o tipo associado àquela variável.

Em PHP, uma variável começa sempre com o símbolo **$**. Após este símbolo, deve haver um identificador, que é o nome da variável. Este nome servirá para referenciar essa variável durante a execução do programa.

O nome de uma variável deve iniciar com uma letra ou o caractere sublinhado (_) apenas, nunca um número. Os números podem aparecer nas demais posições do nome da variável. Na Tabela 3.2, são apresentados alguns exemplos de identificadores válidos e inválidos para variáveis em PHP.

Válidas	$nota1
	$casal20
	$bisc8
Inválidas	$100vergonha
	$5
	$20assustar
	$60nacadeira

Tabela 3.2 - Exemplos de identificadores válidos e inválidos.

Fonte: Niderauer (2017).

Cuidado!

Para nomes de variáveis, a linguagem PHP faz distinção entre letras minúsculas e maiúsculas. Se houver uso dos dois tipos de letras, pode ocorrer confusão na utilização da variável. Por exemplo, a variável **$nota_aluno** não é mesma coisa que **$Nota_aluno**.

Variáveis podem conter valores numéricos, alfanuméricos (*strings*), *arrays* ou objetos. Variáveis numéricas possuem valores inteiros (decimal, hexadecimal, octal ou binário) ou reais (ponto flutuante). A atribuição de valores numéricos a variáveis é apresentada no Exemplo 3.12.

```php
<?php
  $numero_inteiro = 23;
  $numero_hexadecimal = 0x0c;
  $numero_octal = 0234;
  $numero_binario = 0b10101010;
  $numero_float = 3.14;
?>
```

Exemplo 3.12 - Atribuição de valores numéricos a variáveis.

Variáveis alfanuméricas podem recebem cadeias de caracteres delimitadas por aspas simples ou aspas duplas, conforme o Exemplo 3.13.

```php
<?php
  $string_aspas_simples = 'Mega Games';
  $string_aspas_duplas = "A sua loja de jogos";
?>
```

Exemplo 3.13 - Atribuição de valores alfanuméricos a variáveis.

Variáveis que armazenam *arrays* serão vistas posteriormente e variáveis que armazenam objetos não serão abordadas neste livro.

Uma das características da linguagem PHP é que ela possui tipagem de dados fraca. Isto significa que uma variável pode receber valores de diferentes tipos durante o programa. Veja o Exemplo 3.14.

```php
<?php
  $produto = 34584;
  $produto = 14.56;
  $produto = "camiseta";
?>
```

Exemplo 3.14 – Tipagem de dados fraca no PHP.

Nesse exemplo, o tipo da variável $produto muda a cada atribuição de valor. Na linha 2, a variável é do tipo numérico inteiro. Na linha 3, a variável $produto é do tipo numérico de ponto flutuante (***float***) e na linha 4, do tipo *string*.

Apesar desta característica da linguagem, por vezes, é preciso fazer a conversão manual de tipos, para que se obtenha um determinado valor no tipo desejado. Esta operação utiliza um conversor, que deve aparecer entre parênteses antes de uma variável ou de uma expressão. Veja o Exemplo 3.15.

```
1  <?php
2    $x = 325.45;
3    $y = 75;
4    $total = (int) $x + $y;
5  ?>
```

Exemplo 3.15 - Conversão de tipo de uma variável.

Na linha 4 do exemplo, a variável **$x**, que é do tipo ***float***, tem o seu valor convertido para inteiro por meio do conversos ***int***. Isto significa que a parte fracionária do valor de **$x** não é considerada pela soma e, portanto, a variável **$total** irá receber o valor 400. Caso seja necessário converter o valor de toda a expressão aritmética para inteiro, basta colocá-la toda entre parênteses, conforme o Exemplo 3.16.

```
1  <?php
2    $total = (int) ($x + $y);
3  ?>
```

Exemplo 3.16 - Conversão do tipo da expressão toda.

É possível realizar a conversão para outros tipos de dados em PHP. Na Tabela 3.3 são mostrados os conversores disponíveis para os tipos básicos da linguagem.

Conversor	Descrição
(**int**), (**integer**)	Converte para inteiro
(**real**), (**float**), (**double**)	Converte para ponto flutuante
(**string**)	Converte para *string*
(**array**)	Converte para *array*

Tabela 3.3 - Conversores de tipos.

3.5. Operadores

Um operador tem o objetivo de transformar dados ou expressões fornecidas em um outro valor. A linguagem PHP trabalha com três grupos de operadores: unários, binários e ternários.

Operadores unários manipulam apenas um valor e são utilizados para operações como negação (***!***) e incremento (**++**). Operadores binários, que representam a maioria dos operadores da linguagem PHP, manipulam dois valores e retornam um valor. Exemplos são o operador lógico ou (***OR***) e o operador de comparação maior ou igual (**>=**). O operador ternário retorna um resultado dentre dois possíveis, dada a avaliação de um terceiro valor ou expressão.

Os três grupos de operadores são classificados em diferentes tipos: operadores aritméticos; operadores de comparação; operadores de atribuição; operadores lógicos; e operador ternário.

3.5.1 Operadores Aritméticos

Este tipo define operações baseadas nas operações aritméticas básicas, que são adição, subtração, divisão e multiplicação. A Tabela 3.4 mostra os operadores binários definidos nesse tipo.

Operador	Operação	Exemplo
+	Adição	$valor1 + $valor2
-	Subtração	$valor1 - 100
*	Multiplicação	10 * $valor
/	Divisão	$valor / 5
%	Resto	$valor % 2

Tabela 3.4 - Operadores aritméticos binários.

Vale observar neste ponto que o operador de divisão ***"/"*** retorna um valor em ponto flutuante (***real***), mesmo que a operação tenha sido calculada sobre dois valores inteiros.

Alguns operadores unários também realizam operações aritméticas. Estes operadores são úteis para a simplificação de algumas operações comuns na programação, tais como incrementos e decrementos de valores. A Tabela 3.5 exibe os operadores aritméticos unários da linguagem PHP.

Operador	Operação	Exemplo
-operando	Troca o sinal do operando	-$valor

++operando	Pré-incremento	++$valor
--operando	Pré-decremento	--$valor
operando++	Pós-incremento	$valor++
operando--	Pós-decremento	$valor--

Tabela 3.5 - Operadores aritméticos unários.

Apesar de muito parecidos, os operadores de pré-incremento (e pré-decremento) e pós-incremento (e pós-decremento) atuam de forma diferente em uma avaliação de uma expressão. Em operadores "***pré***", o valor do operando é modificado antes da avaliação da expressão em que essa operação está inserida. Já no caso de operadores "***pós***", o valor do operando é modificado após a avaliação da expressão em que a operação está inserida. Vamos compreender melhor essa diferença por meio do Exemplo 3.17.

```
1  <?php
2    $x = 1;
3    $y = 3;
4    $z = 5;
5
6    $r1 = ++$y - $x;
7    $r2 = $z-- + $x;
8    $r3 = --$x + $z++;
9
10   echo "x = $x <br/>";
11   echo "y = $y <br/>";
12   echo "z = $z <br/>";
13
14   echo "r1 = $r1 <br/>";
15   echo "r2 = $r2 <br/>";
16   echo "r3 = $r3";
17 ?>
```

Exemplo 3.17 – Exemplo de operações com pré e pós incremento.

Esse programa em PHP, possui a seguinte saída:

x = 0

y = 4

z = 5

r1 = 3

r2 = 6

r3 = 4

Vamos, primeiramente, analisar a expressão contida na linha 6. Nela, a variável **$r1** recebeu o valor gerado pela expressão **++$y - $x**. Como a variável **$y** foi pré-incrementada, seu valor passou de 3 para 4 e, somente após esse incremento, o valor da variável **$x** foi subtraído. Dessa forma, o resultado armazenado em **$r1** foi 3.

Na linha 7, a variável **$r2** recebeu o valor gerado pela expressão **$z-- + $x**. Nela, primeiramente, o resultado de **$z + $x (5 + 1)** é calculado e armazenado em **$r2**. Logo após, ocorre o decremento da variável **$z**, que passa do valor 5 para o valor 4.

Na linha 8, a variável **$r3** recebeu o valor gerado pela expressão **--$x + $z++**. Antes de qualquer outra operação, é feito o pré-decremento do valor da variável **$x**, passado de 1 para 0. Em seguida, o novo valor de **$x** é somado com **$z**, tendo como resultado 4. Finalmente, a variável **$z** é incrementada e fica com o valor 5.

3.5.2 Operadores de Comparação

Os operadores de comparação realizam, como o próprio nome diz, uma comparação entre dois valores. Tais operadores retornam um valor lógico como resultado, podendo ser verdadeiro (***true***) ou falso (***false***). A Tabela 3.6 mostra uma lista de operadores de comparação.

Operador	Operação	Exemplo
==	Igualdade	$a == $b (Se $a for igual a $b, retorna verdadeiro)
===	Idênticos	$a === $b (Se $a e $b forem iguais e forem do mesmo tipo, retorna verdadeiro)
!= ou <>	Diferente	$a != $b ou $a <>$b (Se $a for diferente de $b, retorna verdadeiro)
!==	Não idênticos	$a !== $b (Se $a e $b forem diferentes e não forem do mesmo tipo, retorna verdadeiro)
<	Menor	$a < $b (Se $a for menor que $b, retorna verdadeiro)
<=	Menor ou igual	$a <= $b (Se $a for menor ou igual a $b, retorna verdadeiro)
>	Maior	$a > $b (Se $a for maior que $b, retorna verdadeiro)
>=	Maior ou igual	$a >= $b (Se $a for maior ou igual a $b, retorna verdadeiro)

Tabela 3.6 - Operadores aritméticos unários.

3.5.3 Operadores de Atribuição

Um operador de atribuição é utilizado para atribuir o valor resultante da expressão à sua direita ao operando que está à sua esquerda, o qual é, geralmente, uma variável.

O operador de atribuição mais comum é o de sinal de igual (**=**). Esse operador pode também ser utilizado em operações combinadas, conforme mostrado no Exemplo 3.18.

```php
<?php
  $desconto = ($venda = 1000) * 0.2;
?>
```

Exemplo 3.18 – Exemplo de operação de atribuição.

No exemplo, a variável **$venda** recebe o valor 1000 e à variável **$desconto** será atribuído o resultado da expressão **$venda** * 0.2, cujo resultado é 200.

Além o uso da atribuição nesse formato, é possível combinar a atribuição com operadores aritméticos e de concatenação, conforme mostrado na Tabela 3.7.

Operador	Operação	Exemplo
=	Atribuição	$a = $b + 10
+=	Atribuição e adição	$a += 10 (equivalente a $a = $a + 10)
-=	Atribuição e subtração	$a -= 10 (equivalente a $a = $a - 10)
*=	Atribuição e multiplicação	$a *= 10 (equivalente a $a = $a * 10)
/=	Atribuição e divisão	$a /= 10 (equivalente a $a = $a / 10)
%=	Atribuição e módulo	$a %=2 (equivalente a $a = $a % 10)
.=	Atribuição e concatenação	$texto .= $t (equivalente a $texto = $texto . $t)

Tabela 3.7 - Combinação de atribuição e operadores aritméticos.

3.5.4 Operadores lógicos

Operadores lógicos são utilizados para realizar comparações entre expressões, retornando os valores verdadeiro (***true***) ou falso (***false***) como resultado. Em geral, as operações são binárias, sendo necessários dois operandos. A exceção a essa regra é o operador ***!*** (negação), que é unário. A Tabela 3.8 a seguir mostra os diferentes operadores lógicos da linguagem PHP.

Operador	Operação	Exemplo
AND	E	$a AND $b (verdadeiro se $a E $b forem verdadeiros)
OR	Ou	$a OR $b (verdadeiro se $a OU $b forem verdadeiros)
XOR	Ou exclusivo	$a XOR $b (verdadeiro se apenas $a ou apenas $b for verdadeiro)
&&	E	$a && $b (verdadeiro se $a E $b forem verdadeiros)
||	Ou	$a || $b (verdadeiro se $a OU $b forem verdadeiros)
!	Negação	!$a (verdadeiro se $a for falso)

Tabela 3.8 - Operadores lógicos.

Embora produzam resultados idênticos, os operadores ***AND*** e ***&&*** e os operadores ***OR*** e **||** diferenciam-se pela precedência. Os operadores ***&&*** e **||** possuem precedência mais alta.

3.5.5 Operador ternário

O operador ternário é uma forma abreviada de uso da estrutura condicional ***if***. Seu uso ocorre da seguinte forma:

condição ? expressão 1 : expressão 2

Nesse comando, incialmente, a condição é avaliada. Caso seja verdadeira, atribui-se o valor resultante da expressão 1. Caso contrário, atribui-se o valor da expressão 2, como apresentado no Exemplo 3.19.

```php
<?php
  $venda_final = ($com_desconto == true) ?
                 ($valor_produto * 0.2) : $valor_produto;
?>
```

Exemplo 3.19 – Exemplo utilizando operador ternário.

3.5.6 Precedência de operadores

Os operadores da linguagem PHP seguem uma ordem de precedência. Conhecer essa ordem é muito importante para que expressões possam ser criadas e compreendidas corretamente. Na Tabela 3.9 é apresentada a precedência dos operadores do PHP em ordem decrescente (da mais alta para a mais baixa).

Operador	Descrição
()	Parênteses
++ -- ! -	Incremento, decremento, negação e negativo
* / %	Multiplicação, divisão e resto
+ - .	Soma, subtração e concatenação
> < >= <=	Maio, menor, maior ou igual, menor ou igual
== != <>	Igualdade, desigualdade
&&	E lógico
||	OU lógico
?:	Operador ternário
= += -= *= /= %=	Operadores de atribuição
AND	E lógico (menor prioridade)
XOR	OU exclusivo (menor prioridade)
OR	OU lógico (menor prioridade)

Tabela 3.9 - Precedência de operadores.

Primeiramente, o interpretador PHP irá executar todas as operações que estiverem entre parênteses. Caso dentro dos parênteses haja outras operações, a ordem que estas serão executadas é definida pela precedência de operadores descrita na tabela. Se em uma expressão houver operadores de mesma prioridade e não existirem parênteses, o PHP resolverá a expressão da esquerda para a direita. No Exemplo 3.20, o resultado mostrado será x = 6 e y = 20.

```php
<?php
  $x = 5;
  $y = 8 + 3 * 2 + ++$x;
  echo "x = $x <br/> y = $y";
?>
```

Exemplo 3.20 – Exemplo de precedência de operadores em uma expressão.

3.6. Constantes

Constantes são valores predefinidos e que não mudam ao longo da execução do programa. Em PHP, constantes são criadas utilizando o comando define e um par nome/valor. Veja o Exemplo 3.21, com a criação e o uso de constantes.

```php
<?php
  define ("desconto_prata", 10);
  define ("desconto_ouro", 20);
  echo "O valor do desconto prata é ". desconto_prata . "<br/>";
  echo "O valor do desconto ouro é ". desconto_ouro;
?>
```

Exemplo 3.21 – Exemplo de uso de constantes.

É importante observar que o nome de uma constante não é precedido pelo símbolo **$**, como acontece com as variáveis. Assim, não é possível realizar interpolação com constantes dentro de aspas duplas. Neste caso, deve-se utilizar o operador de concatenação (**.**), conforme mostrado no Exemplo 3.21.

Algumas constantes são predefinidas na linguagem PHP, tais as apresentadas na Tabela 3.10.

Constante	Descrição
TRUE	Valor verdadeiro
FALSE	Valor falso
__FILE__	Nome do script que está sendo executado
__LINE__	Linha do script que está sendo executado
PHP_VERSION	Versão do PHP
PHP_OS	Sistema operacional no qual o PHP está sendo executado

Tabela 3.10 – Constantes da linguagem PHP.

3.7. Exercícios propostos

Exercício 1. Faça um programa em PHP que armazena os seguintes dados em variáveis e utiliza comandos ***echo*** para apresentar os dados armazenados nessas variáveis:

- Nome (atribua à variável o seu nome);
- Idade (atribua à variável sua idade);
- Sexo (atribua à variável o seu sexo);
- Endereço (atribua à variável o seu endereço);
- Cidade (atribua à variável a sua cidade);
- Telefone (atribua à variável o seu telefone).

Exercício 2. Um funcionário recebe um salário fixo de R$ 1500,00 mais 4% de comissão sobre as suas vendas. Faça um programa em PHP para calcular e mostrar a comissão e o salário final do funcionário, sabendo-se que ele vendeu R$10.000,00.

Exercício 3. Sabe-se que um(a) aluno(a) da disciplina IEB do Curso Técnico em Informática obteve nota 9.0 na primeira prova, 7.5 na segunda prova e 6.0 no projeto prático. Sabe-se ainda que o peso da primeira prova é 3, da segunda prova 4 e do projeto prático 3. Faça um programa em PHP para calcular e apresentar a nota final do aluno no bimestre.

Exercício 4. Dado que uma pessoa nasceu no ano de 1999 e o ano atual é 2017, faça um programa em PHP que calcule e mostre:

- A idade da pessoa em anos;
- A idade da pessoa em meses;
- A idade da pessoa em dias;
- A idade da pessoa em semanas.

Exercício 5. Sabe-se que uma pessoa fez um depósito de R$ 4000,00 em sua conta poupança. Sabe-se também que a taxa de juros foi de 0.67% ao mês. Faça um programa em PHP para calcular e mostrar o valor do rendimento após um mês e o valor total depois do rendimento.

Exercício 6. Sabe-se que um garçom vai trabalhar 12 horas em uma festa de Carnaval e que o valor do salário-mínimo é de R$ 937,00 em 2017. Faça um programa em PHP para calcular o salário a receber do empregado, seguindo as seguintes regras:

- A hora trabalhada vale 5% do salário-mínimo;
- O salário bruto equivale ao número de horas trabalhadas multiplicado pelo valor da hora trabalhada;
- O imposto equivale a 3% do salário bruto;
- O salário a receber equivale ao salário bruto menos o imposto.

Exercício 7. Dado que um espetáculo teatral tem o custo de R$ 2665,00 e o preço do convite esse espetáculo é R$ 65,00. Faça um programa em PHP para calcular e mostrar:

- A quantidade de convites que devem ser vendidos para que pelo menos o custo do espetáculo seja alcançado.
- A quantidade de convites que devem ser vendidos para que se tenha um lucro de 23%.

Exercício 8. Faça um programa em PHP para efetuar o cálculo da quantidade de combustível gasto em uma viagem, sabendo-se que o automóvel que faz 12Km por litro, que o tempo gasto foi de 2 horas e meia e a velocidade média durante a viagem foi 96 Km/hora.

Exercício 9. Sabe-se que o quilowatt de energia custa R$ 0,48 e que uma residência consumiu 137 quilowatts no mês de fevereiro de 2017. Faça um programa em PHP que calcule e mostre:

- O valor, em reais, a ser pago por essa residência.
- O valor, em reais, a ser pago com desconto de 15%.

Exercício 10. Um hotel deseja fazer uma promoção especial de final de semana, concedendo um desconto de 25% na diária. O hotel possui 42 apartamentos e o valor

normal da diária por apartamento é R$ 167,00. Implemente um programa em PHP para calcular:

- Valor promocional da diária;
- Valor total a ser arrecadado caso a ocupação neste final de semana (2 diárias por apartamento) atinja 100%;
- Valor total a ser arrecadado caso a ocupação neste final de semana atinja 70%;
- Valor que o hotel deixará de arrecadar em virtude da promoção, caso a ocupação atinja 100%.

Exercício 11. Faça um programa em PHP que calcule o índice de massa corpórea de uma pessoa com 67 kg de peso e a 172 cm de altura. O índice de massa corpórea mede a relação entre peso e altura (peso em Kg, dividido pelo quadrado da altura em metros).

Capítulo 4. Entrada de dados

Em sites e aplicações Web, a principal estratégia para permitir a entrada de dados é baseada no conceito de formulários em HTML. Assim como os formulários em papel que precisamos preencher em muitas situações, formulários na Web permitem que um determinado conjunto de dados seja informado e processado de alguma forma. Portanto, com base em formulários, aplicações recebem dados, efetuam algum tipo de processamento (por exemplo, consulta ao banco de dados) e devolvem um retorno (por exemplo, uma página de erro ou de sucesso). A comunicação entre cliente e servidor no momento que os dados de um formulário de autenticação são submetidos pelo usuário é ilustrada na Figura 4.1.

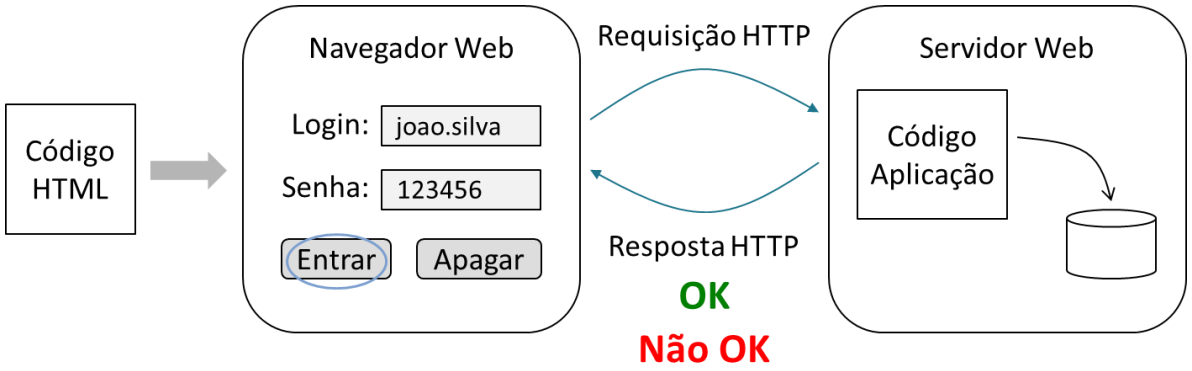


Figura 4.1 - Comunicação entre cliente e servidor na submissão de um formulário.

Conforme é mostrado na Figura 4.1, ao preenchermos e submetermos nossos dados (login e senha) para realizar a autenticação em uma aplicação Web, uma requisição HTTP é efetuada ao servidor que contém a aplicação responsável pelo processamento dos dados (por exemplo, um script em PHP). O código da aplicação é executado, e os dados são processados. Neste caso, o processamento envolve basicamente consultar o banco de dados para verificar se as credenciais informadas são válidas, isto é, se o login existe e se a senha informada é igual à senha cadastrada para o login. Em caso positivo, uma página de entrada poderia ser apresentada ao usuário; caso contrário, uma página de erro, ou ainda uma mensagem de erro, poderia ser exibida.

4.1. Formulários em HTML

Para a especificação de um formulário em HTML, devemos utilizar o elemento ***form***, composto pelas *tags* ***<form>*** e ***</form>*** delimitando a estrutura de um formulário. Propriedades do formulário são definidas por meio de atributos especificados na tag ***<form>***. Uma das propriedades fundamentais para qualquer formulário em HTML é o seu

destino, isto é, para onde os dados devem ser enviados assim que o formulário for submetido. O destino é definido por meio do atributo ***action***. O valor deste atributo pode ser o endereço (URL) de uma aplicação ou um endereço de e-mail. No último caso, o valor do atributo ***action*** deve ser precedido pela *string* "mailto:".

Outra propriedade de qualquer formulário é o método de envio de dados que será utilizado. O método de envio de dados é definido por meio do atributo **method**. Em HTML, os métodos possíveis são ***GET*** e ***POST***. O método ***GET*** é adequado para a solicitação de dados de um recurso na Web, assim, para o envio de dados, é um método com várias limitações. Ele permite apenas o envio de caracteres no padrão ASCII, e os dados enviados são adicionados à própria URL. Dessa forma, existe limite de tamanho dos dados (relacionado ao limite de tamanho da URL), e os dados podem ser facilmente interceptados, além de ficarem gravados no histórico do navegador e em registros (logs) de servidores Web. Como um aspecto fundamental de segurança, não se deve utilizar o método ***GET*** em formulários que permitam o envio de dados sensíveis (senhas, por exemplo).

Para se obter uma segurança um pouco maior e considerando-se os formulários que devem permitir o envio de arquivos (vídeos, fotos, etc.), utiliza-se o método ***POST***. No método ***POST***, os dados são adicionados ao corpo da requisição, isto é, não são enviados como parte da URL. Nas Figuras 4.2 e 4.3 são ilustrados como os dados informados em um formulário qualquer são enviados utilizando-se os métodos ***GET*** e ***POST***, respectivamente.

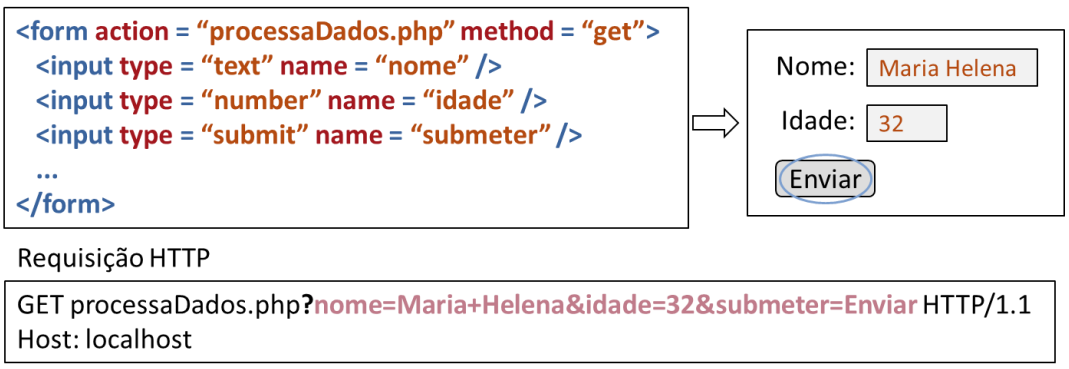


Figura 4.2 - Utilização do método GET na submissão de um formulário.

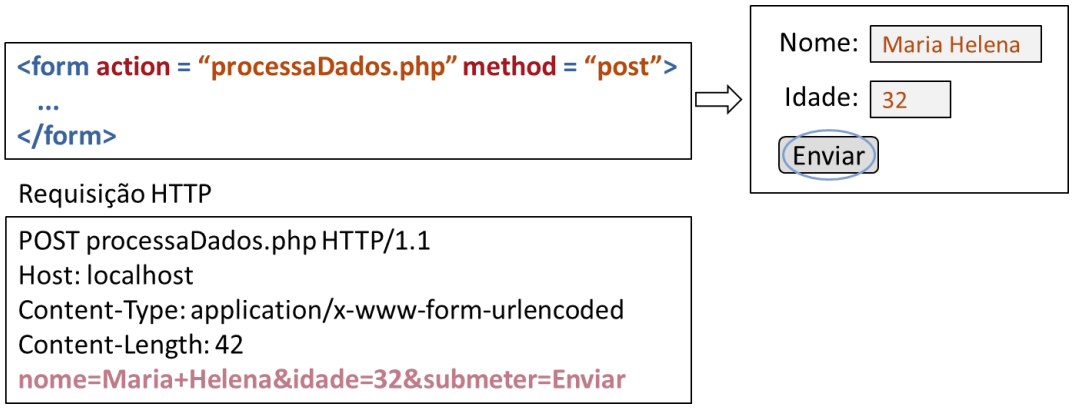


Figura 4.3 - Utilização do método POST na submissão de um formulário.

4.2. Elemento input

Em HTML, um formulário possui, no mínimo, um campo para entrada de dados e um botão para submissão. Existem diferentes tipos de elementos de formulário para a entrada de dados. Um dos elementos de controle mais utilizados é o elemento **input**. Este elemento é representado por uma tag vazia (<input />) e possui dois atributos obrigatórios: **name** e **type**. O atributo name permite definir um nome para um elemento de controle, e é por meio deste nome que o valor informado no campo será enviado e recuperado por uma aplicação. Em outras palavras, podemos pensar no valor do atributo name como o nome de uma variável que guardará o valor informado em um dado campo do formulário.

O elemento input é utilizado para representar vários tipos diferentes de elementos de controle em um formulário (texto, número, data, data e hora, mês, botões de radio, botão de submissão etc.). Dessa forma, é necessário especificar para o elemento input seu tipo, isto é, o tipo de campo que ele deve representar. Este é o objetivo do atributo type. Nas subseções seguintes, serão apresentados os principais tipos do elemento input.

4.2.1. Tipos text, password, submit e reset

No Exemplo 4.1 a seguir, é mostrado o código de um formulário composto pelos tipos ***text***, ***password***, ***submit*** e ***reset***. O tipo ***text*** (linha 5) permite a entrada de quaisquer caracteres (letras, dígitos e caracteres especiais). O tipo ***password*** (linha 9) é como o tipo ***text***, entretanto mascara os caracteres digitados (por exemplo, cada caractere é exibido como um ponto ou um asterisco). Assim, o tipo ***password*** é indicado para a digitação de senhas, por exemplo. O formulário ainda possui dois botões: um do tipo ***submit*** (linha 10), que permite submeter os dados informados (os dados serão enviados para o endereço especificado como valor para o atributo ***action*** da tag ***form***), e outro do tipo ***reset*** (linha 11)

para “resetar” o formulário, isto é, permite apagar todos os dados informados ou opções selecionadas no formulário.

```
1  <form action="processa.php" method="post">
2      <label for="login">
3          Login:
4      </label>
5      <input type="text" name="login" id="login" />
6      <label for="senha">
7          Senha:
8      </label>
9      <input type="password" name="senha" id="senha" />
10     <input type="submit" name="botao" value="Enviar" />
11     <input type="reset" name="botao" value="Apagar" />
12 </form>
```

Exemplo 4.1 - Exemplo de formulário composto por elementos input.

Conforme o exemplo mostra, os campos para a entrada do login (tipo ***text***) e da senha (tipo ***password***) possuem rótulos, e estes são especificados por meio do elemento ***label***. Este elemento permite especificar rótulos que sejam vinculados a diversos elementos de controle de formulário. Para se definir o vínculo entre um rótulo (tag ***label***) e um campo de formulário, deve-se atribuir uma identificação ao campo (atributo ***id***) e especificar o atributo for na tag ***label*** cujo valor seja igual à identificação atribuída ao campo (valor do atributo ***id***).

A especificação de rótulos não apenas auxilia os usuários no preenchimento de um formulário, mas permite que quaisquer ferramentas como leitores de tela, navegadores e outras identifiquem quais rótulos estão associados a quais campos. Por exemplo, a partir do vínculo entre um rótulo X e um campo Y, um leitor de tela poderá ler para uma pessoa com deficiência visual o conteúdo do rótulo X quando o campo Y receber foco. Pode-se verificar, portanto, que se trata de um mecanismo de acessibilidade importante em formulários na Web.

4.2.2. Tipos radio e checkbox

Para certos itens de informação a serem obtidos em um formulário, há um conjunto de valores predefinidos, e somente um destes valores deve ser escolhido pelo usuário. Por exemplo, em um formulário de atualização de dados de uma instituição acadêmica, considere um campo que solicite a situação de um dado estudante em um curso. Sabemos que existe um conjunto de situações possíveis (matriculado, trancado, cancelado etc.). Da mesma forma, em um formulário de cadastro, considere um campo que solicite o estado

civil de uma pessoa. De acordo com a lei brasileira, existem apenas 5 possibilidades: solteiro, casado, separado, divorciado e viúvo. Além disso, nestes dois casos, apenas um valor deve ser escolhido (por exemplo, a situação de um estudante no curso não pode ser “matriculado” e “trancado” ao mesmo tempo, e uma pessoa não pode estar casada e solteira ao mesmo tempo).

Para itens de informação cujo valor provém de um conjunto de valores predefinidos, uma possibilidade é utilizar botões de ***radio***. Botões de ***radio*** são especificados por meio do elemento input do tipo ***radio***. Cada botão representando um valor possível é especificado por meio de uma tag ***input***. No Exemplo 4.2 a seguir, é apresentado o código de um formulário que simula uma questão de múltipla escolha. Cada alternativa da questão é especificada por meio de uma tag ***input*** do tipo ***radio***. Observe que cada tag ***input*** possui um rótulo (tag ***label***), e apenas a primeira possui o atributo ***checked***, que permite deixar um botão marcado assim que o formulário é carregado.

Outro ponto fundamental é que os botões de ***radio*** possuem o mesmo nome, isto é, o mesmo valor para o atributo ***name***. Especificar o mesmo valor para o atributo ***name***, neste caso, é necessário para que o navegador e outras ferramentas identifiquem que as opções representadas pelos botões de ***radio*** pertencem ao mesmo grupo. Assim, a ação de marcar uma opção faz alguma outra (aquela que estiver marcada) ser desmarcada. Além disso, deve-se observar que cada tag ***input*** possui o atributo ***value***, necessário para especificar o valor de cada opção que será submetido ao servidor, caso a opção correspondente seja marcada.

Ainda no exemplo, foi utilizado o elemento ***fieldset*** composto pelo elemento ***legend***. O elemento ***fieldset*** deve ser utilizado para agrupar elementos de controle em um formulário. O elemento ***legend*** permite definir um título para o agrupamento. Neste exemplo, o elemento ***fieldset*** foi utilizado para fornecer um agrupamento semântico para todos os botões de ***radio***. Por meio deste agrupamento, foi possível, por exemplo, especificar o enunciado da pergunta de tal forma que qualquer ferramenta saiba que existe um relacionamento entre o enunciado (título do agrupamento) e as alternativas (botões de ***radio***).

```
<form action="processa.php" method="post">
    <fieldset>
        <legend>
            Para definir a fonte calibri da maneira mais adequada em CSS,
            utilizamos a declaração:
        </legend>
        <input type="radio" name="p1" value="a" id="a" checked />
        <label for="a">
            font-type: calibri;
        </label>
        <input type="radio" name="p1" value="b" id="b" />
        <label for="b">
            font-style: calibri;
        </label>
        <input type="radio" name="p1" value="c" id="c" />
        <label for="c">
            font-family: calibri;
        </label>
        <input type="radio" name="p1" value="d" id="d" />
        <label for="d">
            font-family: calibri, arial;
        </label>
        <input type="radio" name="p1" value="e" id="e" />
        <label for="e">
            font-family: calibri, sans-serif;
        </label>
    </fieldset>
    <input type="submit" name="botao" value="Enviar" />
</form>
```

Exemplo 4.2 - Exemplo de formulário composto por botões de radio.

Suponha que o formulário apresentado no Exemplo 4.2 deva ser estendido para incluir outra questão de múltipla escolha. Entretanto, nesta nova questão, pode haver mais de uma alternativa correta. Em outras palavras, novamente será apresentado um grupo de opções, mas o usuário poderá marcar mais de uma opção ao mesmo tempo. Neste tipo de situação, uma possibilidade é utilizar caixas de seleção múltipla, especificadas por meio do elemento input do tipo ***checkbox***. O Exemplo 4.3 ilustra como este elemento é utilizado. Observe que os mesmos atributos utilizados nos botões de ***radio*** são utilizados em caixas do tipo ***checkbox***.

```
<form action="processa.php" method="post">
    <fieldset>
        <legend>
            Quais dos atributos a seguir são específicos da HTML 5?
        </legend>
        <input type="checkbox" name="p2" value="a" id="p2_a" />
        <label for="p2_a">
            type
        </label>
        <input type="checkbox" name="p2" value="b" id="p2_b" />
        <label for="p2_b">
            maxlength
        </label>
        <input type="checkbox" name="p2" value="c" id="p2_c" />
        <label for="p2_c">
            required
        </label>
        <input type="checkbox" name="p2" value="d" id="p2_d" />
        <label for="p2_d">
            placeholder
        </label>
        <input type="checkbox" name="p2" value="e" id="p2_e" />
        <label for="p2_e">
            name
        </label>
    </fieldset>
    <input type="submit" name="botao" value="Enviar" />
</form>
```

Exemplo 4.3 - Exemplo de formulário composto por caixas de múltipla seleção.

4.2.3. Tipo file

Em muitos formulários HTML, é necessário permitir o envio de um ou mais arquivos. Por exemplo, um formulário de cadastro de pessoas poderia exigir o envio de uma foto de quem está sendo cadastrado; um formulário para inscrição em um processo seletivo poderia exigir o envio de vários documentos (RG, CPF, diploma etc.), e assim por diante. Em todos os casos em que um formulário deve permitir o envio (*upload*) de um ou mais arquivos, deve-se utilizar o elemento input do tipo ***file***. Este tipo define um campo composto por um botão e, quando o campo ou o botão é clicado, uma janela para seleção de arquivos é carregada pelo navegador. Por meio desta janela, o usuário pode selecionar um ou mais arquivos de seu computador ou qualquer outro dispositivo (por exemplo, um *pen-drive*) para serem enviados. O Exemplo 4.4 a seguir apresenta o código necessário para a implementação correta de um formulário que deva permitir o envio de arquivos.

Neste exemplo, o formulário permite o envio de um único arquivo ao mesmo tempo. Para permitir o envio de múltiplos arquivos utilizando-se o mesmo campo (tag ***input*** do tipo ***file***), deve-se especificar o atributo ***multiple*** na tag ***input***. Além disso, conforme o exemplo mostra, a tag ***input*** do tipo ***file*** possui o atributo ***accept***. Este atributo permite especificar o(s) tipo(s) de arquivos a serem selecionados e enviados. Trata-se apenas de um mecanismo que permite orientar os usuários sobre o tipo de arquivo que se espera na aplicação, e não de um mecanismo de validação. Em outras palavras, especificar o atributo ***accept*** não garante que o tipo definido será respeitado. No Exemplo 4.4, especificou-se como tipo esperado, o valor "***image/jpeg***", que significa que a aplicação espera que seja enviado um arquivo de imagem no formato JPEG (.jpg, .jpeg etc.). Apesar disso, conforme apresentado, o usuário ainda pode enviar arquivos em formatos diferentes, e é responsabilidade da aplicação (lado do servidor) conduzir a validação para se garantir que somente imagens no formato JPEG serão aceitas.

```
1  <form action="processa.php" method="post" enctype="multipart/form-data">
2      <label for="c_desc">
3          Descrição do arquivo:
4      </label>
5      <input type="text" name="desc" id="c_desc" required="required" />
6      <label for="c_arquivo">
7          Arquivo:
8      </label>
9      <input type="file" name="arquivo" id="c_arquivo" accept="image/jpeg" />
10     <input type="submit" name="botao" value="Enviar" />
11     <input type="reset" name="botao" value="Apagar" />
12 </form>
```

Exemplo 4.4 - Exemplo de formulário composto pelo elemento input do tipo file.

O atributo ***accept*** admite valores especificados de diferentes formas para o tipo do arquivo a ser enviado. A Tabela 4.1 a seguir mostra as possibilidades de valores para o atributo accept.

Tabela 4.1 - Relação de valores possíveis para o atributo accept do elemento file.

Valor	Descrição
Extensões de arquivos	Exemplos: .gif, .jpg, .pdf, .doc, ...
audio/*	Quaisquer arquivos de áudio
video/*	Quaisquer arquivos de vídeo
image/*	Quaisquer arquivos de imagem
Mime type	Exemplos: image/gif, audio/ogg, text/html, application/pdf, ...

Para que o envio de arquivos efetivamente funcione, o formulário deve satisfazer a duas condições: 1) o método de transmissão deve ser **POST**; e 2) a codificação dos dados deve ser ***multipart/form-data***, conforme é mostrado no Exemplo 4.4. Para especificar um tipo de codificação diferente do tipo *default* (***application/x-www-form-urlencoded***), deve-se utilizar o atributo ***enctype*** na tag ***form***. Portanto, o atributo ***enctype*** permite definir a maneira pela qual os dados serão codificados para serem enviados. As Figuras 4.1 e 4.2 já mostraram como os dados são codificados por meio do tipo *default*, isto é, quando não especificamos o atributo ***enctype***. Por exemplo, espaços são substituídos pelo caractere '+', e caracteres especiais são convertidos para valores hexadecimais em ASCII. Entretanto, em formulários com envio de arquivos, os dados destes arquivos (dados binários) não podem passar por nenhum tipo de conversão. Para garantir que os dados não serão codificados, utilizamos o atributo ***enctype*** com o valor "***multipart/form-data***".

4.2.4. Tipos novos da HTML 5

A linguagem HTML 5 representou um avanço significativo em relação à versão anterior da linguagem. Especialmente em formulários, vários elementos de controle foram criados. A seguir, são apresentados os principais elementos de controle de formulário introduzidos na HTML 5 (todos estes tipos são representados pelo elemento ***input***).

- Tipo ***number***: utilizado para campos numéricos. Permite definir o valor mínimo (atributo ***min***) e o valor máximo (atributo ***max***). Exemplo:
    - <input type="number" name="idade" min="0" max="120" ... />
    - Neste exemplo, o campo deve permitir apenas números entre 0 e 120.
- Tipo ***date***: utilizado para datas (dia, mês e ano). Também permite definir a data mínima e a data máxima (atributos ***min*** e ***max***). Exemplo:
    - <input type="date" name="data" min="1900-02-01" ... />
    - Neste exemplo, o campo deve permitir apenas datas iguais ou superiores a 01 de fevereiro de 1900. Embora fosse possível, não especificamos uma data máxima. Em ambos os casos (data mínima ou data máxima), o formato para validação deve ser ***aaaa-mm-dd***.
- Tipo ***time***: utilizado para instantes de tempo em horas e minutos, ou em horas, minutos e segundos. Formato para validação: ***hh:mm:ss***.

- Tipo ***datetime-local***: “junta” os tipos ***date*** e ***time*** em um único campo. Formato para validação: ***aaaa-mm-ddThh:mm:ss***.
- Tipo ***month***: utilizado para selecionar o mês de um ano. Formato para validação: ***aaaa-mm***.
- Tipo ***week***: utilizado para selecionar a semana de um ano. Formato para validação: ***aaaa-Wss***, onde ***ss*** é uma semana do ano (01 - 52).
- Tipo ***range***: utilizado para escolher um valor qualitativo dentro de um intervalo (o intervalo *default* é 0-100), isto é, um número cujo valor exato não é importante.
- Tipo ***email***: utilizado para endereços de e-mail. Efetua algumas validações básicas para o valor informado. O atributo ***multiple*** pode ser utilizado para que o usuário informe mais de um endereço de e-mail.

Uma observação importante é que os atributos ***min*** e ***max*** podem ser utilizados nos tipos ***time***, ***datetime-local***, ***month***, ***week*** e ***range*** (além dos tipos ***number*** e ***date***).

4.3. Elemento select

Suponha que estejamos elaborando um formulário HTML para cadastro de pessoas e um dos itens de informação seja a unidade federativa (estado) de onde a pessoa é. Sabemos que, no Brasil, existem 27 unidades federativas (26 estados e o Distrito Federal) e que alguém só pode ser de uma unidade federativa ao mesmo tempo. Dessa forma, a primeira abordagem que vem à mente para resolver este problema é utilizar botões de ***radio***, conforme foi visto. Entretanto, nesta abordagem, serão necessários 27 botões de ***radio*** (cada unidade federativa sendo representada por um botão de ***radio***). Especificar muitos botões de ***radio*** pode afetar negativamente a usabilidade do formulário e levar a um código mais difícil de manter.

```
<form action="processa.php" method="post">
    <label for="uf">
        Unidade federativa:
    </label>
    <select name="uf" id="uf">
        <optgroup label="Sudeste">
            <option value="SP">SP</option>
            <option value="RJ">RJ</option>
        </optgroup>
        <optgroup label="Nordeste">
            <option value="BA">BA</option>
        </optgroup>
        <optgroup label="Sul">
            <option value="SC">SC</option>
        </optgroup>
    </select>
    <input type="submit" name="botao" value="Enviar" />
</form>
```

Exemplo 4.5 - Exemplo de formulário composto pelo elemento select.

Assim, uma estratégia melhor em situações deste tipo é utilizar o elemento ***select***, que representa uma lista de seleção do tipo *drop-down*. As opções da lista apenas são exibidas se o usuário clicar sobre a lista. Cada opção é especificada por meio do elemento ***option*** (tags ***<option>*** e ***</option>***), que é filho do elemento ***select***. O Exemplo 4.5 apresenta o código de um formulário que possui o elemento ***select***.

Conforme o exemplo mostra, ao utilizarmos o elemento ***select***, é possível agrupar as opções conforme algum critério. Esta estratégia pode ajudar o usuário a encontrar a opção desejada mais facilmente especialmente quando há muitas opções envolvidas. Para agrupar as opções, deve-se utilizar o elemento ***optgroup***, conforme é mostrado no Exemplo 4.5. Cada agrupamento deve possuir um título, especificado por meio do atributo ***label***. Para deixar uma opção selecionada por *default*, utilizamos o atributo ***selected*** na tag ***option*** referente à opção *default*. Outro ponto fundamental do elemento ***select*** é que ele pode ser utilizado para representar uma lista de seleção múltipla, isto é, que permita que mais de uma opção seja selecionada ao mesmo tempo. Para alcançar este objetivo, utilizamos o atributo ***multiple***.

4.4. Elemento textarea

O elemento ***textarea*** é particularmente útil para os itens de informação cujo valor de entrada é um texto longo. Por exemplo, campo para entrada de um código, campo para

envio de observações, campo para envio de reclamações etc. O elemento ***textarea*** representa uma área de texto que possui várias linhas, diferentemente do elemento ***input*** do tipo ***text***, que é adequado para textos menores (de uma única linha). O Exemplo 4.6 a seguir ilustra o código para a especificação do elemento ***textarea*** em um formulário.

```
1  <form action="processa.php" method="post">
2      <label for="nome">Nome:</label>
3      <input type="text" name="nome" id="nome" />
4      <label for="obs">Observações:</label>
5      <textarea name="obs" id="obs">
6          Texto default que será exibido na área
7      </textarea>
8  </form>
```

Exemplo 4.6 - Exemplo de formulário composto pelo elemento textarea.

Observe que o elemento ***textarea*** é especificado por meio das tags ***<textarea>*** e ***</textarea>*** (linhas 5 e 7). Como qualquer outro campo de formulário, o elemento ***textarea*** inclui o atributo ***name*** (e o atributo ***id*** para o vínculo entre o campo e o rótulo).

4.5. Processamento dos dados em PHP

Os dados do formulário enviados pelo navegador ao servidor são disponibilizados a um script em PHP por meio de *arrays* superglobais. O *array* a ser utilizado depende do método escolhido para envio (***GET*** ou ***POST***). Dados enviados pelo método ***GET*** são armazenados no *array* ***$_GET*** e dados enviados pelo método ***POST*** são armazenados no *array* ***$_POST***.

Assim, o programa em PHP deve acessar o *array* apropriado, fornecendo o nome dos campos do formulário como chave associativa. Em outras palavras, em vez de acessarmos os elementos do *array* da forma tradicional, utilizando índices numéricos, *strings* serão utilizadas como índices. O Exemplo 4.7 a seguir ilustra o código de um script em PHP necessário para acessar e manipular os dados de campos cujo nome (atributo ***name***) são ***"nome"*** e ***"idade"***. No caso do envio de arquivos, para que um script em PHP acesse e manipule os dados dos arquivos, é necessário utilizar o *array* superglobal ***$_FILES***.

```php
<?php
    $nome = $_POST["nome"];
    $idade = $_POST["idade"];
    echo "Nome recebido: $nome<br />";
    echo "Idade recebida: $idade<br />";
?>
```

Exemplo 4.7 - Exemplo de um script em PHP recebendo dados de um formulário.

4.6. Exercícios propostos

Exercício 1. Crie um formulário de cadastro de usuários com os seguintes campos:

a. Nome (valor obrigatório).

b. Senha (valor obrigatório e não pode haver senhas com mais de 10 caracteres).

c. Vídeo e foto (apenas a foto é obrigatória).

d. Interesses (Esporte, Política e Economia). O usuário pode selecionar mais de um interesse.

e. Deseja receber notícias pelo e-mail? (Sim ou Não).

f. O formulário deve possuir dois botões: um para submeter os dados e outro para fazer o formulário voltar ao seu estado original.

Exercício 2. Crie um formulário que permita aos clientes de uma concessionária submeterem os seguintes dados sobre os veículos adquiridos:

a. Marca. Valor obrigatório de, no máximo, 10 caracteres.

b. Combustível. As opções possíveis são álcool e gasolina. Os dois valores devem ser escolhidos caso o veículo seja bicombustível.

c. Motor. Os valores possíveis são 1.0, 1.6 e 2.0.

d. Câmbio. Os valores possíveis são manual e automático.

e. Descrição. Texto longo não obrigatório.

f. Data de fabricação do veículo. A data deve ser um valor superior ou igual a 01/01/1990.

g. Data e hora de aquisição do veículo. A data deve ser um valor entre 01/01/2000 e 01/01/2018 (qualquer hora).

h. Semana do ano em que o veículo foi adquirido. O ano deve ser igual ou superior a 2000 (qualquer semana).

i. Preço unitário. O preço unitário deve ser um número positivo. Permita até duas casas decimais de precisão.

j. Endereço de e-mail do proprietário do veículo. Mais de um endereço pode ser informado.

k. Não utilize o mesmo tipo de elemento de controle por mais de uma vez no formulário.

Capítulo 5. Estruturas de Controle

Neste capítulo, são apresentadas as estruturas de controle que são utilizadas para realizar decisões lógicas, testar se determinada expressão é verdadeira, repetir um bloco de comandos por determinado número de vezes ou até para que uma condição seja atingida.

5.1. Estruturas de Controle Condicional

Até este ponto, vimos programas em PHP que são apenas sequências simples de comandos. Tais programas possuem um único fluxo de execução, assim como um automóvel que trafega em uma via sem desvios ou retornos. O Exemplo 5.1. ilustra um código com estrutura sequencial.

```php
<?php
  echo "Bem vindo ao meu programa!";
  echo "Abastecendo...";
  echo "Volte sempre!";
?>
```

Exemplo 5.1 – Código utilizando estrutura sequencial.

Na Figura 5.1 é apresentado analogamente um automóvel que trafega em uma via sem desvios ou retornos, assim com um programa sequencial.

Figura 5.1 – Programa com um único fluxo de execução.

Em geral, programas computacionais envolvem diversas estruturas que permitem acionar fluxos de execução diferentes, caso certas condições sejam ou não atendidas. De forma análoga, é como se um automóvel estivesse agora trafegando pelas ruas de uma

cidade e, dependendo das condições e necessidades, diferentes caminhos pudessem ser tomados.

Os pontos de decisão de um programa são semelhantes a interseções de vias, que podem levar a caminhos diferentes. Assim, o mesmo programa poderia ter diversos caminhos, mas somente são percorridos aqueles que atendem às condições verificadas nos pontos de decisão. Vários pontos de decisão em um programa são ilustrados na Figura 5.2.

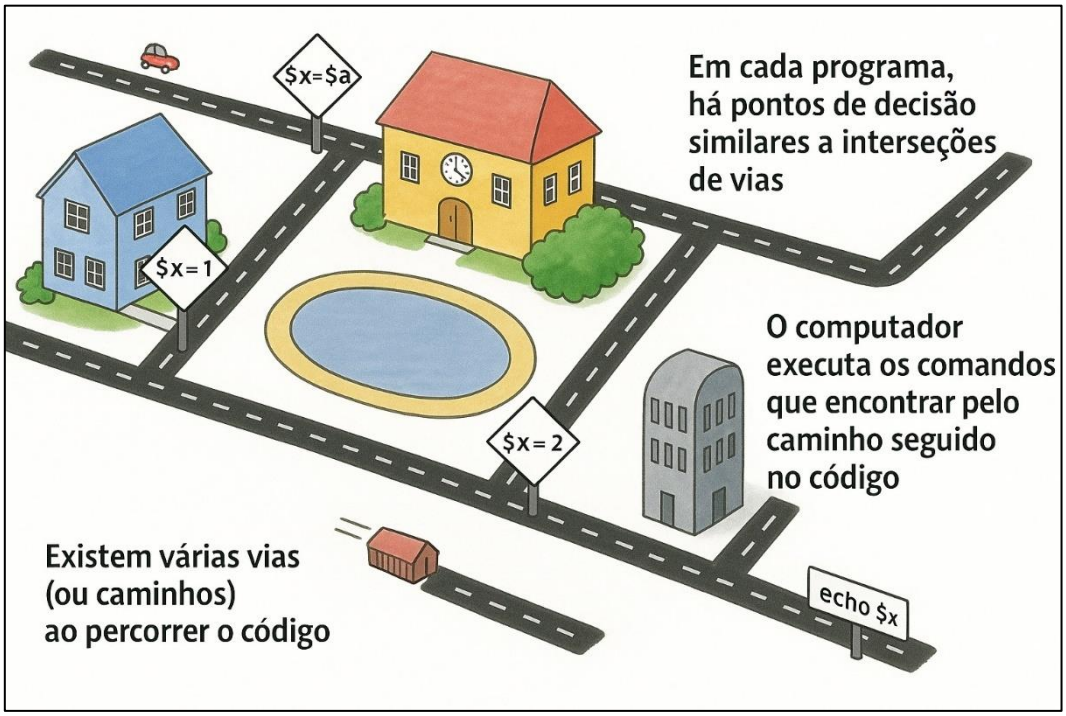


Figura 5.2 – Um programa pode ter diversos caminhos possíveis.

Para permitir o acionamento de diferentes caminhos ou fluxos de execução nos programas em PHP, serão vistas, inicialmente, as estruturas de controle condicional, representadas pelos comandos ***if*** e ***switch***.

5.1.1. Comando if

Este comando é formado pela palavra reservada ***if***, que em português significa “se”, seguida de uma expressão entre parênteses. Quando o comando é executado, a expressão é avaliada e, caso ela seja verdadeira (***true***), o bloco de comandos que estiver entre chaves será executado. Se a expressão entre parênteses for falsa (***false***), o bloco de comandos dentro das chaves não será executado.

```
if ( expressão ) {
   //instruções...
}
```

O Exemplo 5.2 mostra um programa em PHP que verifica se um determinado número é par. Nele é utilizado um comando ***if*** simples.

```php
<?php
  $num = 4;
  if($num % 2 == 0){
    echo "Número é par!";
  }
?>
```

Exemplo 5.2 – Exemplo do comando if simples.

O comando ***if*** pode ser utilizado também de forma composta, com o comando ***else*** (em português, “senão”) na sequência para especificar o bloco de comandos que será executado caso a expressão seja avaliada como falsa.

```
if ( expressão ) {
  //instruções...
  } else {
  //instruções…
}
```

Se o bloco de comandos possuir somente um comando, o uso das chaves torna-se opcional. Considera-se, contudo, uma boa prática de programação o uso das chaves mesmo nesta situação. No Exemplo 5.3, o comando ***if*** testa se um determinado número é par. Caso o teste seja avaliado como negativo, o bloco de comandos dentro do ***else*** é executado.

```php
<?php
  $num = 7;
  if($num % 2 == 0){
    echo "Número é par!";
  }else{
    echo "Número é ímpar!";
  }
?>
```

Exemplo 5.3 – Exemplo do comando if composto.

Analogamente, um programa executar um comando condicional é como um automóvel que trafega por uma via e se depara com uma bifurcação. Neste momento, baseado na avaliação feita pelo condutor (no caso do programa, uma condição), toma-se uma das direções, conforme a Figura 5.3.

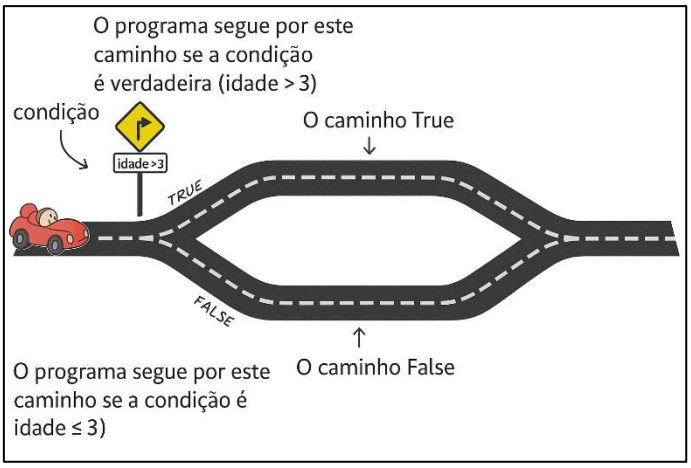


Figura 5.3: A avaliação feita pelo programa determina qual direção seguir.

Continuando a analogia, é possível ainda que, tomada umas das direções, mais a frente se encontre outra bifurcação. Neste caso, deve-se fazer uma nova avaliação para determinar o caminho a seguir.

Considere o código do exemplo 5.4. Caso o valor da variável **$salario** seja maior que 1200, a execução do programa segue para o bloco de comandos dentro do ***else***. Ali há outro comando condicional que avalia novamente a variável **$salario*.*** Em situações como esta, onde há estruturas condicionais dentro de estruturas condicionais, dizemos que há estruturas condicionais aninhadas.

```
1
2  <?php
3    $salario = $_POST["salario"];
4
5    if ($salario <= 1200){
6      $aumento = $salario * 0.2;
7    } else {
8      if ($salario <= 1500) {
9        $aumento = $salario * 0.3;
10     } else {
11       $aumento = $salario * 0.1;
12     }
13   }
14
15   $novoSalario = $salario + $aumento;
16   echo "Valor do Aumento: R$ " . $aumento;
17   echo "<br/><br/>Novo Salario: R$" . $novoSalario;
18 ?>
19
```

Exemplo 5.4 – Estruturas condicionais aninhadas.

As estruturas condicionais aninhadas são ilustradas na Figura 5.4, utilizando a analogia com as vias de trânsito. Caso a primeira condição avaliada seja falsa, ainda deve-se avaliar outras condições.

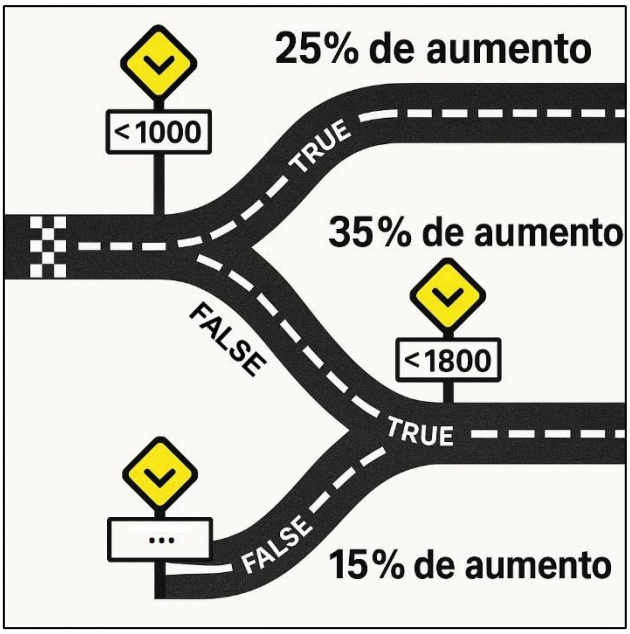


Figura 5.4. Analogia com vias de trânsito e estruturas condicionais aninhadas.

Para resolver exemplos como esse, pode-se utilizar o comando ***elseif***. Este comando é apropriado para os casos em que há múltiplas avaliações de condições e onde cada condição leva a um fluxo de execução diferente. Reescrevendo o Exemplo 5.4, tem-se o código do Exemplo 5.5.

```
1
2  <?php
3    $salario = $_POST["salario"];
4
5    if ($salario <= 1200){
6      $aumento = $salario * 0.2;
7    } elseif ($salario <= 1500) {
8      $aumento = $salario * 0.3;
9    } else {
10     $aumento = $salario * 0.1;
11   }
12
13   $novoSalario = $salario + $aumento;
14   echo "Valor do Aumento: R$ " . $aumento;
15   echo "<br/><br/>Novo Salario: R$" . $novoSalario;
16 ?>
17
```

Exemplo 5.5 – Estruturas condicionais aninhadas com o comando elseif.

O comando ***else*** ao final permite contemplar todas as demais possibilidades que as condições anteriores não contemplaram.

5.1.2. Comando switch

O comando ***switch*** é parecido com o comando ***if***, uma vez que ambos avaliam o valor de um argumento teste para escolher qual bloco de instruções deve ser executado. O argumento pode ser numérico, um caractere ou uma *string*.

O valor do argumento teste é avaliado e se for igual ao valor de uma das constantes, a execução do código é desviada para aquele ponto.

```
switch ( expressão ) {
  case constante1 :
    //instruções...
  break ;
  case constante2 :
    //instruções...
  break ;
  case constante3 :
    //instruções...
  break ;
  default :
    //instruções...
}
```

Enquanto o ***if*** utiliza várias cláusulas (***if***, ***else***, ***elseif***), a estrutura ***switch..case*** utiliza somente uma cláusula (***case***), tornando o código um pouco mais organizado. O bloco ***default*** é opcional e executado se nenhuma combinação for encontrada.

O Exemplo 5.6 mostra um programa em PHP que verifica se um determinado número é igual a 0, 1, 2, 3 ou 4. Se o número fornecido for diferente destes, o código exibe a mensagem “é cinco ou mais”.

Note que após cada bloco de instruções deve ser utilizado o comando ***break***, para que o comando ***switch*** seja encerrado e a execução do restante do código continue após ele.

```php
<?php
  $i = $_POST["numero"];
  switch ($i){
    case 0:
      echo "$i é zero";
      break;
    case 1:
      echo "$i é um";
      break;
    case 2:
      echo "$i é dois";
      break;
    case 3:
      echo "$i é três";
      break;
    case 4:
      echo "$i é quatro";
      break;
    default:
      echo "$i é cinco ou mais";
  }
?>
```

Exemplo 5.6 – Exemplo da estrutura switch..case com entrada de valor numérico.

```php

<?php
  $mes = date("F", time());

  switch ($mes){
    case "January":
      echo "Janeiro";
      break;
    case "February":
      echo "Fevereiro";
      break;
    case "March":
      echo "Março";
      break;
    case "April":
      echo "Abril";
      break;
    default:
      echo "Outro mês";
  }
?>
```

Exemplo 5.7 – Exemplo da estrutura switch..case com entrada de valor string.

No exemplo 5.10, o valor do argumento teste é o conteúdo da variável **$mes** (meses do ano em inglês). As saídas possíveis são ***"janeiro"***, ***"fevereiro"*** ou ***"março"***, ou ainda ***"outro mês"***, se o valor fornecido for diferente destes.

5.1.3 Exercícios propostos

Exercício 1. Faça uma página HTML para enviar três notas de um aluno para um programa PHP, que recupera as notas, calcula e retorna uma nova página HTML com a média aritmética das notas do aluno e a mensagem constante na tabela a seguir:

Média aritmética	Mensagem
Média < 4,0	Reprovado
4,0 >= Média < 6,0	Reavaliação
Média >= 6,0	Aprovado

Exercício 2. Faça uma página HTML para enviar três números inteiros para um programa PHP, que recupera os números e retorna-os uma nova página HTML em ordem crescente. Suponha que o usuário sempre digitará três números diferentes.

Exercício 3. Elabore um programa PHP que calcule o valor a ser pago por uma compra, considerando o preço normal de etiqueta e a escolha da condição de pagamento (enviados por meio de um formulário HTML), e de acordo com os seguintes critérios:

Código	Condição de pagamento
1	À vista em dinheiro ou cheque, recebe 10% de desconto.
2	À vista no cartão de crédito, recebe 5% de desconto.
3	Em 2 vezes, preço normal da etiqueta sem juros.
4	Em 3 vezes, preço normal da etiqueta mais juros de 10%.

Exercício 4. Elaborar um programa em PHP que recebe de uma página web 3 valores inteiros e positivos A, B, C e verifica se eles formam ou não um triângulo. Caso os valores formem um triângulo, deve-se calcular e imprimir o valor do perímetro do triângulo, indicando com uma mensagem se este é equilátero (três lados iguais), isósceles (apenas 2 lados iguais) ou escaleno (todos os lados são diferentes). Se os valores não formam um triângulo, escrever uma mensagem informando o fato. Lembre-se que em um triângulo, o comprimento de cada lado deve ser menor que a soma dos outros dois lados.

Exercício 5. Escreva um programa em PHP que recebe de uma página HTML as coordenadas (X, Y) de um ponto no sistema cartesiano e escrever o quadrante ao qual o

ponto pertence. Caso o ponto não pertença a nenhum quadrante, escrever se ele está sobre o eixo X, eixo Y ou na origem.

Exercício 6. Uma empresa concederá um aumento de salário aos seus funcionários, variável de acordo com o cargo, conforme a tabela abaixo. Faça um programa em PHP que recebe de uma página web o salário e o cargo de um funcionário, e calcule o novo salário. Se o cargo do funcionário não estiver na tabela, ele deverá, então, receber 40% de aumento. Mostre o salário antigo, o novo salário e a diferença entre o salário antigo e o novo salário.

Cargo	**Percentual**
Gerente	10%
Engenheiro	20%
Técnico	30%

Exercício 7. O IMC (Índice de Massa Corporal) é um critério da Organização Mundial de Saúde para dar uma indicação sobre a condição de peso de uma pessoa adulta. A fórmula é IMC = peso / (altura)^2. Elabore um programa em PHP que receba o peso e a altura de um adulto e mostre sua condição de acordo com a tabela abaixo.

IMC em adultos	**Condição**
Abaixo de 18,5	Abaixodo peso
Entre 18,5 e 25	Peso normal
Entre 25 e 30	Acima do peso
Acima de 30	Obeso

Exercício 8. Fazer um programa na Linguagem PHP para receber a idade de uma pessoa e informar sua classe eleitoral, que pode ser:

- Não eleitor (abaixo de 16 anos);
- Eleitor obrigatório (entre 18 e 65 anos);
- Eleitor facultativo (entre 16 e 18 anos e maior de 65).

Exercício 9. Faça uma página HTML para enviar número inteiro entre 1 e 7 para um programa PHP, que deverá escreva o dia da semana correspondente. Caso o usuário digite um número fora desse intervalo, deverá aparecer uma mensagem informando que não existe dia da semana com esse número.

Exercício 10. Criar um programa em PHP que informe a quantidade total de calorias de uma refeição a partir do usuário, que deverá informar o prato, a sobremesa e a bebida por meio de uma página HTML. Veja a tabela de calorias a seguir:

Prato	Calorias	Sobremesa	Calorias	Bebida	Calorias
Vegetariano	180 cal	Abacaxi	75 cal	Chá	20 cal
Peixe	230 cal	Sorvete *diet*	110 cal	Suco de laranja	70 cal
Frango	250 cal	Mouse *diet*	170 cal	Suco de melão	100 cal
Carne	350 cal	Mouse chocolate	200 cal	Refrigerante *diet*	65 cal

Exercício 11. O governo federal abriu uma linha de crédito para os servidores federais. O valor máximo da prestação não poderá ultrapassar 30% do salário bruto. Fazer um programa em PHP que receba, por meio de uma página HTML, o salário bruto e o valor da prestação, e informar se o empréstimo pode ou não ser concedido.

Exercício 12 (**Desafio**). Criar uma página HTML que leia o número correspondente ao mês atual e os dígitos (somente os quatro números) de uma placa de veículo, e através do número finalizador da placa (algarismo da casa das unidades) determine se o IPVA do veículo vence no mês corrente, de acordo com a tabela abaixo.

Final 1 – mês (1) – Janeiro	Final 6 – mês (6) – Junho
Final 2 – mês (2) – Fevereiro	Final 7 – mês (7) – Julho
Final 3 – mês (3) – Março	Final 8 – mês (8) – Agosto
Final 4 – mês (4) – Abril	Final 9 – mês (9) – Setembro
Final 5 – mês (5) – Maio	Final 0 – mês (10) – Outubro

Exercício 13 (**Desafio**). Criar um programa em PHP que, a partir da idade e peso do paciente recebido por meio de uma página HTML, calcule a dosagem de determinado medicamento e imprima a receita informando quantas gotas do medicamento o paciente deve tomar por dose. Considere que o medicamento em questão possui 500 mg por ml, e que cada ml corresponde a 20 gotas.

- Adultos ou adolescentes desde 12 anos, inclusive, se tiverem peso igual ou acima de 60 quilos devem tomar 1000 mg; com peso abaixo de 60 quilos devem tomar 875 mg.
- Para crianças e adolescentes abaixo de 12 anos a dosagem é calculada pelo peso corpóreo conforme a tabela a seguir:

Peso	Dosagem
5 kg a 9 kg	125 mg
9.1 kg a 16 kg	250 mg
16.1 kg a 24 kg	375 mg
24.1 kg a 30 kg	500 mg
Acima de 30 kg	750 mg

5.2. Estruturas de Controle de Repetição

Os comandos de repetição são utilizados para repetir a execução um conjunto de instruções por um número determinado de vezes ou até que uma condição seja atingida. Em termos práticos, isto significa repetir comandos até que uma variável atinja determinado valor ou que seja diferente de um valor.

A linguagem PHP define vários comandos de repetição. Veremos os comandos ***for*****,** ***while*** e ***do..while.*** O comando ***foreach*** é utilizado para percorre *arrays* e será estudado mais adiante.

5.2.1 Comando for

O comando ***for*** é utilizado para repetir a execução de um conjunto de instruções por um número determinado de vezes. Existem três parâmetros para o comando ***for***: inicialização, condição de parada e operação de incremento ou decremento. A sintaxe do comando ***for*** é:

```
for (inicialização ; condição_de_parada ; in (de) crement o){
  // instruções
}
```

O parâmetro de ***inicialização*** define o valor inicial da variável que controlará a repetição das instruções. Por exemplo, o parâmetro de inicialização **$contador = 0** define o valor inicial da variável **$contador** que será utilizado para controlar a repetição. O comando da inicialização é executado uma única vez, ao iniciar o laço for. Em seguida, a condição de parada é avaliada. O parâmetro ***condição_de_parada*** define a condição para que a repetição continue executando.

Por exemplo, o parâmetro de condição de parada **$contador < 10** define que o conjunto de instruções deverá ser repetido enquanto esta condição seja verdadeira. Quando esta condição for falsa, a repetição será encerrada. Após a execução das instruções dentro do laço for, o comando de ***in(de)cremento*** é executado. No exemplo

abaixo esse comando é o **$contador++**. Após o incremento, a condição de parada é avaliada novamente. Caso a condição seja verdadeira, as instruções são executadas novamente e, caso a condição seja falsa, a repetição é encerrada, conforme o Exemplo 5.8.

```php
<?php
  for($contador = 0; $contador < 10; $contador++){
    echo "$contador ";
  }
?>
```

Exemplo 5.8 – Exemplo do comando for.

O Exemplo 5.9 contém um programa que efetua a soma dos números inteiros do intervalo de 1 a 10. Nele, a variável **$soma** é responsável por acumular as somas dos números a cada repetição. Para isso, deve-se inicializar a variável **$soma** antes do laço, de forma que ela tenha um valor válido quando for utilizada pela primeira vez.

```php
<?php
  $soma = 0;
  for($i = 1; $i <= 10; $i++){
    $soma += $i; // $soma = $soma + $i;
  }
  echo "A soma dos números inteiros do intervalo de 1 a 10 é <b>$soma</b>.";
?>
```

Exemplo 5.9 – Exemplo do comando for usando uma variável acumuladora ($soma).

5.2.2 Comando while

Neste tipo de estrutura de repetição, não se conhece previamente o número de repetições que serão executadas. Também são chamadas de estruturas de repetição condicionais pelo fato de encerrarem sua execução mediante uma determinada condição.

O número de repetições está ligado a uma condição sujeita à modificação pelas instruções do interior do laço. A sintaxe do comando ***while*** é:

```
// inicialização
while ( condição_de_parada ){
  // instruções
  // in(de)cremento
}
```

Vale ressaltar que, ao contrário do comando ***for***, na instrução ***while*** deve-se fazer o gerenciamento do incremento. No Exemplo 5.10, é descrito um programa que exibe a soma dos números inteiros do intervalo de 1 a 10.

```php
<?php
  $soma = 0;
  $i = 1;
  while($i <= 10){
    $soma += $i; // $soma = $soma + $i;
    $i++;
  }
  echo "A soma dos números inteiros do intervalo de 1 a 10 é <b>$soma</b>.";
?>
```

Exemplo 5.10 – Exemplo do comando while.

A sequência de comandos será repetida enquanto a condição for verdadeira (**$i <= 10**). Se a condição não for mais verdadeira, a repetição é interrompida e a sequência de comandos, que estiver logo após o **}** da estrutura, passa a ser executada.

5.2.3 Comando do..while

A estrutura do comando ***do..while*** é muito parecida com a estrutura ***while***. A diferença está no fato de que o bloco de instruções é executado no mínimo uma vez. A sintaxe do comando é:

```
do {
  // instruções
  // in(de)cremento
} while ( condição_de_parada );
```

No Exemplo 5.11, é descrito um programa que exibe a soma dos números inteiros do intervalo de 1 a 10.

```php
<?php
  $soma = 0;
  $i = 1;
  do{
    $soma += $i; // $soma = $soma + $i;
    $i++;
  }while($i <= 10);
  echo "A soma dos números inteiros do intervalo de 1 a 10 é <b>$soma</b>.";
?>
```

Exemplo 5.11 – Exemplo do comando do..while.

Note que o ***do..while*** executa o bloco de comandos e depois testa a condição (**$i <= 10**) no final da estrutura. Assim como na estrutura ***while***, deve-se fazer o gerenciamento do incremento (**$i++;**).

5.2.4 Comando foreach

A estrutura ***foreach*** oferece uma maneira mais conveniente de percorrer os elementos de um *array*. A sintaxe do comando é:

```
foreach ( $ nome_do_vetor as $ elemento ){
  // instruções
}
```

O vetor **$nome_do_vetor** é percorrido do primeiro ao último índice e a cada iteração o valor do elemento corrente do vetor é atribuído à variável **$elemento** e o ponteiro interno do *array* é avançado.

5.3. Juntando tudo!

Até aqui, foram apresentados diversos conceitos básicos para a construção de uma página Web dinâmica utilizando a linguagem PHP. O exemplo descrito nessa seção visa juntar os conteúdos já vistos e aplicá-los em um caso prático.

Considere um sistema que permita o cálculo e comparação de vários orçamentos de três produtos (*smartphone*, *notebook* e *smart tv*) em várias lojas. Em seguida, automaticamente, o sistema informa se o valor da compra está dentro do orçamento (R$ 10.000,00) e também o melhor orçamento, o valor mais baixo. O número de lojas deve ser informado ao sistema e este deve fornecer um formulário que permita inserir os preços dos três produtos (*smartphone*, *notebook* e *smart tv*). As informações são o nome da loja e o valor de cada produto. O sistema deve calcular total da compra de cada loja, que consiste na soma do valor dos três produtos.

Caso o total da compra da loja for maior que dez mil reais, ele está acima do orçamento. Se o total da compra for menor que três mil reais, ele estará dentro abaixo do orçamento. Neste caso, o nome da loja e o valor do orçamento serão armazenados e exibidos posteriormente. Se o total da compra estiver entre três e dez mil reais (inclusive), ele estará dentro do orçamento. Ao final da execução, o sistema deve apresentar uma página Web com uma tabela, cujas linhas mostram o nome de cada loja, o nome e valor de cada produto, o orçamento de cada loja, além da loja mais barateira.

A página ***index.html***, Exemplo 5.12, pede ao usuário do sistema a quantidade de lojas, cujas valores dos produtos serão inseridos.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
      <title>Orçamentos - Página Inicial</title>
  </head>
  <body>
    <form action="form_prod.php" method="post">
      <fieldset>
        <legend>Digite a quantidade de Orçamentos:</legend>
        <input type="number" name="quantidadeOrca" min="1" step="1" />
        <input type="submit" value="Enviar" />
      </fieldset>
    </form>
  </body>
</html>
```

Exemplo 5.12 – Página index.html.

A página ***form_prod.php***, Figura 5.13, recebe a quantidade de lojas e gera o formulário com os campos apropriados para inserir as informações de nome e valores dos produtos. É importante ressaltar aqui, a estratégia para gerar os valores dos nomes das tags ***input.*** Para que haja campos com nomes únicos para cada loja, utiliza-se a variável **$i** concatenada ao valor do atributo ***name*** das tags.

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Orçamentos - Formulário de Produtos</title>
  </head>
  <body>
    <form action="processa_prod.php" method="post">
      <?php
        $quantidadeOrca = $_POST["quantidadeOrca"];
        for($i = 1; $i <= $quantidadeOrca; $i++){
          echo '<fieldset>
            <legend>Dados da Loja ' . $i . ':</legend>
            <p>
              <label>Nome da loja:</label>
              <input type="text" name="nome_loja' . $i . '" />
            </p>
            <p>
              <label>Smartfone:</label>
              <input type="text" name="smartfone' . $i . '" min="0" max="10000" />
              <label>Notebook:</label>
              <input type="text" name="notebook' . $i . '" min="0" max="10000" />
              <label>Smartv:</label>
              <input type="text" name="smartv' . $i . '" min="0" max="10000" />
            </p>
            </fieldset>';
          }
          echo '<input type="hidden" name="quantidadeOrca" value="'. $quantidadeOrca .'" />';
      ?>
      <input type="submit" value="Enviar" />
      <input type="reset" value="Limpar" />
    </form>
  </body>
</html>
```

Exemplo 5.13 – Página form_prod.php.

No Exemplo 5.14, a página ***processa_prod.php*** é responsável por receber os valores, calcular os orçamentos e exibir a situação final de cada loja. As informações de cada loja são recebidas, utilizando um mecanismo semelhante ao da página anterior. Aqui, contudo, a *string* usada como índice associativo da superglobal (**$_GET** ou **$_POST**) é interpolada com a variável **$i**, obtendo-se assim, os nomes dos campos referentes às lojas são enviados por meio do formulário.

Após o cálculo dos totais e a definição da situação da loja, uma linha da tabela é gerada para cada loja com suas informações.

```php
<!DOCTYPE html>
<html>
    <head>
      <meta charset="utf-8" />
      <title>Orçamentos - Resultado</title>
    </head>
    <body>
      <table border="1">
        <thead>
          <tr>
            <th>Nome da loja</th>
            <th>Smartfone</th>
            <th>Notebook</th>
            <th>Smartv</th>
            <th>Total</th>
            <th>Situação</th>
          </tr>
        </thead>
        <tbody>
          <?php
            $quantidadeOrca = $_POST["quantidadeOrca"];
            $totalCompra = 0;
            $menor = 99999;
            $lojaBarateira = '';
            for($i = 1; $i <= $quantidadeOrca; $i++){
              $nome_loja = $_POST["nome_loja$i"];
              $smartfone = $_POST["smartfone$i"];
              $notebook = $_POST["notebook$i"];
              $smartv = $_POST["smartv$i"];
              $total = $smartfone + $notebook + $smartv;
              if($total >= 10000){
                $situacao = "Acima do orçamento";
              }elseif($total <= 3000){
                $situacao = "Abaixo do orçamento";
              }else{
                $situacao = "Dentro do orçamento";
              }
              $totalCompra += $total;
              if ($total < $menor) {
                $menor=$total;
                $lojaBarateira=$nome_loja;
              }
              echo '<tr>
                  <td>' . $nome_loja . '</td>
                  <td>' . number_format($smartfone, 2) . '</td>
                  <td>' . number_format($notebook, 2) . '</td>
                  <td>' . number_format($smartv, 2) . '</td>
                  <td>' . number_format($total, 2) . '</td>
                  <td>' . $situacao . '</td>
                </tr>';
            }
          ?>
        </tbody>
        <tfoot>
          <tr>
            <td colspan="5">Melhor Orçamento</td>
            <td>
              <?php
                echo "Loja: ".$lojaBarateira;
                echo "<br>Orçamento:".number_format($menor, 2);
              ?>
            </td>
          </tr>
        </tfoot>
      </table>
    </body>
</html>
```

Exemplo 5.14 – Página processa_prod.php.

5.4. Exercícios propostos

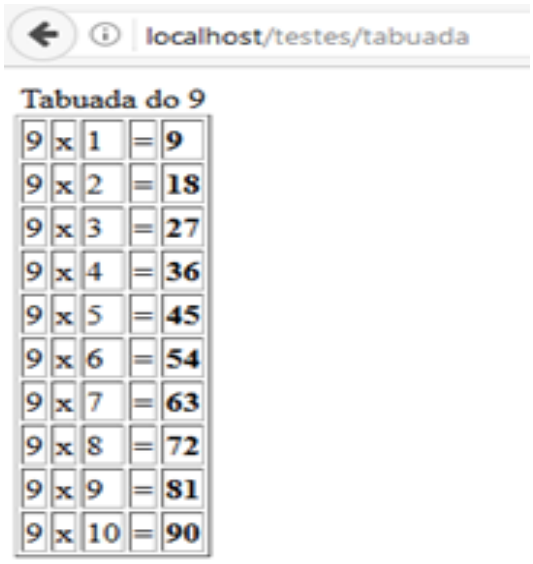
Exercício 1. Faça uma página HTML para enviar dois números inteiros quaisquer para um programa PHP, que deverá calcular e escrever a multiplicação entre os números recebidos utilizando para isso apenas o operador “+”, por exemplo:

(3 * 5) = 5 + 5 + 5

(4 * 12) = 12 + 12 + 12 + 12

Exercício 2. Dado que, um número é primo quando é divisível apenas por 1 e por ele mesmo, faça um programa em PHP que receba, por meio de uma página HTML, um número inteiro maior que 1, verifique se o número fornecido é primo ou não e mostre uma mensagem de “número primo” ou de “número não primo”.

Exercício 3. Faça um programa em PHP que receba, por meio de uma página HTML, um número, calcule e mostre a tabela (gerar uma tabela do HTML) deste número, conforme figura abaixo:



Exercício 4. Um funcionário de uma empresa recebe, anualmente, aumento salarial. Sabe-se que:

- Esse funcionário foi contratado em 2005, com salário inicial de R$ 1000,00;
- Em 2006, ele recebeu um aumento de 1,5% sobre seu salário inicial;
- A partir de 2007 (inclusive), os aumentos salariais sempre corresponderam ao percentual do ano anterior mais 0,1%.

Desta forma, criar um programa em PHP que obtém o ano atual (do sistema), que determina e apresenta o salário atual deste funcionário.

Exercício 5. Criar um programa em PHP que receba de uma página HTML o valor de um carro novo, calcule e mostre uma tabela (tabela HTML) com os seguintes dados: preço final, quantidade de parcelas e valor das parcelas. Considere que:

- O preço final para compra à vista tem desconto de 20%;
- A quantidade de parcelas pode ser: 6, 12, 18, 24, 30, 36, 42, 48, 54 e 60; e
- Os percentuais de acréscimo, dependendo da quantidade de parcelas, encontra-se na tabela a seguir:

Quantidade de parcelas	**Percentual de acréscimo sobre o preço final**
6	3%
12	6%
18	9%
24	12%
30	15%
36	18%
42	21%
48	24%
54	27%
60	30%

Exercício 6. Fazer um programa em PHP para gerar e escrever os 4 (quatro) primeiros números perfeitos. Um número perfeito é aquele que é igual à soma dos seus divisores. Por exemplo:

6 = 1 + 2 + 3

28 = 1 + 2 + 4 + 7 + 14

Exercício 7. Fazer um programa em PHP para escrever os X primeiros termos da Sequência de Fibonacci. O valor de X é lido por meio de um formulário HTML. Por exemplo, os oito primeiros termos são:

0 – 1 – 1 – 2 – 3 – 5 – 8 – 13

Exercício 8. Faça uma página HTML para enviar um número inteiro positivo para um programa PHP, que deverá calcular e escrever o fatorial do número recebido. Sabe-se que:

n	n!
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5.040
...	...

Exercício 9. Um cinema de Araraquara fez uma pesquisa entre os seus espectadores. Cada espectador respondeu a um questionário, no qual constava sua idade e sua opinião em relação ao filme assistido:

Opinião	Código
Ótimo	3
Bom	2
Regular	1

Fazer um programa em PHP para receber os dados da pesquisa (idades e opiniões dos espectadores, por meio de páginas HTML – uma para saber o número de espectadores e outra com o formulário de coleta de dados), calcule e apresente:

- A média das idades das pessoas que responderam ótimo;
- A quantidade de pessoas que responderam regular;
- A porcentagem de pessoas que responderam bom, entre todos os espectadores analisados.

Exercício 10. Foi feita uma pesquisa sobre a audiência de canal de TV em várias casas de Araraquara, em determinado dia. Para cada casa foram fornecidos o número do canal (4, 5, 9, 11) e o número de pessoas que estavam assistindo aquele canal. Somente as casas com TV ligada entraram na pesquisa. Faça um programa em PHP que:

- Receba um número de casas pesquisadas;
- Gere um formulário para coletar os dados das casas pesquisadas (canal e números de espectadores);
- Calcule e apresente a porcentagem de audiência de cada canal.

Exercício 11. Foi feita uma pesquisa entre os habitantes de uma região da cidade de Araraquara (coletar o número de habitantes que participaram da pesquisa). Foram coletados os dados de idade, sexo (M/F) e salário. Faça um programa em PHP que, após receber os dados, calcule e mostre:

- A média dos salários do grupo pesquisado;
- A maior e menor idade do grupo;
- A quantidade de mulheres com salário até R$ 1000,00;
- A idade e o sexo da pessoa que possui o menor salário.

Capítulo 6. Variáveis compostas

6.1. Vetores

Um vetor é uma variável composta (*array*) que pode armazenar vários valores ao mesmo tempo, agrupados sob o mesmo nome (identificador) e diferenciados entre si através de índices. Trata-se de uma variável unidimensional, em que cada índice representa uma posição de memória em que fica armazenado um elemento do vetor. Os índices podem ser um número ou um texto e devem aparecer entre colchetes "**[ ]**" logo após o identificador do vetor. A Figura 6.1 apresenta uma representação gráfica de um vetor, com as posições e os índices dos elementos do vetor.

Figura 6.1. Representação gráfica de um vetor.

Em PHP não é necessário que um *array* seja declarado antes de seu uso, nem mesmo indicar o número máximo de elementos que ele deve conter. Além disso, os índices de um *array* podem ser numéricos ou caracteres.

No Exemplo 6.1, é descrito um programa que declara quatro vetores de maneiras diferentes.

```php
<?php
  $quantidade = array();
  $quantidade[0] = 1;
  $quantidade[1] = 2;
  $quantidade[2] = 3;

  $precoUnitario = array(4.32, 5.75, 6.12);

  $cliente1 = array();
  $cliente1["codigo"] = 1234;
  $cliente1["nome"] = "Ana Maria";
  $cliente1["endereco"] = "Rua Nova, 999";

  $cliente2 = array("codigo" => 5678, "nome" => "Juliana",
    "endereco" => "Avenida Central, 1000");
?>
```

Exemplo 6.1. Declaração de vetores.

O vetor **$quantidade** possui índices de 0 a 2 e armazena respectivamente números inteiros de 1 a 3 em cada posição. O vetor **$precoUnitario** armazena os números reais 4.32, 5.75 e 6.12, referentes aos preços de produtos e com índices 0, 1 e 2 correspondentes. O vetor **$cliente1** armazena no índice "código" o valor 1234, no índice

“nome” o valor "Ana Maria" e no índice "endereco" o valor "Rua Nova, 999". Note que o vetor **$cliente2** armazena os dados do cliente 2, com os mesmos índices do vetor **$cliente1**, mas é declarado de maneira diferente.

6.1.1. Leitura e escrita

Em diversas situações, necessitamos ler e/ou escrever em vetores. Para tanto, é comum a utilização da estrutura de repetição ***for*** ou ***foreach***, como apresenta o Exemplo 6.2.

```php
<?php
  $totalDoPedido = array(63.45, 32.27, 56.98, 12.88, 44.12);

  echo "<h3>Valor total de cada pedido</h3>";
  echo "[ ";
  for($i = 0; $i < sizeof($totalDoPedido); $i++){
    echo "R$ $totalDoPedido[$i] ";
  }
  echo " ] <br /><br />";

  $soma = 0;
  foreach($totalDoPedido as $valor){
    $soma = $soma + $valor;
  }

  echo "Valor total dos pedidos = R$ $soma";
?>
```

Exemplo 6.2. Utilização de estruturas de repetição para leitura/escrita de vetores.

No primeiro laço, cinco posições do vetor **$totalDoPedido** são percorridas e os valores armazenados são exibidos. Note que a função ***sizeof()*** determina o tamanho do vetor. No terceiro laço, é utilizada a instrução ***foreach***, onde o vetor é percorrido do primeiro ao último índice e a cada iteração o valor do elemento corrente do vetor é atribuído à variável **$valor** e o ponteiro interno do *array* é avançado. Cada item do vetor é somado e armazenado na variável acumuladora **$soma**, que é exibida no final do código.

Os dados do formulário enviados pelo navegador ao servidor são disponibilizados ao programa PHP por meio de *arrays* superglobais.

No Exemplo 6.3, é descrito um programa que armazena nas variáveis **$nome** e **$idade** as informações disponibilizadas pela variável superglobal **$_POST**.

```php
<?php
  $nome = $_POST["nome"];
  $idade = $_POST["idade"];

  echo "<h2>Nome recebido: $nome</h2>";
  echo "<h3>Idade recebida: $idade</h3>"
?>
```

Exemplo 6.3. Chaves associativas para acesso aos elementos de um vetor.

O programa em PHP deve acessar o *array* apropriado, fornecendo o nome dos campos do formulário como chave associativa. Isto significa que, ao invés de acessar os elementos do *array* pela forma tradicional, utilizando índices numéricos, *strings* serão utilizadas como índices. No exemplo acima, as chaves associativas são **"nome"** e **"idade"**, respectivamente.

6.2. Juntando tudo!

Neste exemplo, veremos como utilizar um formulário HTML para o envio de dados como vetores para realizar a compra de 5 produtos, a recuperação dos dados dos vetores e a geração de uma tabela HTML com os dados recuperados.

O Exemplo 6.4 apresenta o código da página ***form_compras.php***.

```php
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <title>Formulário de Compras</title>
</head>
<body>
  <form action="carrinho_compras.php" method="POST">
    <?php
      for($i = 0; $i < 5; $i++){
        echo '
          <fieldset>
            <legend>Informe os dados do produto '. ($i + 1) . '</legend>
            <p>
              <label>Nome:</label>
              <input type="text" name="nomes[]" />
            </p>
            <p>
              <label>Quantidade:</label>
              <input type="number" name="quantidades[]" />
            </p>
            <p>
              <label>Preço unitário:</label>
              <input type="number" step="0.01" name="precos[]" />
            </p>
          </fieldset>
        ';
      }
    ?>
    <input type="submit" value="Enviar" />
  </form>
</body>
</html>
```

Exemplo 6.4. Código da página form_compras.php.

Dica:

Para obter o resultado do <form> enviado como um array para o script PHP, pode-se nomear (atributo name) os elementos <input>, <select> ou <textarea>, tais como:

```
<input type="text" name="nomes[]" />

<input type="number" name="quantidades[]" />

<input type="number" step="0.01" name="precos[]" />
```

Observe os colchetes após o nome da variável, é isso que torna uma variável um array. Opcionalmente, é possível atribuir chaves específicas para os arrays no HTML. Se não especificar as chaves, o array preenche na ordem sequencial em que os elementos aparecem no formulário, por exemplo, chaves 0, 1, 2, 3 e assim por diante.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <title>Carrinho de Compras</title>
6  </head>
7  <body>
8      <table border="1">
9          <thead>
10             <tr>
11                 <th>Nome</th>
12                 <th>Quantidade</th>
13                 <th>Preço unitário</th>
14                 <th>Total unitário</th>
15             </tr>
16         </thead>
17         <tbody>
18         <?php
19             $nomes = $_POST["nomes"];
20             $quantidades = $_POST["quantidades"];
21             $precos = $_POST["precos"];
22             $valorTotal = 0;
23             for($i = 0; $i < sizeof($nomes); $i++){
24                 echo '
25                     <tr>
26                         <td>'. $nomes[$i] .'</td>
27                         <td>'. $quantidades[$i] .'</td>
28                         <td>'. $precos[$i] .'</td>
29                         <td>'. ($quantidades[$i] * $precos[$i]). '</td>
30                     </tr>
31                 ';
32                 $valorTotal += $quantidades[$i] * $precos[$i];
33             }
34         ?>
35         </tbody>
36         <tfoot>
37             <tr>
38                 <th colspan="3">Valor Total</th>
39                 <th><?php echo $valorTotal; ?></th>
40             </tr>
41         </tfoot>
42     </table>
43 </body>
44 </html>
```

Exemplo 6.5. Código da página carrinho_compras.php.

O Exemplo 6.5 apresenta o código da página ***carrinho_compras.php***. Note que os comandos das linhas 19, 20 e 21 recuperam os nomes, as quantidades e os preços dos 5 produtos, que foram inseridos no formulário HTML do Exemplo 6.4.

Após a recuperação dos dados dos produtos, o programa PHP gera uma tabela HTML com os dados recuperados, além disso, calcula e apresenta o total de cada produto e o total da compra.

6.3. Matrizes

Os *arrays* são estruturas lineares, ou seja, cada elemento de um *array* armazena um valor por vez, por exemplo um número ou uma *string*. Em algumas situações, pode ser necessário que cada elemento de um *array* armazene vários valores por vez, ou seja, armazene um outro *array*. Por exemplo, um *array* pode armazenar uma agenda telefônica,

sendo que cada elemento corresponde a um contato e pode conter um outro *array* armazenando uma lista de números de telefone daquele contato. Para tanto, devem ser utilizadas estruturas multidimensionais chamadas **matrizes**.

As matrizes possuem um único identificador, mas possuem dois ou mais índices para referenciar uma posição de memória. O acesso para cada dimensão de uma matriz é realizado por meio dos colchetes ***[]***. No Exemplo 6.6, que apresenta uma agenda telefônica, **$matrizAgendaTel[0]** permite acessar o *array* que contém a lista de números de telefone do primeiro contato. Para acessar o segundo número de telefone do primeiro contato, deve-se utilizar **$matrizAgendaTel[0][1]**. São definidos três vetores com números de telefone e depois criada uma matriz para armazenar esses vetores. Depois, para cada índice da primeira dimensão da matriz, são exibidos na tela os telefones armazenados em cada vetor, utilizando o comando ***foreach***.

```php
<?php
  $telefones0 = array("1234-5678", "98765-4321");
  $telefones1 = array("1234-0011", "98765-9999", "91234-5555");
  $telefones2 = array("3322-8765");
  $matrizAgendaTel = array($telefones0, $telefones1, $telefones2);

  for($i = 0; $i < 3; $i++){
    echo "Telefones do contato " . $i . ": ";
  foreach($matrizAgendaTel[$i] as $telefone){
    echo "$telefone ";
  }
  echo "<br />";
  }
?>
```

Exemplo 6.6. Agenda telefônica usando matrizes.

No Exemplo 6.7, a variável matriz é um *array* com nove elementos (números inteiros). Note que inicialmente, a variável pode ser um vetor (unidimensional) ou uma matriz (bidimensional). Dentro do primeiro laço ***for***, cada posição (índice) da variável **$matriz** é inicializado como um novo *array*, ou seja, é criado um *array* com duas dimensões (matriz). No segundo laço ***for***, cada posição da matriz armazena o valor da variável **$cont.**

```php
<?php
  $matriz = array();
  $cont = 1;
  for($i = 0; $i < 3; $i++){
    $matriz[$i] = array();
    for($j = 0; $j < 3; $j++){
      $matriz[$i][$j] = $cont++;
    }
  }
  foreach($matriz as $vetor){
    echo "[ ";
    foreach($vetor as $item){
      echo "$item ";
    }
    echo " ]<br />";
  }
?>
```

Exemplo 6.7. Escrita e leitura em matrizes.

Na sequência do exemplo, temos duas estruturas ***foreach***: a primeira percorre a variável **$matriz** dividindo-a num vetor e a segunda dividindo este vetor em itens, exibindo-os na tela.

No Exemplo 6.8, são criadas duas variáveis do tipo *array* (vetores) compostas pelas letras (a, b, c, d, e, f) e a matriz é criada a partir destes dois vetores.

```php
<?php
  $vetor1 = array("a", "b", "c");
  $vetor2 = array("d", "e", "f");
  $matriz = array($vetor1, $vetor2);

  echo "[ ";
  foreach($matriz as $vetor){
    echo "[ ";
    foreach($vetor as $item){
      echo "$item ";
    }
    echo " ]";
  }
  echo " ]";
?>
```

Exemplo 6.8. Declaração de matriz a partir de dois vetores.

Novamente, na sequência do exemplo, temos duas estruturas ***foreach***: a primeira percorre a variável **$matriz** dividindo-a num vetor e a segunda dividindo este vetor em itens, exibindo-os na tela.

6.4. Juntando tudo!

Neste exemplo, apresentamos uma variação da seção 6.2 e veremos como utilizar um formulário HTML para o envio de dados como uma matriz para realizar a compra de 5 produtos, a recuperação dos dados da matriz e a geração de uma tabela HTML com os dados recuperados.

O Exemplo 6.9 apresenta o código da página ***form_compras.php***.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <title>Formulário de Compras</title>
</head>
<body>
  <form action="carrinho_compras2.php" method="POST">
    <?php
      for($i = 0; $i < 5; $i++){
        echo '
          <fieldset>
            <legend>Informe os dados do produto '. ($i + 1) . '</legend>
            <p>
              <label>Nome:</label>
              <input type="text" name="produtos[nome][]" />
            </p>
            <p>
              <label>Quantidade:</label>
              <input type="number" name="produtos[quantidade][]" />
            </p>
            <p>
              <label>Preço unitário:</label>
              <input type="number" step="0.01" name="produtos[preco][]" />
            </p>
          </fieldset>
        ';
      }
    ?>
    <input type="submit" value="Enviar" />
  </form>
</body>
</html>
```

Exemplo 6.9. Código da página form_compras.php.

Após a recuperação dos dados dos produtos, o programa PHP gera uma tabela HTML com os dados recuperados, além disso, calcula e apresenta o total de cada produto e o total da compra, como mostra o Exemplo 6.10.

Dica:

Para obter o resultado do <form> enviado como um array para o script PHP, pode-se nomear (atributo name) os elementos <input>, <select> ou <textarea>, tais como:

```
<input type="text" name="produtos[nome][]" />

<input type="number" name="produtos[quantidade][]" />

<input type="number" step="0.01" name="produtos[preco][]" />
```

Observe os colchetes após o nome da variável, é isso que torna uma variável um array. Opcionalmente, é possível atribuir chaves específicas para os arrays no HTML, por exemplo, “nome”, “quantidade” e “preco”. Se não especificar as chaves, o array preenche na ordem sequencial em que os elementos aparecem no formulário, por exemplo, chaves 0, 1, 2, 3 e assim por diante.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Carrinho de Compras</title>
</head>
<body>
  <table border="1">
    <thead>
      <tr>
        <th>Nome</th>
        <th>Quantidade</th>
        <th>Preço unitário</th>
        <th>Total unitário</th>
      </tr>
    </thead>
    <tbody>
    <?php
      $produtos = $_POST["produtos"];
      $valorTotal = 0;
      for($i = 0; $i < sizeof($produtos["nome"]); $i++){
        $totalUnitario = $produtos["quantidade"][$i] * $produtos["preco"][$i];
        echo '
          <tr>
            <td>'. $produtos["nome"][$i] .'</td>
            <td>'. $produtos["quantidade"][$i] .'</td>
            <td>'. $produtos["preco"][$i] .'</td>
            <td>'. $totalUnitario. '</td>
          </tr>
        ';
        $valorTotal += $totalUnitario;
      }
    ?>
    </tbody>
    <tfoot>
      <tr>
        <th colspan="3">Valor Total</th>
        <th><?php echo $valorTotal; ?></th>
      </tr>
    </tfoot>
  </table>
</body>
</html>
```

Exemplo 6.10. Código da página carrinho_compras.php.

6.5. Exercícios propostos

Exercício 1. Faça um formulário HTML para ler os elementos de um *array* (vetor) com 10 valores reais, enviar os elementos lidos para um programa em PHP que encontra e mostra o maior e o menor valor armazenado no *array*.

Exercício 2. Faça um formulário HTML para ler os elementos de dois vetores: R de 3 elementos inteiros e S de 7 elementos inteiros. Os números lidos devem ser enviados para um programa em PHP que gera um vetor X de 10 elementos cujas 3 primeiras posições contenham os elementos de R e as 7 últimas posições, os elementos de S. Mostrar o vetor X.

Exercício 3. Faça um formulário HTML para ler um vetor U de 10 elementos reais. A seguir, os dados devem ser enviados para um programa em PHP, que troca o primeiro elemento com o último, o segundo com penúltimo etc. até o quinto com o sexto e escreve o vetor U assim modificado.

Exercício 4. Faça um formulário HTML para ler dois vetores, X e Y de 10 elementos inteiros cada um. Os dados lidos são enviados a um programa em PHP, que intercala os elementos desses dois vetores, formando assim um novo vetor R de 20 elementos, onde nas posições ímpares de R estejam os elementos de X e nas posições pares os elementos de Y (Considere o zero como PAR). O programa deve escrever o vetor R, após sua completa geração.

Exercício 5. Faça um programa em PHP que receba o total das vendas de cada vendedor de uma loja e armazene-os em um vetor. Receba também o percentual de comissão de cada vendedor e armazene-os em outro vetor. Receba os nomes desses vendedores e armazene-os em um terceiro vetor. Existem apenas 5 vendedores na loja. O programa deve calcular e mostrar:

- Um relatório com os nomes dos vendedores e os valores a receber referentes à comissão por suas vendas;
- O total das vendas de todos os vendedores;
- O maior valor a receber e o nome de quem o receberá;
- O menor valor a receber e o nome de quem o receberá.

Exercício 6. Faça um programa em PHP que armazena os nomes de sete alunos em um vetor e que armazena também a média final desses alunos. O programa deve calcular e mostrar:

- O nome do aluno com maior média (desconsiderar empates);
- A média da turma;
- A diferença entre a maior e a menor médias.

Exercício 7. Faça um programa em PHP que armazena os nomes de cinco produtos em um vetor e os seus respectivos preços em outro vetor. O programa deve calcular e mostrar:

- A quantidade de produtos com preço inferior a R$ 50,00;
- O nome dos produtos com preço entre R$ 50,00 e R$ 100,00;
- A média dos preços dos produtos com preço superior a R$ 100,00.

Exercício 8. Faça uma página Web com um formulário HTML para preencher uma matriz 5x5 com números inteiros. Um programa em PHP deve calcular e mostrar a soma:

- dos elementos da linha 4;
- dos elementos da coluna 2;
- dos elementos da diagonal principal;
- dos elementos da diagonal secundária;
- de todos os elementos da matriz.

Exercício 9. Faça uma página Web com um formulário HTML para receber os valores das vendas de uma loja. Os dados devem ser armazenados em uma matriz 12x4, em que cada linha representa um mês do ano e cada coluna representa uma semana do mês. Um programa deverá calcular e mostrar:

- O total vendido em cada mês do ano, mostrando o nome do mês por extenso;
- O total vendido em cada semana durante todo o ano, que os proprietários da loja possam identificar a semana que mais teve vendas;
- O total vendido pela loja no ano.

Exercício 10. Faça uma página Web com um formulário HTML para receber o estoque atual e o custo de três produtos de uma distribuidora, que são armazenados em quatro

armazéns. Os dados devem ser armazenados em uma matriz 5x3, sendo que as 4 primeiras linhas contêm os estoques dos produtos nos armazéns e a última linha contém o custo de cada produto. Um programa deverá calcular e mostrar (OBS: devem ser desconsiderados empates):

- A quantidade de itens de produto armazenados em cada armazém;
- Qual armazém possui maior estoque do produto 2;
- Qual armazém possui menor estoque;
- Qual o custo total de cada produto;
- Qual o custo total de cada armazém.

Capítulo 7. Funções

Ao resolver um problema complexo, é comum dividi-lo em partes menores, ou seja, resolver o problema em etapas. A cada uma das partes menores bem definidas chamamos de módulo.

As funções (***functions***) são muito úteis para deixar o código dos programas mais organizado e mais modular. Além disso, as funções nos poupam da tarefa de ter de repetir determinado código toda vez que se precisa realizar uma tarefa.

7.1. Definição

As funções são programas menores inseridos em um programa principal ou em outro arquivo, que pode ser chamado a qualquer instante para executar uma tarefa específica. As funções podem realizar qualquer tipo de tarefa, por exemplo: somar dois números, testar se o valor de uma variável é válido, verificar se um número de CPF é válido, transformar uma *string* para letras maiúsculas, entre outras. A sintaxe para a construção de uma função é:

```
function nome_funcao (arg1, arg2, arg3, ... , argn){
  //instruções
  [ return < expressão > ]
}
```

Onde ***function*** **nome_funcao** é um identificador único, seguindo as mesmas regras de declaração de variáveis: não pode iniciar com número, não pode conter espaços, vírgulas, ponto, entre outras.

Quando uma função é chamada, ela pode receber diversos valores, denominados argumentos (**arg1, arg2,..., argn**). Estes argumentos ou parâmetros são valores recebidos pela função no momento que ela é chamada. Após a função ser chamada, estes valores são processados por ela. Cabe observar que é opcional a utilização de parâmetros. Neste caso, a sintaxe é:

```
function nome_funcao (){
  //instruções
  [ return < expressão > ]
}
```

Neste caso, utilizamos o comando ***return*** quando queremos atribuir o valor retornado a uma variável ou quando precisamos testar o valor de retorno de uma função. No exemplo 7.1, a função **somar** é criada e uma chamada para ela é incluída no programa principal.

```php
<?php

  function somar($operando1, $operando2){

      $resultado = $operando1 + $operando2;

      echo "$operando1 + $operando2 = $resultado";

  }

  $numero1 = 5;

  $numero2 = 10;

  somar($numero1, $numero2);

?>
```

Exemplo 7.1 – Código contendo função com parâmetros.

No momento que a função é chamada, a variável **$operando1** receberá o valor da variável **$numero1** (primeiro argumento), e a variável **$operando2** receberá o valor da variável **$numero2** (segundo argumento). Observe que as variáveis passadas como parâmetro não precisam ter o mesmo nome dos argumentos definidos na função. O objetivo da função é somar dois números e mostrar o resultado na tela e por isso o comando ***return*** não é usado.

Já no exemplo 7.2, a mesma função **somar** é criada, porém, o valor final (resultado da soma) é atribuído a variável **$resultado** (no programa principal).

```php
<?php

  function somar($operando1, $operando2){

    return $operando1 + $operando2;

  }

  $numero1 = 5;

  $numero2 = 10;

  somar($numero1, $numero2);

  echo "$numero1 + $numero2 = " . somar($numero1, $numero2);

?>
```

Exemplo 7.2 – Código contendo função com parâmetros e retorno.

Uma função também pode retornar um ***array*** contendo vários elementos ao invés de somente um, conforme o exemplo 7.3.

```php
<?php

  function ordenar($vetor){

    for($i = 0; $i < sizeof($vetor); $i++){

      for($j = $i + 1; $j < sizeof($vetor); $j++){

        if($vetor[$i] > $vetor[$j]){

          $aux = $vetor[$i];

          $vetor[$i] = $vetor[$j];

          $vetor[$j] = $aux;

        }

      }

    }

    return $vetor;

  }

  $nomes = array("Juliana","Mariana", "Eliana", "Ana", "Adriana");

  $nomes = ordenar($nomes);

  foreach($nomes as $nome){

    echo "$nome ";

  }

?>
```

Exemplo 7.3 – Código contendo função que retorna um array.

No exemplo 7.5, a função ordenar classifica os nomes contidos na variável $vetor de forma a ordená-lo alfabeticamente.

7.2. Escopo de variáveis

O escopo de uma variável se refere ao contexto em que ela foi definida. As variáveis que são declaradas dentro de uma função são chamadas variáveis locais e são conhecidas somente dentro de seu próprio bloco, entre os símbolos abre e fecha chaves (***{ }***). Variáveis locais existem apenas durante a execução do bloco de instruções onde estão declaradas, ou seja, são criadas quando se entra no bloco e destruída na saída. O exemplo 7.4 apresenta variáveis locais e globais.

```php
<?php
  $a = 1; // escopo global

  function Teste() {
    echo 'valor de a: '.$a; //erro: var indefinida
    $b = 2; // variavel local
    echo 'valor de b: '.$b;
  }
  Teste();
?>
```

Exemplo 7.4 - Código contendo variáveis globais e locais.

Note que a variável **$b** é local, só existe no contexto da função Teste. Já a linha 5 contém um erro (mensagem de advertência) uma vez que uma variável global é referenciada dentro da função. Para corrigir este erro é necessário inserir a palavra reservada ***global*** antes do nome da variável, conforme descrito no exemplo 7.5.

```php
<?php
  $a = 1; // escopo global

  function Teste() {
    global $a; // variável global
    echo 'valor de a: '.$a;
    $b = 2; // variavel local
    echo 'valor de b: '.$b;
  }
  Teste();
?>
```

Exemplo 7.5 – Código contendo variáveis globais e locais - corrigido.

As variáveis super globais são variáveis nativas do PHP e recebem esse nome pois estarão presentes em qualquer escopo do programa. As variáveis **$_GET**, **$_POST** (variáveis de requisição) e **$_SESSION** são consideradas superglobais pois são *arrays* predefinidos contendo as variáveis do servidor Web.

Dica:

Para não haver ambiguidade nos códigos, utilize sempre variáveis locais. A maioria das variáveis do PHP tem somente escopo local. Este escopo local inclui os arquivos

incluídos e requeridos (ver capítulo 9). Qualquer variável utilizada dentro de um código ou dentro de uma função é, por padrão, limitada ao escopo local.

7.3. Passagem de parâmetros: valor e referência

Passagem de parâmetros por valor significa que a função receberá cópias dos valores passados no momento que for chamada. Passagem de parâmetros por referência significa que a função receberá endereços de memória com o conteúdo de variáveis. O Exemplo 7.6 contém uma ilustração das duas situações.

```php
<?php

    function dobro1($a){

      $a = 2 * $a;

    }

    function dobro2(&$b){

      $b = 2 * $b;

    }

    $valor = 5;

    dobro1($valor);

    echo "Valor = $valor <br/>";

    dobro2($valor);

    echo "Valor = $valor";

?>
```

Exemplo 7.6 – Código contendo funções com passagem de parâmetros.

No exemplo acima, a função **dobro1** não altera o valor da variável após sua chamada, pois seu valor é alterado somente dentro da função e não gera retorno. Já a função **dobro2** altera o valor da variável após sua chamada, pois o endereço de memória da variável é acessado e seu valor pode ser acessado por outras funções ou pelo programa principal.

7.4. Funções recursivas

Uma função recursiva é definida em termos de si mesma. Ou seja, é recursiva quando dentro dela está chamada a ela própria. O Exemplo 7.7 apresenta o código que calcula o fatorial de um número utilizando uma função recursiva.

```php
<?php

   function fatorial($numero){

      if ($numero < 0){

        return -1;

      }

      if ($numero <=1){

        return 1;

      }

      return $numero * fatorial($numero-1);

   }

   echo "O fatorial de 3 é " . fatorial(3) . " <br/>";

   echo "O fatorial de 4 é " . fatorial(4) . " <br/>";

   echo "O fatorial de 5 é " . fatorial(5) . " <br/>";

?>
```

Exemplo 7.7 – Código que calcula o fatorial de um número utilizando uma função recursiva.

No momento que a função é chamada, um número inteiro **$numero** é passado como parâmetro e dentro da função o valor **$numero-1** é passado como parâmetro novamente. A mesma variável **$numero** acumula os valores das multiplicações sucessivas e em seguida é retornada.

7.5. Juntando tudo!

Até aqui, foram apresentados diversos conceitos básicos para a construção de uma página web dinâmica utilizando a linguagem PHP. O exemplo descrito nessa seção visa juntar os conteúdos já vistos e aplicá-los em um caso prático.

Primeiramente, uma página ***index.html*** é criada com um formulário esperando os dados de entrada. O usuário digita o preço de três mercadorias, *smartphone*, *notebook* e *smart tv*. Em seguida, ele escolhe a forma de pagamento: a vista, a prazo em 3 vezes ou a prazo em 5 vezes, conforme ilustrado no Exemplo 7.8.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
      <title>Descontos - Página Inicial</title>
  </head>
  <body>
    <form action="calc_desc.php" method="post">
      <fieldset>
        <legend>Digite o valor de cada produto:</legend>
        <fieldset>
            <p>
              <label>Smartfone:</label>
              <input type="text" name="smartfone" />
              <label>Notebook:</label>
              <input type="text" name="notebook" />
              <label>Smartv:</label>
              <input type="text" name="smartv" />
            </p>
      </fieldset>
            <p><label>Forma de pagamento:</label>
        <p><input type="radio" name="condicao" value="vista">a vista</p>
        <p><input type="radio" name="condicao" value="prazo3x">a prazo em 3x</p>
        <p><input type="radio" name="condicao" value="prazo5x">a prazo em 5x</p>

        <input type="submit" value="Calcular desconto" />
      </fieldset>
    </form>
  </body>
</html>
```

Exemplo 7.8 – Página index.html que contém o formulário.

Quando o usuário submete o formulário, a página ***calc_desc.php*** é chamada. Esta página calcula o valor da compra à vista, com ou sem desconto, conforme o Exemplo 7.9.

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Orçamentos - Formulário de Produtos</title>
  </head>
  <body>
    <?php
      $smartfone = $_POST["smartfone"];
      $notebook = $_POST["notebook"];
      $smartv = $_POST["smartv"];
      $cond = $_POST["condicao"];

      $total=$smartfone+$notebook+$smartv;

      echo '<p>Valor da compra sem desconto:'.$total;

      echo '<p>Valor da compra com desconto:'.calcular($total, $cond);

      function calcular($t1, $c1){
        if ($c1=="vista"){
          $t1=$t1*0.90;

        }else if ($c1=="prazo3x"){
          $t1=$t1*1.10;
        } else {
          $t1=$t1*1.20;
        }
        return $t1;
      }
    ?>
  </body>
</html>
```

Exemplo 7.9 – Página calc_desc.php que calcula o valor da compra.

A função calcular recebe como parâmetros o total da compra (**$total**) e a condição de pagamento (**$cond**). Se a compra for a vista, o desconto será de 10%. Se a compra for a prazo em três vezes, terá um acréscimo de 10% e se for a prazo em 5 vezes, um acréscimo de 20%.

7.6. Exercícios propostos

Exercício 1. Faça um programa em PHP que receba três valores (obrigatoriamente maiores que zero), representando as medidas dos três lados de um triângulo. Elabore funções (sub-rotinas) para:

- Validar se os valores informados são maiores que zero;

- Determinar se esses lados informados forma um triângulo (sabe-se que, para ser triângulo, a medida de um lado qualquer deve ser inferior à soma das medidas dos outros dois lados);
- Determinar e retornar o tipo de triângulo (equilátero, isósceles ou escaleno), caso as medidas formem um triângulo.

* Todas as mensagens devem ser mostradas no programa principal (fora que qualquer função).

Exercício 2. Faça um programa em PHP para analisar as temperaturas médias de cada mês do ano. Assim o programa deve conter as seguintes funções para:

- Receber a temperatura média de cada mês do ano e armazene-as em um vetor (*array*);
- Receber um mês em número e retornar o mês por extenso: 0 – janeiro; 1 – fevereiro; 2 – março; ...);
- Calcular e retornar a maior temperatura do ano e em qual mês ocorreu;
- Calcular e retornar a menor temperatura do ano e em qual mês ocorreu;
- Calcular e retornar a média anual de temperaturas.

* Todas as mensagens devem ser mostradas no programa principal (fora que qualquer função).

Capítulo 8. Cookies e Sessões

As aplicações desenvolvidas para Web podem oferecer diversas funcionalidades a partir dos dados sobre os usuários enquanto eles navegam pelas páginas ou até mesmo após eles deixarem um Website, possibilitando um aprimoramento da sua experiência de navegação ou utilização do Website. Para implementação de tais funcionalidades, é importante conhecer e saber manipular cookies e sessões por meio da linguagem PHP.

Quando diferentes usuários acessam e navegam nas páginas HTML de um Website, o servidor Web trata esses acessos como requisições independentes. Ele não sabe que pessoas diferentes estão acessando e requisitando suas páginas. Os mecanismos fornecidos pelos cookies e pelas sessões permitem armazenar os dados de cada usuário enquanto eles estiverem navegando entre as páginas de um Website, possibilitando a utilização desses dados para diversos fins, tais como a autenticação de usuário, carrinho de compras, exibição de anúncios e personalização de páginas, entre outros.

8.1 Cookies

Um cookie é um pequeno arquivo de texto que fica armazenado no dispositivo (computador ou *smartphone*) do usuário. Ao requisitar uma página, o servidor Web envia a resposta da requisição para o navegador. Em conjunto com essa resposta, o servidor também pode enviar um cabeçalho HTTP Set-Cookie, solicitando que o navegador crie cookies em seu dispositivo.

Os dados de um cookie podem ser recuperados posteriormente pelo servidor Web e utilizados para identificação do usuário e lembrança de suas preferências (configurações escolhidas, links clicados, tempo navegando no Website, itens de um carrinho de compras etc.), permitindo inclusive, a personalização da página de acordo com o perfil do usuário. É possível realizar também o compartilhamento de dados entre diferentes páginas, ou até mesmo, diferentes acessos. Após sua criação, os dados do cookie são enviados para o servidor Web em todas as novas requisições realizadas pelo navegador, permitindo a troca de dados entre o navegador e o servidor.

Um cookie é formado por um par ***nome => valor*** (também descrito como ***chave => valor***), ou seja, ele possui um nome (ou chave) pelo qual é referenciado e um valor associado a este nome (ou chave). Cookies podem permanecer armazenados no computador/smartphone dos usuários por vários dias ou somente durante o tempo em que o navegador do usuário permanecer aberto. Em PHP, essas configurações do tempo para

o cookie expirar são realizadas no momento de sua criação por meio do parâmetro ***expire***, que faz parte da função ***setcookie()***, uma das funções disponibilizadas para a manipulação de cookies.

A função ***setcookie()*** é utilizada para enviar cookies ao dispositivo do usuário. A execução dessa função permite tanto a definição de um cookie, quanto sua exclusão. Por exemplo, para definição de um cookie que armazena o nome de uma pessoa, pode-se utilizar o comando:

```
setcookie("usuario", "IFSP");
```

Neste caso, o valor “IFSP” estará associado ao nome/chave “usuario”. Ao fechar o navegador, encerrando sua sessão, o cookie será excluído automaticamente.

Para que o cookie seja armazenado por um determinado período, é necessário estabelecer uma data de expiração, como descrito a seguir:

```
setcookie("usuario", "IFSP", time() + 60 * 60);
```

Neste caso, o cookie será criado com a data atual (fornecida pela função ***time()***) acrescida de uma hora (60 segundos vezes 60 minutos).

Para excluir esse cookie, basta utilizar a mesma função da seguinte forma:

```
setcookie("usuario", "IFSP", time() - 1);
```

Ao especificar “-1”, a data de expiração é configurada para um momento no passado, acionando o mecanismo de remoção de cookies do navegador.

Importante:

Para que a remoção do cookie seja realizada, é necessário manter a mesma quantidade de parâmetros que foram especificados no momento de sua criação. Caso isso não ocorra, o cookie não será removido. No exemplo acima foram especificados os parâmetros nome, valor e validade para a criação do cookie. Dessa forma, os mesmos parâmetros devem ser especificados na função de remoção desse cookie.

Outros parâmetros possíveis para a função **setcookie()** estão descritos na tabela a seguir:

Parâmetro	Descrição
Nome (name)	Indica o nome do cookie que está sendo enviado e é o único parâmetro obrigatório para a função
Valor (value)	É o valor do cookie. Se não for fornecido, o servidor tentará excluir o cookie com o nome especificado
Validade (expire)	Define o tempo de validado do cookie. Deve ser expresso no formato-padrão de tempo do UNIX (número de segundos após 1º de janeiro de 1970, às 0h).
Caminho (path)	Caminho no servidor para o qual o cookie estará disponível. Se for definido o valor "/", ele estará disponível para todo o domínio especificado no parâmetro domínio. O valor padrão é o diretório corrente a partir do qual o cookie foi definido
Domínio (domain)	Domínio para o qual o cookie estará disponível.
Seguro (secure)	É um valor inteiro (0 ou 1) que indica se o cookie é seguro. Se for utilizado o valor 1, o cookie só será transmitido se a conexão for segura (HTTPS).
Somente HTTP (httponly)	Se for igual a TRUE, o cookie estará acessível apenas sobre o protocolo HTTP e não acessível para linguagens de script, tal como JavaScript.

Para que o código PHP de uma página consiga acessar os dados de um cookie no dispositivo (computador ou smartphone) do usuário, utiliza-se o array superglobal ***$_COOKIE***. Caso o cookie seja definido como:

```
setcookie("usuario", "IFSP");
```

na próxima página acessada pelo usuário, esse valor pode ser recuperado da seguinte forma:

```
$_COOKIE["usuario"];
```

Saiba mais:

Com o início de vigência da **Lei Geral de Proteção de Dados** Pessoais (LGPD), em agosto de 2020, que dispõe sobre o tratamento de dados pessoais, inclusive nos meios digitais, tornou-se fundamental o gerenciamento adequado dos cookies criados por sua aplicação Web de acordo com a padronização de normas e práticas

estabelecidas por essa lei, sob risco de sanções e multas devido ao seu descumprimento. Dessa maneira, sua aplicação Web deve coletar, utilizar ou compartilhar dados pessoais em conformidade com as regulações estabelecidas pela lei. Para mais informações, acesse:

LEI Nº 13.709, DE 14 DE AGOSTO DE 2018

(http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm).

8.1.1. Juntando tudo

No primeiro exemplo, é realizada a criação de um cookie por meio da função ***setcookie()***. O parâmetro inicial especifica a descrição (nome) do cookie, e o segundo parâmetro especifica o valor associado ao nome. No ***body*** da página, o código PHP verifica se o cookie foi criado. Em caso afirmativo, o nome do usuário é exibido. Caso contrário, a mensagem genérica “Bem vindo, visitante!” é apresentada.

```
1  <?php
2    setcookie("usuario", "IFSP");
3  ?>
4  <!DOCTYPE html>
5  <html lang="pt-BR">
6    <head>
7      <meta charset="UTF-8">
8      <title>Cookies - Exemplo 1</title>
9      <link rel="stylesheet" href="css/estilo.css">
10   </head>
11   <body>
12     <?php
13       if (isset($_COOKIE["usuario"])) {
14         echo "Bem vindo, {$_COOKIE["usuario"]}!";
15       } else {
16         echo "Bem vindo, visitante!";
17       }
18     ?>
19   </body>
20 </html>
```

Exemplo 8.1 – Criação de cookie

A estilização a seguir foi utilizada para uma formatação simples da página:

```css
* {
  margin: 0;
  padding: 0; }

body {
  font-family: sans-serif;
  background-color: #afe6de;
  padding: 10px 20px; }
```

Exemplo 8.2 – Criação de cookie (Estilização)

No segundo exemplo, é realizada a manipulação de cookies por meio de sua criação, recuperação e remoção. Na página ***index.php*** são inseridos três formulários. Em cada um deles, uma ação é associada a um script PHP específico, responsável pela implementação da funcionalidade descrita pelo atributo ***value*** de cada botão do tipo ***submit*** incluído no respectivo formulário.

Verifique o código fonte do arquivo ***index.php***:

```html
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8">
    <title>Cookies - Exemplo 2</title>
    <link rel="stylesheet" href="css/estilo.css">
  </head>
  <body>
    <header>
      <h4>Manipulação de Cookies</h4>
    </header>
    <section>
      <br>
      <form action="registrar_acesso.php" method="post">
        <label for="id_usuario">Informe o nome do usuário:</label>
        <input type="text" name="usuario" id="id_usuario">
        <br><br>
        <input type="submit" value="Registrar Usuário">
      </form>
      <form action="listar_usuario.php" method="post">
        <input type="submit" value="Listar Usuário">
      </form>
      <form action="remover_acesso.php" method="post">
        <input type="submit" value="Remover Usuário">
      </form>
    </section>
  </body>
</html>
```

Exemplo 8.3 – Manipulação de cookies

No script ***registrar_acesso.php***, o nome de usuário digitado no campo de texto "usuario" do formulário é recuperado por meio da superglobal ***$_POST*** e atribuído à variável ***$nome_usuario***. A seguir, é atribuído à variável ***$expira*** a hora atual acrescida de 1 hora, especificando a validade do cookie. Após isso, a função ***setcookie()*** registra o cookie.

```php
<?php
  $nome_usuario = $_POST["usuario"];

  $expira = time() + 60 * 60;
  setcookie("usuario", $nome_usuario, $expira);
  echo "Registro de {$nome_usuario} realizado com sucesso!";

  echo("<p><a href=\"index.php\">Voltar</a></p>");
?>
```

Exemplo 8.4 – Criação de cookie

Já no script ***listar_usuario.php***, utiliza-se a função ***isset()*** para verificar se existe um nome/chave identificado por “usuario” no array de cookies ***$_COOKIE***. Caso essa chave exista no array, o valor associado a ela (nome do usuário) é exibido. Caso contrário, a mensagem “Cookie inexistente!” é mostrada.

```
1  <?php
2    if (isset($_COOKIE["usuario"])) {
3      echo("O usuário {$_COOKIE["usuario"]} está registrado!");
4    } else {
5      echo "Cookie inexistente!";
6    }
7    echo("<p><a href=\"index.php\">Voltar</a></p>");
8  ?>
```

Exemplo 8.5 – Acesso e exibição do cookie criado

Por fim, no script ***remover_acesso.php***, é realizada uma verificação similar a do script ***listar_usuario.php***. Se o cookie usuário existir, ele será removido por meio do código da linha 3, que especifica “***time() -1***” para o parâmetro ***expire***, configurando a data de expiração para um momento no passado, de maneira que o mecanismo de remoção de cookies do navegador é acionado.

```
1  <?php
2    if (isset($_COOKIE["usuario"])) {
3      setcookie("usuario", "", time() - 1);
4      echo("Registro de {$_COOKIE["usuario"]} removido com sucesso!");
5    } else {
6      echo "Cookie inexistente!";
7    }
8    echo("<p><a href=\"index.php\">Voltar</a></p>");
9  ?>
```

Exemplo 8.6 – Remoção de cookie

8.2 Sessões

Uma sessão pode ser definida como o período de tempo no qual uma pessoa navega pelas páginas de um Website. Assim, quando o site é acessado, uma nova sessão pode ser iniciada. Nela, são registradas diversas variáveis que ficam armazenadas em arquivos no servidor e que podem ser acessadas em qualquer página da aplicação, enquanto a sessão estiver aberta. Cada sessão possui um identificador único, chamado de session ID (SID).

Em PHP, uma sessão pode ser criada utilizando-se a função ***session_start()***, que possibilita também a restauração os dados de uma sessão, com base no session ID corrente. Essa função deve ser chamada antes de qualquer saída produzida pelo navegador (no início do arquivo). Uma outra forma de criar uma sessão é habilitando a diretiva ***session.auto_start*** do arquivo ***php.ini***, presente no servidor Web. Com essa diretiva, sempre que um usuário entrar no site, uma sessão será criada automaticamente. A primeira forma será adotada nos exemplos apresentados.

Ao registrarmos uma variável em uma sessão, ela se torna disponível para todas as páginas que serão acessadas até o encerramento da sessão. Para registrar uma variável, deve-se adicionar diretamente entradas ao array superglobal ***$_SESSION***, conforme o exemplo:

```
$_SESSION["usuario"] = "IFSP";
```

Caso uma variável de sessão tenha sido registrada e não for mais utilizada nos próximos acessos, ela pode ser eliminada. Isto é feito por meio da função ***unset()*** do PHP, como é mostrado no exemplo:

```
if (isset($_SESSION["usuario"])){
  unset($_SESSION["usuario"]);
}
```

Para demonstrar o funcionamento do registro de variáveis em uma sessão, considere o exemplo abaixo, que utiliza duas páginas chamadas de ***pagina_1.php*** e ***pagina_2.php***. Na primeira página, são definidas e registradas duas variáveis de sessão e é exibido um link para a segunda página. A segunda página recupera os dados da sessão e exibe os valores atribuídos ainda na primeira página. O código da ***pagina_1.php*** é o seguinte:

```
1  <?php
2    session_start();
3    echo "Página 1";
4    $_SESSION["nome"] = "José";
5    $_SESSION["sobrenome"] = "de Souza";
6    echo "<br><a href=\"pagina_2.php\">Página 2</a>";
7  ?>
```

Exemplo 8.7 – Sessões: Página 1

Na segunda página, é preciso, primeiramente, restaurar os dados da sessão por meio da função ***session_start()***. Em seguida, os valores das variáveis “nome” e “sobrenome” são recuperados pelo acesso ao vetor ***$_SESSION***. Veja o código:

```php
<?php
   session_start();
   echo "Página 2" . "<br>";
   echo $_SESSION["nome"] . "<br>";
   echo $_SESSION["sobrenome"] . "<br>";
   echo "<br><a href=\"pagina_1.php\">Página 1</a>";
?>
```

Exemplo 8.8 – Sessões: Página 2

8.2.1. Juntando tudo

Neste exemplo utilizaremos formulários, envio de informações e sessões para realizar o cadastro de diversos produtos em uma sessão utilizando uma única página HTML.

```php
<?php
  session_start();
?>
<html lang="pt-BR">
  <head>
    <meta charset="utf-8"/>
    <title>Formulário de Cadastro de Produtos</title>
  </head>
  <body>
    <?php
      if(empty($_POST)){
        $_SESSION["cadastrou"] = 0;
        echo '<form action="form_cadastro_produto.php" method="post">
                <fieldset>
                  <legend>Cadastro de produto</legend>
                  <p>
                    <label>Nome:</label>
                    <input type="text" name="nome" size="30"/>
                  </p>
                  <p>
                    <label>Preço</label>
                    <input type="number" step="0.01" name="preco"/>
                  </p>
                  <input type="submit" value="Enviar"/>
                </fieldset>
              </form>
              <a href="lista_produtos.php">Listar produtos cadastrados</a>';
      } else {
        if($_SESSION["cadastrou"] == 0){
          $_SESSION["nomes"][] = $_POST["nome"];
          $_SESSION["precos"][] = $_POST["preco"];
          $_SESSION["cadastrou"] = 1;
          echo '<br/><br/>
                <div class="center">
                  <h2>Produto cadastrado com sucesso.</h2>
                  <p><a href="form_cadastro_produto.php">Voltar</a></p>
                </div>';
        } else {
          echo '<br/><br/>
                <div class="center">
                  <h2>Produto já cadastrado.</h2>
                  <p><a href="form_cadastro_produto.php">Voltar</a></p>
                </div>';
        }
      }
    ?>
  </body>
</html>
```

Exemplo 8.9 – Formulário de cadastro dos produtos

Antes de entender o exemplo, é preciso conhecer a função ***empty()***. Esta função testa se uma determinada variável, informada como parâmetro, está vazia. Uma variável é

considerada vazia se ela não existir ou se o seu valor for igual a ***FALSE*** (falso). Para um array, isso significa não possuir elemento algum.

A função ***empty()*** será utilizada para determinar o contexto do carregamento da página. Ao aplicar, por exemplo, a função ***empty()*** no vetor superglobal ***$_POST***, é possível verificar se o carregamento da página atual foi realizado por meio da submissão de um formulário (***$_POST*** contendo dados de formulário), ou se a página está sendo aberta sem receber informações da página anterior (***$_POST*** vazio). Utilizaremos esse status do vetor superglobal para estabelecer o momento de exibir o formulário de cadastro do produto e o momento de registrar as informações de um produto na sessão.

No exemplo, o vetor superglobal ***$_SESSION*** também armazena uma flag com o nome “cadastrou” para verificar se os dados de um formulário já foram devidamente submetidos e registrados na sessão. Isto previne que, ao atualizar a página, o usuário registre novamente os dados submetidos pelo formulário. Este artifício não previne, contudo, que usuários cadastrem produtos com o mesmo nome.

O código a seguir pertence à página ***lista_produtos.php***. Sua função é exibir todos os produtos cadastrados na sessão e seus respectivos preços em uma tabela. O link para essa página encontra-se abaixo do formulário exibido na página ***form_cadastro_produto.php***. Caso a sessão não possua produtos cadastrados (vetor ***$_SESSION*** vazio), a página emite uma mensagem de alerta.

```
1  <?php
2    session_start();
3  ?>
4  <html lang="pt-BR">
5    <head>
6      <meta charset="utf-8"/>
7      <title>Lista de Produtos</title>
8      <link rel="stylesheet" href="css/estilo.css">
9    </head>
10   <body>
11     <?php
12       if(!empty($_SESSION["nomes"])){
13         echo '
14           <div class="center">
15             <table class="tabela1" width="30%" border="1">
16               <thead>
17                 <tr>
18                   <th>Nome</th>
19                   <th>Preço</th>
20                 </tr>
21               </thead>
22               <tbody>';
```

Exemplo 8.10 – Listagem de produtos (Parte 1)

```
23                  for($i = 0; $i < sizeof($_SESSION["nomes"]); $i++){
24                    $nome = $_SESSION["nomes"][$i];
25                    $preco = $_SESSION["precos"][$i];
26                    echo '
27                      <tr>
28                        <td>' . $nome . '</td>
29                        <td>' . $preco . '</td>
30                      </tr>';
31                  }
32                  echo '
33                    </tbody>
34                      </table>
35                        </div>
36                        <a href="form_cadastro_produto.php">Cadastrar produto</a>';
37              } else {
38                echo '
39                  <br/><br/>
40                  <div class="center">
41                    <h2>Nenhum produto cadastrado.</h2>
42                  </div>
43                  <a href="form_cadastro_produto.php">Cadastrar produto</a>';
44              }
45      ?>
46    </body>
47  </html>
```

Exemplo 8.11 – Listagem de produtos (Parte 2)

A estilização a seguir foi utilizada para uma formatação simples da página:

```
1   .center{
2     margin: auto;
3   }
4
5   .tabela1 tbody tr:nth-child(odd){
6     background-color: #E9E9E9;
7   }
```

Exemplo 8.12 – Cadastro de produtos (Estilização)

8.3. Exercícios de fixação

Exercício 1. Faça uma página HTML com um formulário para o usuário informar as preferências: cor de fundo (background-color) e tamanho do texto (text-size). Os dados informados devem ser enviados para um programa PHP, que recebe os dados e os armazena em cookies. Crie também um arquivo PHP que recupera os dados dos cookies e gera uma página com as preferências do usuário.

Exercício 2. Faça um programa em PHP que gerencia os dados de pessoas, armazenando-os em uma agenda de contatos. Assim, crie:

- Um programa form_cadastro_pessoa.php, como mostra a Figura 1, para receber os dados de uma pessoa (nome, telefone e endereço) e armazene-os na sessão. Após armazenar os dados, o programa deve apresentar a mensagem de sucesso.
- Um programa lista_pessoas.php para gerar uma tabela com os dados de todas as pessoas, ou seja, a agenda de contatos do usuário, Figura 2.
- Um programa form_consulta_pessoa.php para que o usuário informe o nome da pessoa e retorne os dados dessa pessoa ou uma mensagem que a pessoa não foi encontrada, Figura 3.

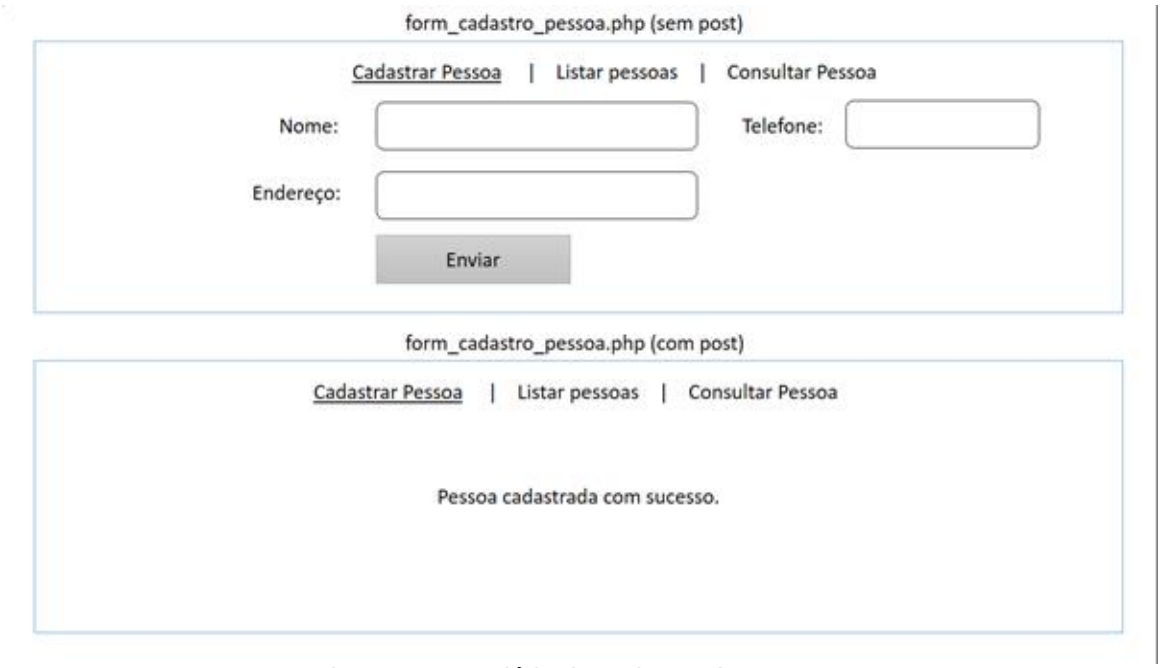


Figura 1. Formulário de cadastro de pessoas

Figura 2. Lista de pessoas cadastradas

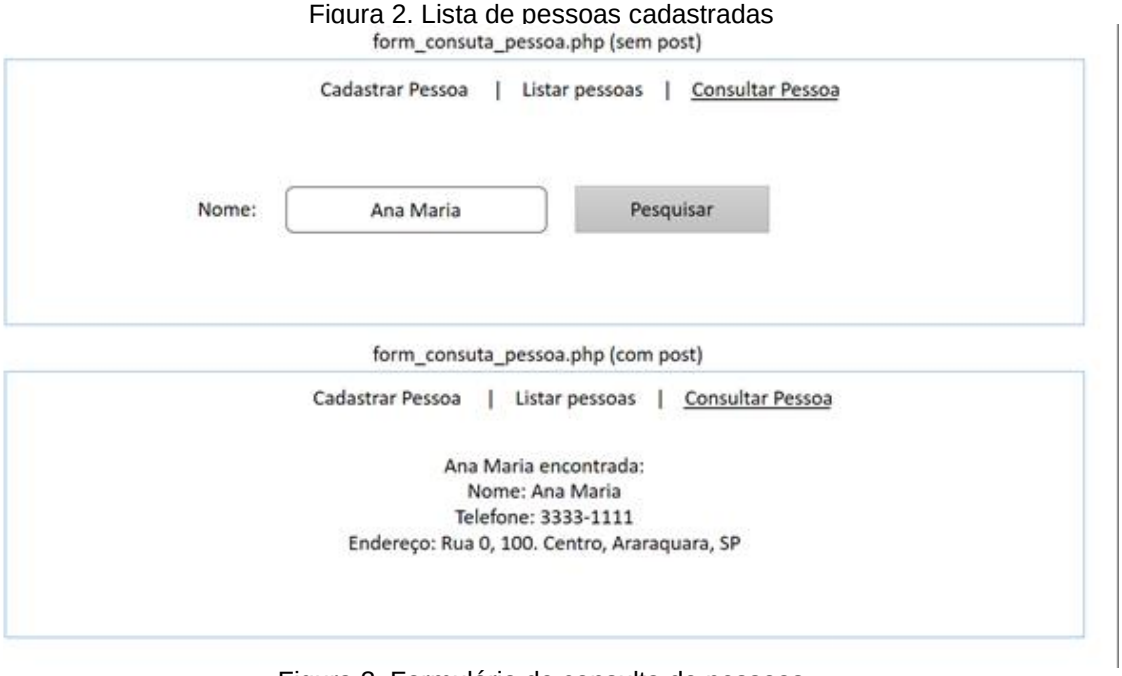


Figura 3. Formulário de consulta de pessoas

Capítulo 9. Importação com Include e Require

No desenvolvimento de software, existem muitas estratégias diferentes que podem ser adotadas com o objetivo de reutilizarmos código. A reutilização de código permite reduzir o esforço e custo de desenvolvimento e contribui para a melhoria da qualidade do produto, uma vez que o código a ser reutilizado, muitas vezes, já é parte de outras aplicações, portanto, foi testado e está funcionando. Em PHP, uma das estratégias de reutilização de código aponta para a definição e inclusão de arquivos que contenham código comum a diferentes partes da aplicação.

9.1. Comandos include e require

Suponha que seja necessário desenvolver uma aplicação Web que possua 20 páginas, e todas elas devem exibir um determinado menu (o mesmo menu). Uma primeira abordagem para resolver este problema seria simplesmente escrever o código do menu nas 20 páginas, mas esta seria uma solução ineficiente e difícil de manter. Se, em algum momento, o menu tiver que ser modificado (por exemplo, adicionando-se ou removendo-se algum link), todas as páginas que dependem deste menu precisarão ser alteradas. Dessa forma, uma solução mais adequada seria definir o menu em uma página à parte e incluir esta página em todas as partes da aplicação que necessitam dela. Em PHP, um dos comandos que permitem incluir o conteúdo de uma página em outras páginas é o comando ***include***.

Quando o comando ***include*** é utilizado, todo o conteúdo do arquivo incluído é copiado e avaliado no ponto do código onde o comando foi especificado. Por exemplo, se o arquivo ***X.php*** incluir, em um dado ponto do código, o arquivo ***Y.php***, todo o conteúdo de ***Y.php*** (pode ser código HTML, PHP etc.) será copiado para o ponto do código onde o comando ***include*** foi utilizado (é como se copiássemos manualmente todo o código de um arquivo em outro). Os Exemplos 9.1 e 9.2 ilustram, respectivamente, o código do menu a ser incluído e o código de um script qualquer que necessita do menu.

```
1  <p><strong>Cursos</strong></p>
2  <p><a href="ads.php">ADS</a></p>
3  <p><a href="si.php">Sistemas de Informação</a></p>
4  <p><a href="cc.php">Ciência da Computação</a></p>
```

Exemplo 9.1 - Código do arquivo “menu.inc”.

Uma vez que o arquivo ***menu.inc*** foi criado, ele poderá ser incluído por qualquer arquivo PHP por meio do comando ***include***. Conforme o Exemplo 9.1 mostra, o conteúdo do arquivo ***menu.inc*** é código HTML apenas, mas poderia conter código PHP também. Além disso, a extensão ***".inc"*** é apenas uma convenção para indicar que se trata de um arquivo cujo conteúdo é incluído por outros arquivos na aplicação. O Exemplo 9.2 mostra o código de um script em PHP que utiliza o comando ***include*** para incluir o conteúdo do arquivo ***menu.inc***. Retomando-se o exemplo inicial, nas 20 páginas da aplicação, o comando ***include*** deverá ser especificado no local onde deve aparecer o menu. Caso seja necessário modificar o menu, bastará modificar o código HTML do arquivo ***menu.inc***, e as alterações serão "enxergadas" por todas as páginas.

```php
<?php
   //algum código aqui
   include "menu.inc";
   //algum código aqui
?>
```

Exemplo 9.2 - Código de um arquivo em PHP que inclui o conteúdo de "menu.inc".

De forma similar, no mesmo exemplo, outros dois comandos ***include*** poderiam ser especificados para a reutilização de cabeçalhos e rodapés de uma aplicação. Para isso, seriam criados os arquivos ***cabecalho.inc*** e ***rodape.inc***, e em cada página da aplicação, no topo e no rodapé, seria especificado o comando ***include*** para os arquivos criados. Assim, um arquivo PHP pode incluir um ou vários arquivos por meio do comando ***include***. Nesse momento, vale fazermos a seguinte reflexão: O que acontecerá no código se o arquivo a ser incluído não for encontrado ou não puder ser incluído por qualquer motivo? Utilizando-se o comando ***include***, uma advertência (não um erro) será produzida, e o código continuará sendo executado. No exemplo, caso o arquivo ***menu.inc*** não seja encontrado, o código do script que inclui este arquivo continuará funcionando (todas as instruções especificadas após a chamada ao ***include*** serão executadas normalmente).

Entretanto, há situações em que não podemos permitir que o código continue sendo executado caso o arquivo não possa ser incluído. Um exemplo seria o código de uma aplicação que precise acessar um banco de dados. Suponha que, nesta aplicação, existem vários scripts (vários arquivos PHP) que acessam e manipulam dados de tabelas em um banco de dados. Sabemos que o primeiro passo para que uma aplicação se comunique com um banco de dados é estabelecer uma conexão. Como existem vários scripts que

precisarão de uma conexão com o banco de dados, todo o código responsável pela conexão pode ser escrito em um arquivo à parte e incluído pelos outros scripts. Neste caso, se o arquivo responsável pela conexão não puder ser incluído, não poderemos permitir que o código continue sendo executado, afinal a execução dos outros scripts depende da obtenção de uma conexão com o banco de dados.

Assim, o comando ***include*** não é o mais adequado quando a inclusão de um ou mais arquivos é crítica para a execução de um dado script. Neste tipo de situação, devemos utilizar o comando ***require***. Este comando é idêntico ao comando ***include***, exceto pelo fato de que, se utilizarmos o comando ***require***, e o arquivo exigido não puder ser incluído, um erro fatal será gerado pelo comando, e a execução do script será **interrompida**. O Exemplo 9.3 a seguir ilustra esta situação.

```php
<?php
   //algum código aqui (sempre será executado)
   require "conexao.php";
   //algum código aqui
   //(somente será executado se conexao.php for incluído)
?>
```

Exemplo 9.3 - Utilização do comando require em vez do comando include.

No exemplo inicial, como foi utilizado o comando ***include*** para a inclusão do menu, caso, por algum motivo, o arquivo com o código do menu não pudesse ser incluído, o código subsequente dos scripts que dependem do menu continuaria sendo executado normalmente. Apenas o menu não seria exibido. Se esta é uma situação a ser evitada, isto é, se o código dos scripts depende fortemente da inclusão do menu, o comando ***require*** deverá ser utilizado no lugar do comando ***include***.

9.2. Juntando tudo

Considere uma aplicação para locação de carros, contendo páginas para cadastro de clientes, cadastro de carros e cadastro de locações realizadas. Todas as páginas devem conter o mesmo cabeçalho e um menu com as funcionalidades da aplicação. Assim, estes elementos podem ser definidos em arquivos separados e incluídos em todas as páginas (utilizando-se *include* ou *require*). O código HTML do arquivo ***cabecalho.inc*** é apresentado no Exemplo 9.4.

```
<header>
    <img src="carro.png" width="80" height="80"
    alt="foto de um carro" />
    <h2>IFSP-Car - Sua Locadora de Carros</h2>
    <hr />
</header>
```

Exemplo 9.4 - Arquivo de cabeçalho a ser incluído em outras páginas (“cabecalho.inc”).

O código HTML do arquivo ***menu.inc*** é apresentado no Exemplo 9.5 a seguir. Dessa forma, todas as páginas da aplicação deverão conter o comando ***include*** para os arquivos de cabeçalho e menu.

```
<nav>
    <ul>
     <li><a href="index.php">Home</a></li>
     <li><a href="form_cadastro_cliente.php">Cadastro de Clientes</a></li>
     <li><a href="form_cadastro_carro.php">Cadastro de Carros</a></li>
     <li><a href="form_locacao_carros.php">Cadastro de Locações</a></li>
    </ul>
</nav>
```

Exemplo 9.5 - Arquivo com o menu a ser incluído em outras páginas (menu.inc).

O código da página inicial da aplicação Web (***index.php***) é apresentado no Exemplo 9.6 a seguir. Note que todas as páginas que utilizam comandos ***include*** ou ***require*** devem ser arquivos escritos em PHP.

```php
<!DOCTYPE html>
<html lang="pt-BR">
    <head>
        <meta charset="utf-8">
        <title>Locadora de Carros - Página Inicial</title>
    </head>
    <body>
        <div class="container">
            <?php
                include "cabecalho.inc";
                include "menu.inc";
            ?>
            <h2>Seja bem-vindo(a)!</h2>
        </div>
    </body>
</html>
```

Exemplo 9.6 - Página inicial da aplicação (index.php).

O código da página de cadastro de clientes (arquivo ***form_cadastro_cliente.php***) é apresentado no Exemplo 9.7 a seguir. Neste arquivo, é necessário incluir, além dos arquivos para o cabeçalho e o menu, o arquivo ***funcoes.inc***, conforme será explicado adiante. Se o *array* ***$_POST*** estiver vazio, ou seja, o conteúdo do formulário ainda não foi submetido via método POST, o arquivo ***form_cliente.inc*** será incluído. O Exemplo 9.8 a seguir apresenta o código do arquivo ***form_cliente.inc***.

O arquivo ***form_cliente.inc*** contém o formulário HTML de cadastro de clientes, com os campos nome, e-mail, endereço e telefone. Quando o usuário inserir os dados e clicar no botão “Enviar”, o arquivo ***form_cadastro_cliente.php*** efetuará o tratamento da requisição enviada via método POST. Dessa forma, o *array* ***$_POST*** não será mais vazio, pois estará preenchido com os dados dos campos do formulário. Na sequência, será chamada a função ***ler_dados_cliente()***, definida no arquivo ***funcoes.inc***, cujo código é mostrado no Exemplo 9.9 a seguir.

```php
<?php
    session_start();
?>
<!DOCTYPE html>
<html lang="pt-BR">
    <head>
        <meta charset="utf-8">
        <title>IFSP-Car - Cadastro de Clientes</title>
    </head>
    <body>
        <div class="container">
            <?php
                include "funcoes.inc";
                include "cabecalho.inc";
                include "menu.inc";
                if(empty($_POST)){
                    include "form_cliente.inc";
                }else{
                    ler_dados_cliente();
                }
            ?>
        </div>
    </body>
</html>
```

Exemplo 9.7 - Página de cadastro de clientes (form_cadastro_cliente.php).

```php
<article>
    <form action="form_cadastro_cliente.php" method="post">
        <fieldset>
            <legend>Cadastro de cliente</legend>
            <p>
                <label for="nome">Nome:</label>
                <input type="text" name="nome" id="nome" />
                <label for="email">E-mail</label>
                <input type="email" name="email" id="email" />
            </p>
            <p>
                <label for="end">Endereço:</label>
                <input type="text" name="endereco" id="end" />
                <label for="tel">Telefone:</label>
                <input type="text" name="telefone" id="tel" />
            </p>
            <input type="submit" value="Enviar" />
        </fieldset>
    </form>
</article>
```

Exemplo 9.8 - Formulário a ser incluído (form_cliente.inc)

A função ***ler_dados_cliente()*** adiciona os dados recebidos do *array* ***$_POST*** no *array* superglobal ***$_SESSION***, para armazenar os clientes cadastrados durante uma sessão. No fim da operação, a função exibe uma mensagem informando que o cliente foi cadastrado com sucesso.

```php
<?php
    function ler_dados_cliente(){
        $_SESSION["nomesCliente"][] = $_POST["nome"];
        $_SESSION["emailCliente"][] = $_POST["email"];
        $_SESSION["enderecoCliente"][] = $_POST["endereco"];
        $_SESSION["telefoneCliente"][] = $_POST["telefone"];
        echo "<section><h2>Cliente cadastrado com sucesso.</h2></section>";
    }
?>
```

Exemplo 9.9 - Código do arquivo “funcoes.inc”

9.3. Comandos include_once e require_once

Suponha que exista um arquivo PHP denominado ***X.php*** que inclui (comando ***include***) um arquivo denominado ***Y.php***, e ambos necessitam do arquivo ***Z.php***. Os Exemplos 9.10, 9.11 e 9.12 a seguir mostram, respectivamente, o código dos arquivos PHP X, Y e Z. Conforme é mostrado, no arquivo ***X.php***, existe uma função denominada ***testeX()*** que chama as funções ***testeY()*** (***Y.php***) e ***testeZ()*** (***Z.php***). A função ***testeY()***, por sua vez, também chama a função ***testeZ()***.

Este código não é executado corretamente, e o erro produzido será *“PHP Fatal error: Cannot redeclare testeZ()”*. O motivo é que, no arquivo ***X.php***, são incluídos o conteúdo de ***Y.php*** e o conteúdo de ***Z.php***, mas, como ***Y.php*** também inclui ***Z.php***, a função ***testeZ()*** será especificada por duas vezes no script. Em outras palavras, é como se tivéssemos declarado a mesma função por duas vezes. Para resolver este problema, deve-se adotar uma estratégia para garantir que, caso um dado arquivo já tenha sido incluído na execução de um script, ele não seja incluído novamente.

```php
<?php
    include "Y.php";
    include "Z.php";
    function testeX() {
        echo "Estou em X<br />";
        testeY();
        testeZ();
    }
    testeX();
?>
```

Exemplo 9.10 - Código do arquivo X.php.

```php
<?php
    include "Z.php";
    function testeY() {
        echo "Estou em Y<br />";
        testeZ();
    }
?>
```

Exemplo 9.11 - Código do arquivo Y.php.

```php
<?php
    function testeZ() {
        echo "Estou em Z<br />";
    }
?>
```

Exemplo 9.12 - Código do arquivo Z.php.

Em PHP, existe um comando que permite incluir um dado arquivo somente se ele não tiver sido incluído na execução do script. Este é o comando ***include_once***. O comportamento deste comando é similar ao comportamento do comando ***include***, exceto pelo fato de que, se o código de um dado arquivo já foi incluído, este não será incluído novamente. Assim, a utilização deste comando permitirá evitar problemas como redefinição de funções, conforme a situação anterior ilustrou. Para corrigir o código, as instruções ***include "Z.php"*** devem ser trocadas por ***include_once "Z.php"***, conforme é mostrado nos Exemplos 9.13 e 9.14.

```php
<?php
    include "Y.php";
    include_once "Z.php";
    //algum código aqui
?>
```

Exemplo 9.13 - Código do arquivo X.php utilizando ***include_once.***

```php
<?php
    include_once "Z.php";
    //algum código aqui
?>
```

Exemplo 9.14 - Código do arquivo Y.php utilizando ***include_once.***

Com base na correção, poderíamos pensar se seria necessário especificar o comando ***include_once*** tanto no arquivo ***X.php*** (Exemplo 9.13) quanto no arquivo ***Y.php*** (Exemplo 9.14). De fato, se, no arquivo ***X.php***, é incluído primeiramente o arquivo ***Y.php***, e depois o arquivo ***Z.php***, para que o script seja executado corretamente, o comando ***include_once*** poderia ser utilizado apenas em ***X.php*** (***Y.php*** incluirá primeiramente, e o arquivo não será mais incluído). Entretanto, se, no arquivo ***X.php***, a ordem de inclusão for invertida, isto é, se o arquivo ***X.php*** incluir o arquivo ***Z.php*** e depois o arquivo ***Y.php***, então o comando ***include_once*** deverá ser utilizado em ***Y.php***. Para que a corretude da solução não dependa da ordem de inclusão dos arquivos, utilizamos o comando ***include_once*** em ambos os arquivos.

Da mesma forma que a linguagem PHP disponibiliza o comando ***include_once*** para garantir que um dado arquivo não seja incluído por mais de uma vez no mesmo script, existe o comando ***require_once***, utilizado quando o script exige um determinado arquivo e se deve garantir que o arquivo seja exigido somente por uma vez. Portanto, o comportamento deste comando é similar ao comportamento do comando ***require***, exceto pelo fato de que, se o código de um dado arquivo já foi incluído (exigido), este não será incluído novamente. Note que os comandos ***include_once*** e ***require_once*** somente incluirão/exigirão um arquivo se ele não tiver sido incluído por meio de qualquer um dos seguintes comandos: ***include***, ***require***, ***include_once*** e ***require_once***.

9.4. Exercícios propostos

Exercício 1. Com base no site de locadora de carros definido na Seção 9.2., faça o programa PHP para o cadastro de carros (***form_cadastro_carro.php*** presente no menu), de forma similar ao cadastro de clientes. Devem ser incluídos os códigos HTML do cabeçalho e do menu e o código do arquivo ***funcoes.inc***, que deve conter uma função chamada ***ler_dados_carro()*** para cadastrar os dados dos carros na sessão. Os dados de cadastro dos carros são: marca, modelo, ano, placa, valor da diária em reais e estado (valor 0 para disponível e valor 1 para locado). Decida quais arquivos devem ser incluídos por meio do comando include e quais devem ser exigidos por meio do comando require.

Exercício 2. A partir do exercício anterior, faça um programa em PHP para o cadastro de locações (***form_locacao_carros.php*** presente no menu), de forma similar ao cadastro de clientes e de carros (inclua todos os arquivos necessários). No arquivo ***funcoes.inc***, implemente uma função chamada ***ler_dados_locacao()*** para cadastrar os dados das locações na sessão. Os dados de cadastro das locações são: data da locação, uma lista para selecionar um cliente dentre os clientes já cadastrados, uma lista para selecionar um carro dentre os carros já cadastrados e disponíveis, e um campo para inserir a quantidade de diárias.

Capítulo 10. Manipulação de arquivos

Uma das maneiras de armazenar dados com o objetivo de recuperá-los posteriormente é trabalhando com o sistema de arquivos do sistema operacional. O PHP permite abrir, fechar, ler, escrever e realizar outras funções sobre um arquivo de formato conhecido. Para tanto, não é necessária a instalação de um SGBD (Sistema Gerenciador de Banco de Dados) no servidor.

Começamos o capítulo apresentando exemplos de dados sendo gravados em arquivos no formato texto. Vale lembrar que a manipulação desses arquivos é recomendada somente quando o volume de dados é pequeno.

10.1. Funções de manipulação de arquivos texto

As principais funções de manipulação de arquivos são: abertura, fechamento, leitura, gravação e escrita.

10.1.1. Abertura – fopen()

O primeiro passo para se ler ou gravar um arquivo é abri-lo, usando a função ***fopen()***, que possui a seguinte sintaxe:

```
bool fopen(string nome_arquivo, string modo_acesso)
```

O parâmetro **nome_arquivo** pode referenciar um arquivo local ou um arquivo em um computador remoto. Se a abertura do arquivo falhar, a função retorna **FALSE**.

O segundo parâmetro se refere ao modo de acesso ao arquivo referenciado. Os principais modos são listados na Tabela 10.1.

É uma boa prática verificar se o arquivo existe usando a função ***file_exists()*** antes da abertura do arquivo no modo de leitura (***"r"*** ou ***"r+"***). O Exemplo 10.1 apresenta a utilização da função de abertura de arquivo, juntamente com a função que verifica se um arquivo existe.

```php
<?php
  if(file_exists("./dados/produtos.txt")){
    $arquivo1 = fopen ("./dados/produtos.txt", "r");
  }else{
    echo "Erro ao abrir o arquivo para leitura.";
  }
  $arquivo2 = fopen ("./dados/compras.txt", "w");
  $arquivo3 = fopen ("./dados/clientes.txt", "a");
?>
```

Exemplo 10.1. Abertura de arquivo texto.

Note que os caminhos de arquivo no Exemplo 10.1 começam com um ponto, que indica para o PHP abrir o arquivo a partir da pasta atual, a pasta do arquivo ***.php*** que está solicitando a abertura de arquivo.

Tabela 10.1 - Modos de acesso ao arquivo.

Modo	**Descrição**
'r'	Abre somente para leitura, colocando o ponteiro no início do arquivo.
'r+'	Abre para leitura e escrita, colocando o ponteiro no início do arquivo.
'w'	Abre somente para escrita, colocando o ponteiro no início do arquivo, deixando-o com tamanho zero. Se o arquivo não existir, tentará criá-lo.
'w+'	Abre para leitura e escrita, colocando o ponteiro no início do arquivo, deixando-o com tamanho zero. Se o arquivo não existir, tentará criá-lo.
'a'	Abre somente para escrita, colocando o ponteiro no final do arquivo. Se o arquivo não existir, tentará criá-lo.
'a+'	Abre para leitura e escrita, colocando o ponteiro no final do arquivo. Se o arquivo não existir, tentará criá-lo.

10.1.2. Fechamento – fclose()

Para fechar um arquivo, utiliza-se a função ***fclose()***, que possui a seguinte sintaxe:

bool fclose($arquivo)

A função retorna **TRUE** se o arquivo foi fechado com sucesso e retorna false se houver alguma falha O parâmetro é o nome da variável para qual foi atribuído a referência do arquivo aberto. O Exemplo 10.2 mostra o fechamento do arquivo, após o seu uso.

```php
<?php
   $arquivo = fopen ("./dados/produtos.txt", "r");
   ...
   fclose($arquivo);
?>
```

Exemplo 10.2. Fechamento de arquivo texto.

10.1.3. Leitura - fread()

Com o arquivo aberto, utiliza-se a função ***fread()*** para obter seus dados, conforme a seguinte sintaxe:

string fread(arquivo, int tamanho)

Essa função percorre o arquivo e lê o número de bytes especificado no parâmetro tamanho. A leitura termina quando o número de bytes especificado é lido ou o fim de arquivo é alcançado. Exemplo 10.3 apresenta a leitura em arquivo texto.

```php
<?php
  if(file_exists("./dados/produtos.txt")){
    $arquivo = fopen("./dados/produtos.txt", "r");
    $conteudo = fread($arquivo, 40);
    echo $conteudo;
    fclose($arquivo);
  }else{
    echo "Erro ao abrir o arquivo.";
  }
?>
```

Exemplo 10.3. Leitura em arquivo texto.

O exemplo acima lê os primeiros 40 bytes do arquivo ***"./dados/produtos.txt"*** e os armazena na variável **$conteudo**. Em seguida o valor obtido é exibido na tela e o arquivo é fechado.

10.1.4. Leitura linha a linha - fgets()

É possível realizar a leitura de um arquivo obtendo uma linha de cada vez. Uma das maneiras de se obter uma linha de um arquivo é utilizando a função fgets():

string fgets(nome_arquivo, int tamanho_da_linha)

A leitura é feita seguindo a quantidade de bytes especificado em **tamanho_da_linha** ou quando a linha terminar (**\n**). O valor padrão do tamanho da linha é 1024 bytes. O Exemplo 10.4 ilustra o uso da função de leitura de linhas em arquivo texto.

```php
<?php
  if(file_exists("./dados/produtos.txt")){
    $arquivo = fopen("./dados/produtos.txt", "r");
    $linha1 = fgets($arquivo, 200);
    echo $linha1;
    $linha2 = fgets($arquivo);
    echo $linha2;
    fclose($arquivo);
  }else{
    echo "Erro ao abrir o arquivo.";
  }
?>
```

Exemplo 10.4. Leitura de linhas em arquivo texto.

10.1.5. Escrita - fwrite()

Para gravar dados em um arquivo, utiliza-se a função ***fwrite***(), depois de ter aberto o arquivo. A sintaxe é apresentada a seguir:

int fwrite(nome_arquivo, string conteúdo)

Essa função escreve o conteúdo especificado na variável **nome_arquivo**. O Exemplo 10.5 ilustra o uso da função para escrita em arquivo texto.

```php
<?php
   $nome = "Produto 1";
   $estoque = 100;
   $preco = 29.99;
   $arquivo = fopen("./dados/produtos.txt", "w");

   fwrite($arquivo,$nome . "\n");
   fwrite($arquivo,$estoque . "\n");
   fwrite($arquivo,$preco. "\n");

   fclose($arquivo);
?>
```

Exemplo 10.5. Escrita de linhas em arquivo texto.

Note que o código acima tenta abrir o arquivo ***"./dados/produtos.txt"*** no modo de escrita, sobrescrevendo seu conteúdo e gravando o texto contido nas variáveis $nome, $estoque e $preco. Se o arquivo não existir, tentará criá-lo e, em seguida, o arquivo é fechado.

10.2. Juntando tudo!

O Exemplo 10.6 apresenta um código completo para leitura e escrito em arquivo texto.

```php
<?php
  $nome1 = "Produto 1";
  $estoque1 = 100;
  $preco1 = 29.99;
  $nome2 = "Produto 2";
  $estoque2 = 200;
  $preco2 = 49.99;

  $arquivo = fopen("./dados/produtos.txt", "w");

  fwrite($arquivo,$nome1 . "\n");
  fwrite($arquivo,$estoque1 . "\n");
  fwrite($arquivo,$preco1 . "\n");
  fwrite($arquivo,$nome2 . "\n");
  fwrite($arquivo,$estoque2 . "\n");
  fwrite($arquivo,$preco2 . "\n");

  fclose($arquivo);

  $arquivo = fopen("./dados/produtos.txt", "r");

  while(!feof($arquivo)){
    $nome = fgets($arquivo);
    $estoque = fgets($arquivo);
    $preco = fgets($arquivo);
    if(!empty($nome)){
      echo "<p>Nome: $nome - Estoque: $estoque - Preço: $preco</p>";
    }
  }

  fclose($arquivo);
?>
```

Exemplo 10.6. Escrita e leitura de linhas em arquivo texto.

No Exemplo 10.6, o arquivo ***"./dados/produtos.txt"*** é aberto com atributo ***"w"*** (aberto para escrita) e em seguida são escritos os dados de 2 produtos (nome, estoque e quantidade), cada dado em uma linha de texto. Em seguida o mesmo arquivo é aberto, com atributo ***"r"*** (aberto para leitura) e seu conteúdo é exibido na tela. Note que a função ***feof()*** é utilizada para identificar se o final do arquivo foi atingido.

É importante ressaltar que é preciso ter permissão de acesso aos arquivos, seja na pasta local ou no servidor Web.

10.3. Manipulação de arquivos no formato JSON

O formato JSON (**J**ava**S**cript **O**bject **N**otation) é baseado em JavaScript, mas é independente de linguagens e plataformas. JSON foi criado por Douglas CrockFord em 1999 e foi formalizado pela RFC 4627 (IETF) em 2006.

JSON tem sido amplamente adotado na indústria de desenvolvimento e permite representar com mais naturalidade certas estruturas como objetos e *arrays*.

A sintaxe JSON é idêntica à sintaxe utilizada para criar objetos em JavaScript, como mostra o Exemplo 10.7.

```
1  [
2    {
3      "codigo": 1,
4      "nome": "Produto 1",
5      "estoque": "100",
6      "preco": "29.99"
7    }
8  ]
```

Exemplo 10.7. Sintaxe do formato JSON.

Como mostra o Exemplo 10.7, um texto em JSON é uma sequência de tokens. O conjunto de tokens inclui seis caracteres, *strings*, números, e três literais. Espaços em branco são permitidos entre os tokens. JSON é baseado em duas estruturas: objetos e *arrays*. A Tabela 10.2 apresenta os caracteres reservados do JSON.

Tabela 20.2 – Caracteres reservados do JSON.

Caractere	Significado
[	Início da definição de um *array*.
{	Início da definição de um objeto.
]	Fechamento de um *array*.
}	Fechamento de um objeto.
:	Permite definir um par *nome/valor*.
,	Permite separar um par *nome/valor* de outro par.

Um valor em JSON pode ser um objeto, um *array*, uma *string*, ou um número (inteiro ou decimal). Há três literais que são aceitas como valores: **null**, **false** e **true**. Todas as

literais devem ser escritas em letras minúsculas e todos os valores devem ser escritos entre aspas duplas, com exceção de números e literais.

Um uso comum de JSON é ler dados de um servidor da Web e exibir os dados em uma página da Web. O PHP tem algumas funções integradas para manipulação de dados no formato JSON.

Os objetos em PHP podem ser convertidos em JSON usando a função ***json_encode()***, como mostra o Exemplo 10.8.

```php
<?php
  $produto->codigo = 1;
  $produto->nome = "Produto 1";
  $produto->estoque = 100;
  $produto->preco = 29.99;

  $produto = json_encode($produto);

  echo $produto;
?>
```

Exemplo 10.8. Conversão de objeto PHP em JSON.

Os *arrays* em PHP também são convertidos em JSON ao usar a função ***json_encode()***, como apresenta o Exemplo 10.9.

```php
<?php
  $produto1->codigo = 1;
  $produto1->nome = "Produto 1";
  $produto1->estoque = 100;
  $produto1->preco = 29.99;

  $produto2->codigo = 2;
  $produto2->nome = "Produto 2";
  $produto2->estoque = 200;
  $produto2->preco = 49.99;

  $produtos = array($produto1,$produto2);

  $produtos = json_encode($produtos);

  echo $produtos;
?>
```

Exemplo 10.9. Conversão de array PHP em JSON.

10.4. Juntando tudo!

Aqui apresenta-se um exemplo para inserção, edição, remoção e listagem de dados de produtos armazenados em arquivos no formato JSON.

10.4.1. Inserção de dados

O arquivo ***produto_cadastro.php***, Exemplo 10.10, inclui o arquivo de funções ***funcoes.php*** e verifica se os dados do produto foram enviados.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <title>Formulário de Cadastro de Produto</title>
</head>
<body>
  <?php
    require_once "funcoes.php";
    if(empty($_POST)){
      include "produto_form_cadastro.html";
    }else{
      cadastrarProduto();
    }
  ?>
</body>
</html>
```

Exemplo 10.10. Código do arquivo produto_cadastro.php.

Se os dados ainda não foram enviados (a superglobal **$_POST** está vazia), o arquivo ***produto_form_cadastro.html***, Exemplo 10.11, é apresentado ao usuário:

```
<form action="produto_cadastro.php" method="POST">
  <fieldset>
    <legend>Informe os dados do produto</legend>
    <p>
      <label>Nome:</label>
      <input type="text" name="nome" />
    </p>
    <p>
      <label>Estoque:</label>
      <input type="number" name="estoque" />
    </p>
    <p>
      <label>Preço:</label>
      <input type="number" step="0.01" name="preco" />
    </p>
  </fieldset>
  <input type="submit" value="Enviar" />
</form>
```

Exemplo 10.11. Código do arquivo produto_form_cadastro.html.

Se os dados já foram enviados (a superglobal **$_POST** está cheia), a função ***cadastrarProduto()*** do arquivo ***funcoes.php*** é chamada, como apresenta o Exemplo 10.12.

```
18  function cadastrarProduto(){
19    $codigo = obterCodigo();
20
21    $produto = array(
22      "codigo" => $codigo,
23      "nome" => $_POST["nome"],
24      "estoque" => $_POST["estoque"],
25      "preco" => $_POST["preco"]
26    );
27
28    if(file_exists("produtos.json")){
29      $dados = file_get_contents("produtos.json");
30      $dados = json_decode($dados, true);
31    }
32    $dados[] = $produto;
33    $dados = json_encode($dados, JSON_PRETTY_PRINT);
34    file_put_contents("produtos.json", $dados);
35
36    atualizarCodigo($codigo);
37
38    echo "<h2>Produto cadastrado com sucesso!</h2>";
39    echo "<p><a href='produto_cadastro.php'>Voltar</a></p>";
40    echo "<p><a href='produto_listagem.php'>Listagem de Produtos</a></p>";
41  }
```

Exemplo 10.12. Código da função cadastrarProduto().

Na linha 19, a função ***obterCodigo()*** é chamada para gerar o código do produto, sequência de números crescente mantida no arquivo ***codigos.json***. Se o arquivo existir, o código é recuperado usando inicialmente a função ***file_get_contents()*** para abrir o obter os dados do arquivo e depois decodificar os dados usando a função ***json_decode()***, como apresenta o Exemplo 10.13.

```
 2  function obterCodigo(){
 3    if(file_exists("codigos.json")){
 4      $codigo = file_get_contents("codigos.json");
 5      $codigo = json_decode($codigo);
 6    }else{
 7      $codigo = 1;
 8    }
 9    return $codigo;
10  }
```

Exemplo 10.13. Código da função obterCodigo().

Após obter o código, os dados do produto são recuperados da superglobal **$_POST** e armazenados no *array* **$produto**. Então, se o arquivo ***produtos.json*** existir, é solicitada

a abertura e a decodificação dos dados em formato JSON, que são armazenados no *array* **$dados**. Os dados do novo produto não adicionados ao *array* **$dados**, codificados novamente no formato JSON e armazenados no arquivo ***produtos.json***, com uso da função ***file_put_contents()***. Na sequência, a função ***atualizarCodigo()*** é chamada para incrementar o valor do código atual, como mostra o Exemplo 10.14.

```
12  function atualizarCodigo($codigo){
13    $codigo = $codigo + 1;
14    $codigo = json_encode($codigo);
15    file_put_contents("codigos.json", $codigo);
16  }
```

Exemplo 10.14. Código da função atualizarCodigo().

Por fim, uma mensagem de sucesso é apresentada ao usuário na tela do navegador.

10.4.2. Listagem de dados

O arquivo ***produto_listagem.php***, Exemplo 10.15, verifica se existe algum produto cadastrado (arquivo ***produtos.json***) e, se existir, o arquivo ***produto_tabela.php*** é incluído. Do contrário, uma mensagem é apresentada para informar a não existência de dados de produtos.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4    <meta charset="UTF-8">
5    <title>Listagem de produtos</title>
6  </head>
7  <body>
8    <?php
9      if(file_exists("produtos.json")){
10       include "produto_tabela.php";
11       echo "<p><a href='produto_cadastro.php'>Cadastrar Produto</a></p>";
12     }else{
13       echo "<h2>Nenhum produto cadastrado</h2>";
14       echo "<a href='produto_cadastro.php'>Cadastrar Produtos</a>";
15     }
16   ?>
17 </body>
18 </html>
```

Exemplo 10.15. Código do arquivo produto_listagem.php.

No arquivo ***produto_tarefa.php***, Exemplo 10.16, o cabeçalho da tabela HTML e montado, o arquivo de ***funcoes.php*** é incluído e a função ***listarProdutos()*** é chamada.

```
<table border="1">
  <thead>
    <tr>
      <th>Código</th>
      <th>Nome</th>
      <th>Estoque</th>
      <th>Preço</th>
      <th>Ações</th>
    </tr>
  </thead>
  <tbody>
  <?php
    require_once "funcoes.php";
    listarProdutos();
  ?>
  </tbody>
</table>
```

Exemplo 10.16. Código do arquivo produto_tabela.php.

O Exemplo 10.17 apresenta o código da função ***listarProdutos()***, que abre o arquivo JSON com os dados dos produtos, decodifica os dados e coloca os dados em linhas da tabela HTML. Note que os hiperlinks para edição e exclusão de cada produto são incluídos na tabela.

```
function listarProdutos(){
  $produtos = file_get_contents("produtos.json");
  $produtos = json_decode($produtos, true);
  $encontrou = false;
  foreach($produtos as $produto){
    if(!empty($produto)){
      echo '
        <tr>
          <td>'. $produto["codigo"] .'</td>
          <td>'. $produto["nome"] .'</td>
          <td>'. $produto["estoque"] .'</td>
          <td>'. $produto["preco"] .'</td>
          <td>
            <a href="produto_edicao.php?codigo_produto='. $produto["codigo"] .'">Editar</a>
            <a href="produto_exclusao.php?codigo_produto='. $produto["codigo"] .'">Excluir</a>
          </td>
        </tr>
      ';
      $encontrou = true;
    }
  }
  if(!$encontrou){
    echo '
          <tr>
            <td colspan="5">Nenhum produto cadastrado</td>';
  }
}
```

Exemplo 10.17. Código da função listarProdutos().

10.4.3. Edição de dados

O arquivo ***produto_edicao.php***, Exemplo 10.18, inclui o arquivo de funções ***funcoes.php*** e verifica se os dados editados do produto foram enviados.

```php
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <title>Formulário de Edição de Produto</title>
</head>
<body>
  <?php
    require_once "funcoes.php";
    if(empty($_POST)){
      include "produto_form_edicao.php";
    }else{
      editarProduto();
    }
  ?>
</body>
</html>
```

Exemplo 10.18. Código do arquivo produto_edicao.php.

Se os dados ainda não foram enviados (a superglobal **$_POST** está vazia), o arquivo ***produto_form_edicao.php***, Exemplo 10.19, é apresentado ao usuário:

```php
<?php
  if(!empty($_GET)){
    $codigo_produto = $_GET["codigo_produto"];
    $produto = buscarProdutoPeloCodigo($codigo_produto);
  }
?>
<form action="produto_edicao.php" method="POST">
  <fieldset>
    <legend>Edição dos dados do produto</legend>
    <p>
      <label>Código:</label>
      <input type="text" name="codigo" value="<?=$produto->codigo?>" readonly />
    </p>
    <p>
      <label>Nome:</label>
      <input type="text" name="nome" value="<?=$produto->nome?>" />
    </p>
    <p>
      <label>Estoque:</label>
      <input type="number" name="estoque" value="<?=$produto->estoque?>" />
    </p>
    <p>
      <label>Preço:</label>
      <input type="number" step="0.01" name="preco" value="<?=$produto->preco?>" />
    </p>
  </fieldset>
  <input type="submit" value="Enviar" />
</form>
```

Exemplo 10.19. Código do arquivo produto_form_edicao.php.

O código do produto que será editado é recuperado da superglobal **$_GET** e os dados do produto são buscados com a chamada da função ***buscarProdutoPeloCodigo()***, como mostra o Exemplo 10.20.

```php
function buscarProdutoPeloCodigo($codigo_produto){
  $produtos = file_get_contents("produtos.json");
  $produtos = json_decode($produtos, true);
  foreach($produtos as $produto){
    if($produto->codigo == $codigo_produto){
      return $produto;
    }
  }
}
```

Exemplo 10.20. Código da função buscarProdutoPeloCodigo().

Os dados do produto são carregados nos campos de texto do formulário HTML e, ao submeter o formulário com os dados editados, a função ***editarProduto()*** é chamada, Exemplo 10.21.

```php
function editarProduto(){
  $dados = file_get_contents("produtos.json");
  $dados = json_decode($dados, true);

  $produto = array(
    "codigo" => $_POST["codigo"],
    "nome" => $_POST["nome"],
    "estoque" => $_POST["estoque"],
    "preco" => $_POST["preco"]
  );

  foreach($dados as $indice=>$prod){
    if($prod["codigo"] == $_POST["codigo"]){
      $dados[$indice] = $produto;
    }
  }

  $dados = json_encode($dados, JSON_PRETTY_PRINT);
  file_put_contents("produtos.json", $dados);

  echo "<h2>Produto editado com sucesso!</h2>";
  echo "<p><a href='produto_listagem.php'>Listagem de Produtos</a></p>";
}
```

Exemplo 10.21. Código da função editarProduto().

10.4.4. Exclusão de dados

O arquivo ***produto_exclusao.php***, Exemplo 10.22, inclui o arquivo de funções ***funcoes.php*** e verifica se o código do produto foi enviado por meio da superglobal **$_GET**.

```php
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <title>Formulário de Cadastro de Produto</title>
</head>
<body>
  <?php
    require_once "funcoes.php";
    if(!empty($_GET)){
      excluirProduto();
    }
  ?>
</body>
</html>
```

Exemplo 10.22. Código do arquivo produto_exclusao.php.

Se o código do produto foi enviado, a função ***excluirProduto()***, Exemplo 10.23, é chamada para efetuar a exclusão dos dados do produto.

```php
function excluirProduto(){
  $dados = file_get_contents("produtos.json");
  $dados = json_decode($dados, true);

  foreach($dados as $indice=>$prod){
    if($prod["codigo"] == $_GET["codigo_produto"]){
      unset($dados[$indice]);
    }
  }

  $dados = json_encode($dados, JSON_PRETTY_PRINT);
  file_put_contents("produtos.json", $dados);

  echo "<h2>Produto excluído com sucesso!</h2>";
  echo "<p><a href='produto_listagem.php'>Listagem de Produtos</a></p>";
}
```

Exemplo 10.23. Código da função excluirProduto().

10.5. Exercícios propostos

Exercício 1. Crie uma página web, com uso da linguagem PHP, que contém formulário (***<form>***) para que o usuário configure a apresentação visual da página de boas-vindas, gerando um arquivo CSS para:

a. Aplicar uma cor de fundo ao corpo da página.

b. Aplicar um tamanho de fonte, uma cor e um alinhamento ao texto (direita, esquerda, centralizado) dos parágrafos.

c. Aplicar um alinhamento e uma cor de texto ao texto dos <h1>.

Exercício 2. Crie uma página web, com uso da linguagem PHP, que contém formulário para entrada de dados de livros (título, autor, ano de publicação, número de páginas e editora). Os dados inseridos no formulário deverão ser salvos em um arquivo texto. Crie também uma página para apresentar os dados dos livros armazenados no arquivo.

Exercício 3. Crie uma página web que contém formulário para entrada de dados de alunos (nome, nota da primeira prova, nota da segunda prova). Os dados inseridos no formulário deverão ser salvos em um arquivo no formato JSON. Crie também uma página para apresentar os dados dos alunos armazenados no arquivo, a média de notas de cada aluno e a média de notas da turma.

Exercício 4. Sistema de Vendas de Produtos. Faça um programa em PHP que atua como um sistema de vendas de produtos, que deve gerenciar basicamente as vendas de produtos de um supermercado. Assim, crie (utilize funções, arquivos ***".inc"*** – include, e arquivos texto):

- Uma página inicial (***index.php***) com um cabeçalho (nome do supermercado), um menu de opções e um rodapé com o contato do supermercado (endereço, telefone, e-mail).
  Uma página de cadastro de clientes (***form_cadastro_cliente.php***) para receber os dados de um cliente (nome, endereço, telefone, e-mail e CPF) e armazene-os em um arquivo no formato JSON ***(clientes.json***). Após armazenar os dados, o programa deve apresentar uma mensagem de sucesso.
- Uma página de cadastro de produtos (***form_cadastro_produto.php***) para receber os dados de um produto (código, descrição e preço) e armazene-os em um arquivo no formato JSON **(produtos.json)**. Após armazenar os dados, o programa deve apresentar uma mensagem de sucesso.
- Uma página de venda de produtos (***form_venda_produtos.php***) para receber os dados de uma venda (data da venda, cliente – previamente cadastrado, produto – previamente cadastrado, e quantidade vendida. Com base no preço do produto e quantidade vendida, o programa deve calcular o valor total da venda. Os dados da venda devem ser armazenados em um arquivo no formato JSON (***vendas.txt***). Após armazenar os dados, o programa deve apresentar uma mensagem de sucesso.

- Uma página de listagem de clientes (***lista_clientes.php***) para gerar uma tabela com os dados de todos os clientes cadastrados.
- Uma página de listagem de produtos (***lista_produtos.php***) para gerar uma tabela com os dados de todos os produtos cadastrados.
- Uma página de listagem de vendas (***lista_vendas.php***) para gerar uma tabela com os dados de todas as vendas realizadas.

Capítulo 11. Acesso a Banco de Dados

O phpMyAdmin é uma ferramenta de software livre escrita em PHP para manipular os SGBDs MySQL e MariaDB (versão 5.5 ou mais recente) por meio de um navegador. As operações usadas com frequência tais como gerenciamento de bancos de dados, tabelas, colunas, relações, índices, usuários, permissões, entre outras, podem ser realizadas por meio de um navegador. Além disso, é possível executar diretamente qualquer instrução SQL. O phpMyAdmin importa dados de CSV e SQL e exporta dados para vários formatos: CSV, SQL, XML, PDF, texto e planilha OpenDocument, Word, LATEX, entre outros.

11.1. Utilização do phpMyAdmin

O phpMyAdmin está presente na instalação do aplicativo XAMPP. Para inicializá-lo, é preciso clicar no botão “Start” dos módulos Apache e MySQL, conforme ilustra a figura 11.1.

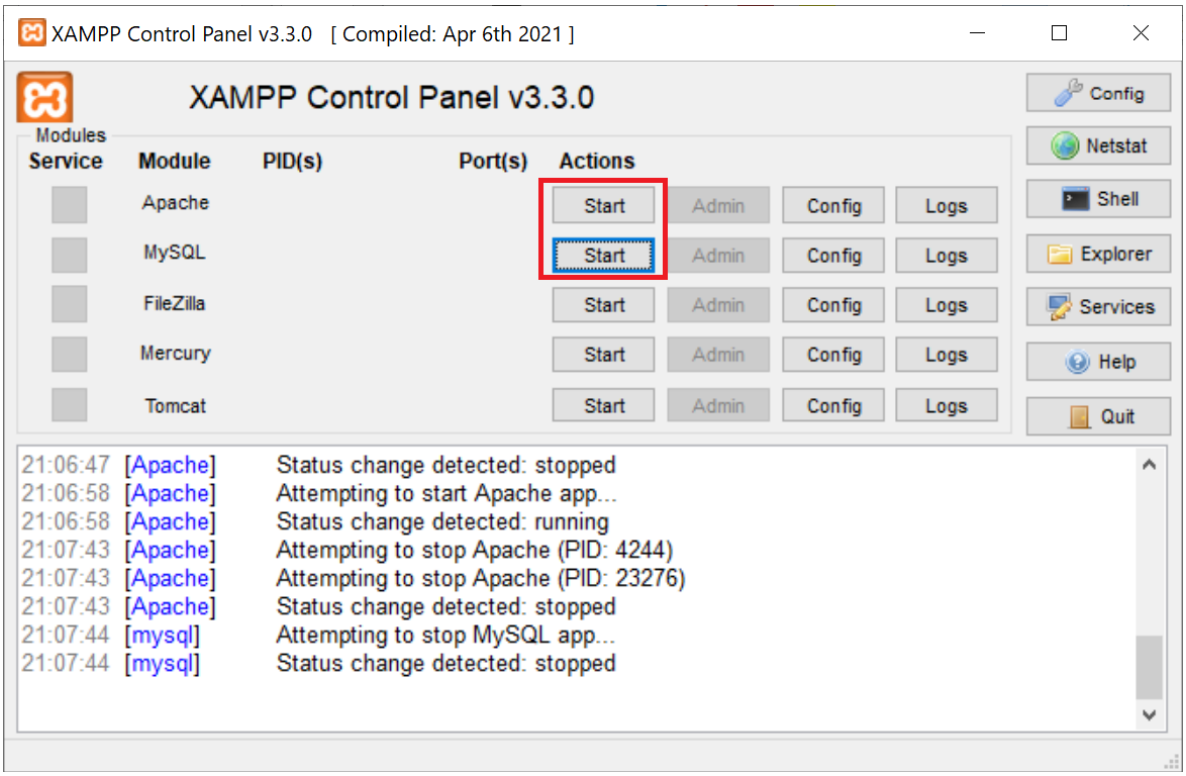


Figura 11.1 – Inicializando o Apache e o PhpMyAdmin.

Em seguida, vá até o browser e digite o URL: http://localhost/phpmyadmin/. Será carregado o phpMyAdmin, conforme ilustra a figura 11.2.

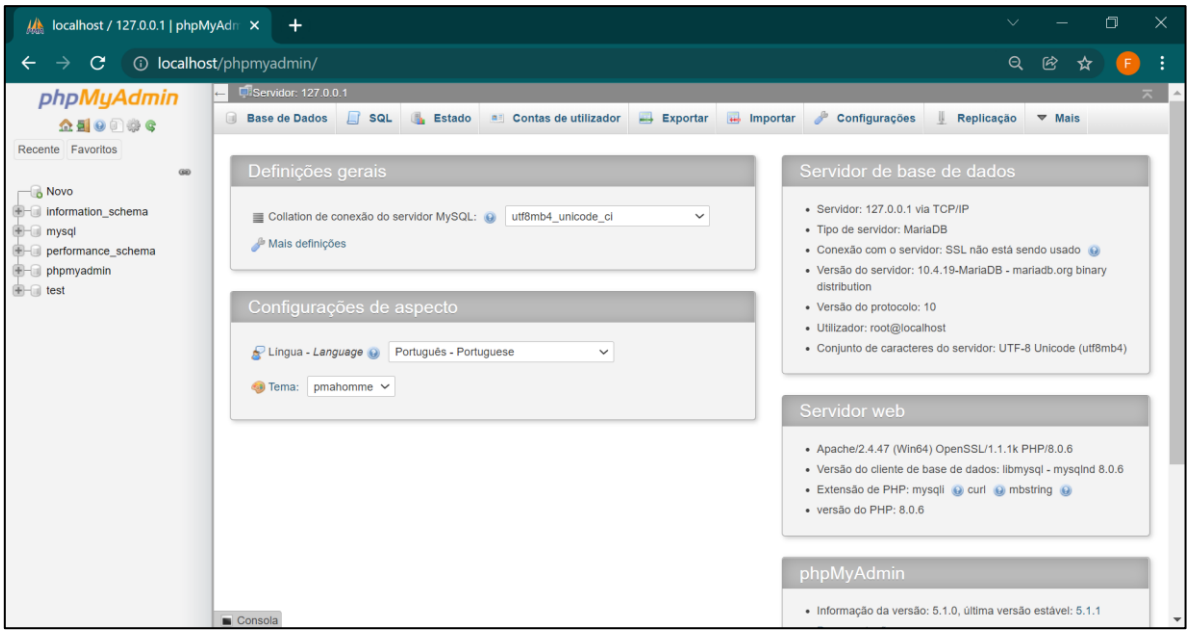


Figura 11.2 – Tela inicial do phpMyAdmin.

No painel à esquerda, é possível criar um banco de dados (botão ‘Novo’) ou escolher entre os bancos de dados de exemplo disponíveis, já instalados por *default* (information_schema, mysql, performance_schema, phpmyadmin e test). No menu de opções, é possível ver a estrutura dos bancos de dados disponíveis, clicando em “Base de Dados”, conforme ilustra a figura 11.3.

Figura 11.3 – Listagem de bancos de dados disponíveis.

O item já preenchido ***‘utf8mb4_general_ci’*** descreve o formato de dados a serem armazenados (indiferente a maiúsculas e minúsculas).

A manipulação de dados de um banco de dados em uma página Web por meio do PHP inclui os seguintes passos:

- Efetuar conexão com o servidor MySQL;
- Selecionar o banco de dados;
- Executar uma consulta SQL (ou adicionar, alterar, excluir registros, entre outras);
- Visualizar os resultados e;
- Encerrar a conexão.

11.2. Formas de acesso ao banco de dados

Existem algumas formas de acesso a um servidor de banco de dados MySQL quando programamos em PHP, a saber, utilizando o MySQLi ou PDO (PHP Data Object – Objeto de Dados PHP).

O MySQLi oferece uma API procedural, o que facilita a compreensão dos novos usuários. A extensão PDO define uma interface para acessar bancos de dados em PHP, fornecendo um acesso aos dados independente do banco de dados que você está usando, usando as mesmas funções para realizar consultas. Na Figura 11.4 são exibidas as duas formas de acesso.

```
// PDO
$pdo = new PDO('mysql:host=localhost;dbname=lojadb','root','password');

// MySQLi
$mysqli = mysqli_connect('localhost', 'username', 'password', 'lojadb');
```

Figura 11.4. Uso de MySQLi e PDO para conexão com o banco de dados.

A principal vantagem do PDO sobre o MySQLi está no suporte aos sistemas gerenciadores de banco de dados. O PDO suporta 12 diferentes tipos de sistemas gerenciadores de banco de dados (incluindo PostgreSQL, IBM, Oracle e MySQL), enquanto o MySQLi suporta apenas MySQL.

Em termos de segurança, tanto o PDO quanto o MySQLi fornecem suporte para consultas preparadas (***prepared statemens***). Isso ajuda a evitar problemas de segurança de injeção de SQL (SQL injection - inserção de códigos impróprios), desde que você use somente consultas preparadas para inserir parâmetros dinâmicos nas consultas.

PDO possui os ***prepared statemens***, que é uma forma da consulta não ser enviada diretamente ao servidor, passando por uma etapa de preparação prévia. Temos uma ilustração de uma consulta preparada na Figura 11.5.

```
<?php
 $sql = $pdo->prepare('SELECT *  FROM clientes  WHERE nome=? AND idade = ?');
 $sql->execute(array('José', 35));
 $resultado = $sql->fetchAll();
?>
```

Figura 11.5. Preparando a consulta antes de enviá-la ao servidor.

Os atributos de busca da cláusula ***WHERE*** são recebidos por ***?***, para que haja a preparação. Depois, a execução recebe os parâmetros reais da ***query***.

Em suma, o MySQLi é um recurso mais antigo e mais desenvolvido, além de ter melhor performance, quando o banco de dados utilizado for o MySQL. Já o PDO proporciona portabilidade entre bancos, além de substituir o MySQLi a partir da versão 7.

11.3. Conexão ao banco de dados

Utilizaremos as conexões por meio da criação de instâncias da classe PDO (PHP Data Object – objeto de dados PHP). A extensão PDO define uma interface para acessar bancos de dados em PHP, fornecendo um acesso aos dados independente do banco de dados que você está usando, usando as mesmas funções para realizar consultas. Para trabalhar com um banco de dados em uma página web, é necessário criar uma variável que contenha uma conexão com o servidor MySQL, conforme sintaxe abaixo:

```
$<nome_var> = new PDO(“<nome_do_servidor>”, “nome_bd”, “usuário”, “senha”);
```

Por exemplo:

```
$pdo = new PDO('mysql:host=localhost;dbname=lojadb','root','');
```

Onde:

- **$pdo** é a variável que contém a conexão com o servidor MySQL;
- **PDO** é uma função do PHP para abrir uma conexão com o servidor;
- ***localhost*** é o nome do servidor local;
- ***root*** é o nome do usuário do banco de dados e;
- ***admin*** é a senha do usuário root.

11.4. Execução de consultas SQL

Após a conexão com o banco de dados, o próximo passo é criar uma variável que contém as instruções SQL para manipulação dos dados das tabelas do banco de dados, conforme sintaxe abaixo:

```
$<nome_var_sql> = "<instrução SQL>";
$<var_resultado_sql> = $<var_pdo> -> prepare(<nome_var_sql>);
$<var_resultado_sql> ->execute();
```

Por exemplo:

```
$consulta = "SELECT * FROM produtos";
$sql = $pdo->prepare($consulta);
$sql ->execute();
```

Onde:

- **$consulta** é a variável que contém a instrução SQL;
- **$pdo** é a variável que contém a conexão com o servidor MySQL e;
- **$sql** é o nome da variável que contém o resultado da instrução SQL.

O exemplo acima seleciona todos os registros da tabela ***"produtos"***.

11.5. Manipulação dos resultados da consulta

O próximo passo incluir a criação de uma variável responsável por armazenar o resultado da execução da instrução SQL, de forma que possa ser exibido pelo navegador, conforme sintaxe abaixo:

```
$<nome_var_resultado> = $<nome_var_sql> -> fetchAll();
```

Por exemplo:

```
$resultado = $sql->fetchAll();
```

Onde:

- **$sql** é o nome da variável que contém a instrução SQL;
- **$resultado** é a variável que contém o resultado da pesquisa SQL (*array*) e;
- ***fetchAll()*** é uma função que retorna um *array* com todas as linhas da consulta.

Na sequência, é preciso formatar o resultado obtido pela consulta. Para a exibição correta dos campos de uma tabela, é possível separar seus registros por linha, utilizando a função ***fetchAll()*** e o laço de repetição ***foreach***, conforme a sintaxe abaixo:

```
foreach ($<nome_var_resultado> as $<chave> -> $<valores>) {
    echo 'Valor1 : '.$value['valor1'];
    echo '<br />';
    echo 'Valor2: '.$value['valor2'];
    echo '<hr>';
}
```

Por exemplo:

```
$resultado = $sql->fetchAll();
foreach ($resultado as $key => $coluna) {
    echo 'Código: '.$coluna['codigo'];
    echo '<br />';
    echo 'Nome: '.$coluna['nome'];
    echo '<hr>';
}
```

Onde:

- **$resultado** é a variável que contém o resultado da pesquisa SQL (*array*);
- **$key** é a variável que contém o ponteiro do *array* e;
- **$coluna** é a variável que contém as colunas da tabela.

11.6. Encerramento da conexão

O último passo se refere ao encerramento da conexão com uma tabela de um banco de dados. Quando terminar de utilizar uma tabela, deve-se fechá-la e encerrar a conexão com o MySQL, seguindo a sintaxe abaixo:

```
$pdo = null;
```

Observação: não é necessário pois a conexão é fechada automaticamente após a execução do script.

11.7. Juntando tudo!

Até aqui, foram apresentados diversos conceitos básicos para o acesso e manipulação de dados de um banco de dados utilizando a linguagem PHP. O exemplo descrito nessa seção visa juntar os conteúdos já vistos e aplicá-los em um caso prático.

Primeiramente, vamos criar um banco de dados para estudar a interface do phpMyAdmin. O nome do banco de dados será "***lojadb***". Em seguida, vamos criar a tabela "***produtos***", conforme é apresentado na Tabela 11.1.

Tabela 11.1 – Estrutura da tabela "produtos".

Nome do Campo	Tipo	Tamanho	Chave primária
Código	int	5	Sim
Descrição	varchar	50	Não
Marca	varchar	50	Não
Preço	float		Não

Em seguida, apresenta-se um exemplo para inserção, edição, remoção e listagem de dados de produtos armazenados na tabela "***produtos***" no banco de dados "***lojadb***".

11.7.1. Inserção de dados

Vamos escrever a página ***form_inserir.html*** que contém um formulário para cadastro de um novo produto, conforme o Exemplo 11.1.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
      <title>Formulário inserir produto</title>
  </head>
  <body>
    <form action="cadastrar.php" method="post">
      <fieldset>
        <legend>Digite as informações do produto</legend>
        <fieldset>
          <p>
            <label>Descrição</label>
            <input type="text" name="descricao" />
          </p>
          <p>
            <label>Marca</label>
            <input type="text" name="marca" />
          </p>
          <p>
            <label>Preço</label>
            <input type="number" step="0.01" name="preco" />
          </p>
      </fieldset>
      <p>
        <input type="submit" name="inserir" value="Enviar" />
      </p>
    </form>
  </body>
</html>
```

Exemplo 11.1 – Página form_inserir.html.

O arquivo ***cadastrar.php***, Exemplo 11.2, realiza o cadastro do produto com as informações recebidas do formulário da página ***form_inserir.html***.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <title>Inserindo produto</title>
6    </head>
7
8    <?php
9      //conectando com o banco de dados:
10     $pdo = new PDO('mysql:host=localhost;dbname=lojadb','root','');
11
12     //testando os dados recebidos do formulário:
13     if (isset($_POST["inserir"])) {
14       $codigo = rand(1,1000); //código aleatório
15       $descricao = $_POST["descricao"];
16       $marca = $_POST["marca"];
17       $preco = $_POST["preco"];
18
19       //escrevendo instrução SQL:
20       $sql = $pdo->prepare("INSERT INTO `produtos` VALUES (?,?,?,?) ");
21
22       //executando instrução SQL:
23       $sql->execute(array($codigo,$descricao,$marca,$preco));
24       echo '<h2>Produto cadastrado!</h2>';
25     }
26   ?>
27 </html>
```

Exemplo 11.2 – Página cadastrar.php.

Note que os dados dos campos passados como parâmetros são inseridos num *array* (linha 23) e então a consulta é executada, impedindo a inserção de dados indevidos diretamente na instrução SQL.

11.7.2. Listagem de dados

Vamos escrever o código que exibe todos os dados da tabela ***"produtos"*** a partir de uma consulta SQL. Os resultados são exibidos no formato de tabela HTML, conforme e exemplo 11.3.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <title>Produtos do banco de dados loja</title>
6    </head>
7
8    <?php
9      //conectando com o banco de dados:
10     $pdo = new PDO('mysql:host=localhost;dbname=lojadb','root','');
11     //escrevendo instrução SQL:
12     $consulta = "SELECT * FROM produtos";
13     //executando instrução SQL:
14
15     $sql = $pdo->prepare($consulta);
16     $sql->execute();
17     //testando consulta SQL:
18     //if(!$sql->execute()){
19     //  echo 'Erro';
20     //}
21   ?>
22
23   <body>
24     <h2>Produtos em estoque</h2>
25     <table cellspacing="0" border="1" width="100%">
26       <tr>
27         <td>Código</td>
28         <td>Descrição</td>
29         <td>Marca</td>
30         <td>Preço</td>
31       </tr>
32
33       <?php
34
35     //criando array com o resultado obtido na consulta:
36     $reg = $sql->fetchAll();
37
38     foreach ($reg as $key => $coluna) {
39       echo '<tr>';
40       echo '<td>'.$coluna['codigo'].'</td>';
41       echo '<td>'.$coluna['descricao'].'</td>';
42       echo '<td>'.$coluna['marca'].'</td>';
43       echo '<td>'.$coluna['preco'].'</td>';
44       echo '</tr>';
45     }
46     $pdo = null;
47     ?>
48     </table>
49   </body>
50 </html>
```

Exemplo 11.3 – Página listar.php.

A conexão com o banco de dados “***lojadb***” é realizada na linha 10. Note que são especificados o servidor “***localhost***” e o usuário “***root***”, sem senha. A variável **$consulta**

contém a instrução SQL responsável por retornar todos os registros da tabela “***produtos***”. Entre as linhas 17 e 20 é possível verificar se houve um erro na execução da consulta.

A linha 36 é criado um *array* **$reg** que contém o resultado obtido na consulta. Em seguida, é criada uma tabela em HTML com os títulos “***código***”, “***descrição***”, “***marca***” e “***preço***”, informações obtidas da tabela “***produtos***” do banco de dados “***lojadb***”, por meio da estrutura ***foreach*.**

11.7.3. Edição de dados

Vamos escrever o código que altera os dados da tabela “***produtos***” a partir de uma consulta SQL. Primeiramente, vamos escrever a página ***form_editar.html*** para obter os dados que serão atualizados, conforme o Exemplo 11.4.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5        <title>Formulário editar produto</title>
6    </head>
7    <body>
8      <form action="editar.php" method="post">
9        <fieldset>
10         <legend>Digite as informações do produto</legend>
11         <fieldset>
12           <p>
13             <label>Código</label>
14             <input type="text" name="codigo" />
15           </p>
16           <p>
17             <label>Descrição</label>
18             <input type="text" name="descricao" />
19           </p>
20           <p>
21             <label>Marca</label>
22             <input type="text" name="marca" />
23           </p>
24           <p>
25             <label>Preço</label>
26             <input type="number" step="0.01" name="preco" />
27           </p>
28       </fieldset>
29       <p>
30         <input type="submit" name="editar" value="Enviar" />
31       </p>
32     </form>
33   </body>
34 </html>
```

Exemplo 11.4 – Página form_editar.html.

Esta página contém um formulário nomeado “***editar***” e carrega a página “***editar.php***” quando clicamos em “***Enviar***”. É pertinente frisar que somente será possível editar os dados

de um produto já existente (código já existente). O Exemplo 11.5 ilustra a página ***editar.php***.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <title>Editando produto</title>
6    </head>
7
8    <?php
9      //conectando com o banco de dados:
10     $pdo = new PDO('mysql:host=localhost;dbname=lojadb','root','');
11
12     //testando os dados recebidos do formulário:
13     if (isset($_POST["editar"])) {
14       $codigo = $_POST["codigo"];
15       $descricao = $_POST["descricao"];
16       $marca = $_POST["marca"];
17       $preco = $_POST["preco"];
18
19       //escrevendo instrução SQL:
20       $sql = $pdo->prepare("UPDATE `produtos` SET descricao=?,marca=?,preco=? WHERE codigo=?");
21
22       //testando/executando instrução SQL:
23       if(!$sql->execute(array($descricao,$marca,$preco,$codigo))){
24         echo 'Produto atualizado!';
25       }
26     }
27   ?>
28 </html>
```

Exemplo 11.5 – Página editar.php.

Primeiramente, os dados recebidos são testados (linha 13). É importante frisar a escrita da condição ***WHERE*** da consulta SQL (linha 20), uma vez que somente será possível editar um produto já existente. Além disso, o código é protegido contra instruções SQL impróprias inserindo-se os caracteres ***'?'*** na instrução SQL (linha 20). Estes caracteres são substituídos por um *array* com os dados das variáveis (na ordem em que aparecem) na instrução ***UPDATE*** quando a instrução é executada (linha 23).

11.7.4. Remoção de dados

Vamos escrever o código que apaga os dados da tabela ***"produtos"*** a partir de uma consulta SQL. Primeiramente, vamos escrever a página ***form_apagar.html*** para obter os dados que serão removidos, conforme o Exemplo 11.6.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
      <title>Formulário apagar produto</title>
  </head>
  <body>
    <form action="apagar.php" method="post">
      <fieldset>
        <legend>Digite as informações do produto</legend>
        <fieldset>
          <p>
            <label>Código</label>
            <input type="text" name="codigo" />
          </p>
          <p>
            <label>Descrição</label>
            <input type="text" name="descricao" />
          </p>
          <p>
            <label>Marca</label>
            <input type="text" name="marca" />
          </p>
          <p>
            <label>Preço</label>
            <input type="number" step="0.01" name="preco" />
          </p>
      </fieldset>
      <p>
        <input type="submit" name="editar" value="Enviar" />
      </p>
    </form>
  </body>
</html>
```

Exemplo 11.6 – Página form_apagar.html

Esta página contém um formulário nomeado "***apagar***" e carrega a página "***apagar.php***" quando clicamos em "***Enviar***". É pertinente frisar que somente será possível remover os dados de um produto já existente (código já existente). O Exemplo 11.7 contém o código da página ***apagar.php***.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <title>Apagando produto</title>
6    </head>
7
8    <?php
9      //conectando com o banco de dados:
10     $pdo = new PDO('mysql:host=localhost;dbname=lojadb','root','');
11
12     //testando os dados recebidos do formulário:
13     if (isset($_POST["editar"])) {
14       $codigo = $_POST["codigo"];
15       $descricao = $_POST["descricao"];
16       $marca = $_POST["marca"];
17       $preco = $_POST["preco"];
18
19       //escrevendo instrução SQL:
20       $sql = $pdo->prepare("DELETE FROM `produtos` WHERE codigo=?");
21
22       //testando/executando instrução SQL:
23       if(!$sql->execute(array($codigo))){
24         echo 'Produto apagado!';
25       }
26     }
27   ?>
28 </html>
```

Exemplo 7.7. Página apagar.php

Primeiramente, os dados recebidos são testados (linha 13). É importante frisar a escrita da condição ***WHERE*** da consulta SQL (linha 21), uma vez que somente será possível editar um produto já existente. Além disso, o código é protegido contra instruções SQL impróprias inserindo-se o caractere ***'?'*** na instrução SQL (linha 20). Este caractere é substituído por um *array* com os dados das variáveis (na ordem em que aparecem) na instrução ***DELETE*** quando a instrução é executada (linha 20).

11.8. Exercícios propostos

Exercício 1. Faça um programa em PHP que faça uma consulta no banco de dados ***"lojadb"***, na tabela ***"produtos"*** listando todas as informações dos produtos ordenadas pelo preço de cada mercadoria (mais baratos primeiro).

Exercício 2. Faça um programa em PHP que faça uma consulta no banco de dados ***"lojadb"***, na tabela ***"produtos"*** listando todas as informações dos produtos ordenadas pela marca de cada mercadoria (ordem alfabética).

Exercício 3. Faça um programa em PHP que faça uma alteração de um registro no banco de dados ***"lojadb"***, na tabela ***"produtos"*** primeiramente listando todos os códigos dos produtos existentes.

Exercício 4. Faça um programa em PHP que faça uma exclusão de um registro no banco de dados ***"lojadb"***, na tabela ***"produtos"*** primeiramente listando todos os códigos dos produtos existentes.

Referências Bibliográficas

LIMA JR, L. E. de **Um pouco da história da linguagem de programação PHP**. Disponível em: <http://www.naninho.blog.br/web/php/um-pouco-da-historia-da-linguagem-de-programacao-php.html>. Acesso em: 26 mar. 2018.

MILANI, A. **Construindo Aplicações Web com PHP e MySQL**. 2 ed. São Paulo: Novatec, 2010.

NIEDERAUER, J. **Desenvolvendo Websites com PHP.** 3. ed. São Paulo: Novatec, 2017.

WATANABE, W. M. **Comunicação cliente/servidor - HTTP**. Disponível em: <https://pt.slideshare.net/watinha1/apresentacao-17018075>. Acesso em: 20 mar. 2018.