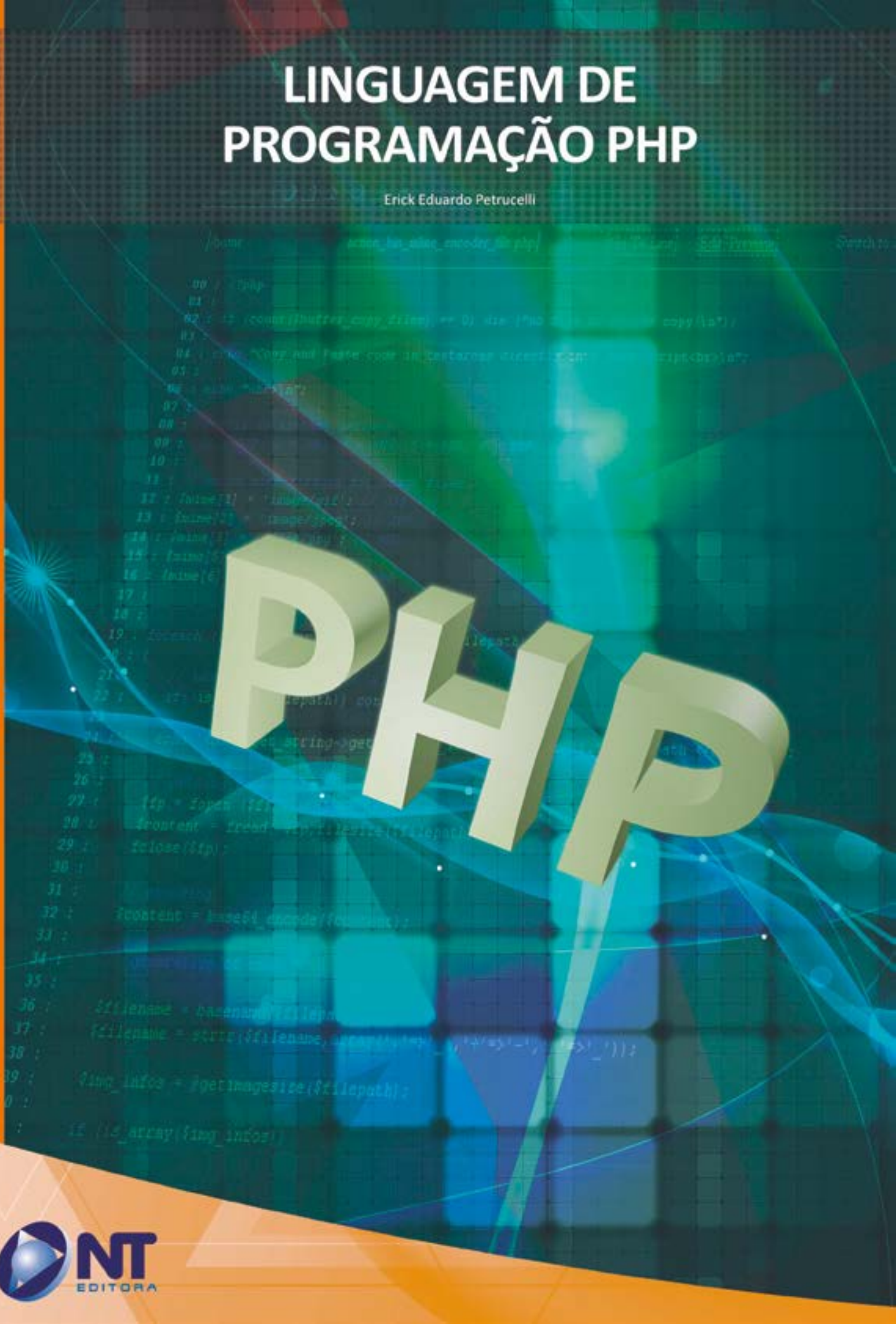
LINGUAGEM DE PROGRAMAÇÃO PHP

Erick Eduardo Petrucelli



INFORMAÇÃO E COMUNICAÇÃO

NT EDITORA

LINGUAGEM DE PROGRAMAÇÃO PHP

Erick Eduardo Petrucelli

INFORMAÇÃO E COMUNICAÇÃO

NT EDITORA

Autor

Erick Eduardo Petrucelli

Mestre em Engenharia de Produção, MBA em Gestão de Tecnologia da Informação, Tecnólogo em Processamento de Dados, Certificado MCP e MCPD. Atua com desenvolvimento de *software* há mais de 20 anos, principalmente com as tecnologias HTML, CSS, JavaScript, Node.js, .NET, PHP e SQL. Professor de Ensino Superior há 10 anos. Coordenador do Curso Superior de Tecnologia em Sistemas para Internet da Faculdade de Tecnologia (Fatec) de Taquaritinga/SP. Eventualmente, palestra sobre desenvolvimento *Web* e sobre mercado de TI. Desenvolvedor por vocação, nas horas vagas se mantém envolvido com o ecossistema *open-source* de desenvolvimento *Web* (principalmente com a plataforma *Web*, Vue.js e Node.js).

Design Instrucional
NT Editora

Projeto Gráfico
NT Editora

Revisão
Valesca Fonseca

Capa
NT Editora

Editoração Eletrônica
Talitta Uchôa

Ilustração
Guilherme Gusmão

NT Editora, uma empresa do Grupo NT
SCS Quadra 2 – Bl. C – 4° andar – Ed. Cedro II
CEP 70.302-914 – Brasília – DF
Fone: (61) 3421-9200
sac@grupont.com.br
www.nteditora.com.br e www.grupont.com.br

Petrucelli, Erick Eduardo.
Linguagem de Programação PHP / Erick Eduardo Petrucelli – 1. ed. – Brasília: NT Editora, 2019.
248 p. il. ; 21,0 X 29,7 cm.

ISBN 978-85-8416-696-1

1. PHP. 2. *Web*.
I. Título

ÍCONES

Prezado(a) aluno(a),
Ao longo dos seus estudos, você encontrará alguns ícones na coluna lateral do material didático. A presença desses ícones o(a) ajudará a compreender melhor o conteúdo abordado e a fazer os exercícios propostos. Conheça os ícones logo abaixo:

Saiba mais
Esse ícone apontará para informações complementares sobre o assunto que você está estudando. Serão curiosidades, temas afins ou exemplos do cotidiano que o ajudarão a fixar o conteúdo estudado.

Importante
O conteúdo indicado com esse ícone tem bastante importância para seus estudos. Leia com atenção e, tendo dúvida, pergunte ao seu tutor.

Dicas
Esse ícone apresenta dicas de estudo.

Exercícios
Toda vez que você vir o ícone de exercícios, responda às questões propostas.

Exercícios
Ao final das lições, você deverá responder aos exercícios no seu livro.

Bons estudos!

Sumário

1 INICIANDO NO ECOSSISTEMA PHP 9
1.1 Conhecendo o PHP 9
1.2 Funcionamento dos servidores *Web* 13
1.3 Escolhendo um servidor local 18
1.4 Configurando o ambiente de desenvolvimento 22
1.5 Habilitando a depuração do código-fonte 26

2 O BÁSICO DA LINGUAGEM 35
2.1 Características gerais da linguagem 35
2.2 Tipos, variáveis, constantes e *arrays* 40
2.3 Operandos e operadores 45
2.4 Estruturas condicionais 50
2.5 Estruturas de repetição 55

3 ORGANIZAÇÃO DOS PROJETOS 63
3.1 Importação entre arquivos 63
3.2 Organização através de funções 68
3.3 Tipagem e tratamento de exceções 73
3.4 Gerenciamento de pacotes 78
3.5 Documentação avançada do código 83

4 ORIENTAÇÃO A OBJETOS 92
4.1 Conceitos básicos da orientação a objetos 92
4.2 Classes, propriedades, métodos e construtores 97
4.3 Estendendo objetos através da herança 102
4.4 Organização do código com *namespaces* 106
4.5 Recursos adicionais de orientação a objetos 112

5 INTERAGINDO COM O CLIENTE 121
5.1 Requisições e formulários 121
5.2 Interação com HTML e JavaScript 127
5.3 Validação e confiabilidade dos dados 133
5.4 Submissão e tratamento de arquivos 139
5.5 Utilizando *cookies* e sessão 145

6 INTEGRAÇÃO COM BANCOS DE DADOS 153
6.1 Primeiros passos com bancos de dados 153
6.2 Criação de tabelas e relacionamentos 158
6.3 Inserindo novos registros nas tabelas 162
6.4 Consultando, excluindo e atualizando registros 167
6.5 Assuntos adicionais sobre a integração 172

7 OUTRAS FUNCIONALIDADES RELEVANTES 182
7.1 Trabalhando com dados não-relacionais 182
7.2 Construção de suas próprias APIs REST 187
7.3 Consumo de APIs e envio de *e-mails* 192
7.4 Segurança, criptografia e autenticação 197
7.5 Expressões regulares e internacionalização 203

8 UTILIZAÇÃO DE *FRAMEWORKS* *212*
8.1 Visão geral do CakePHP 212
8.2 Visão geral do Symfony 219
8.3 Visão geral do Yii 224
8.4 Visão geral do Laravel 229
8.5 Outros *frameworks* conhecidos 235

GLOSSÁRIO 245

BIBLIOGRAFIA 247

APRESENTAÇÃO

Caro(a) estudante,

Seja bem-vindo à **Linguagem de Programação PHP!**

Não sabemos mais viver sem *internet*, seja para comunicação em tempo real, aplicações corporativas, comércio eletrônico, redes sociais, filmes, músicas, jogos, notícias etc. Muito além da camada visual que todos os usuários enxergam e interagem enquanto navegam, essas maravilhas modernas dependem de alguma – ou muita – programação executada nos servidores em que estão hospedadas, através das chamadas linguagens de programação *server-side* ou *back-end*.

***Back-end*:** termo em inglês sem tradução direta, utilizado na área de software para designar a camada de uma aplicação responsável pelo processamento dos dados.

Linguagens de programação executadas nos servidores *Web* não são novidade e, atualmente, dispomos de uma considerável gama para escolher. Entre aquelas criadas especificamente para o desenvolvimento *Web*, temos a *PHP* (*Hypertext Preprocessor*), ou simplesmente PHP, como uma das primeiras que surgiram, uma das mais conhecidas, uma das que mais evoluíram ao longo dos anos e, claro, uma das mais utilizadas e com forte apelo no mercado de trabalho.

Considerada por muitos uma linguagem fácil de aprender, a PHP oferece muita flexibilidade. Por outro lado, isso também é motivo para críticas, com desenvolvedores reclamando que a linguagem parece uma "colcha de retalhos" na qual coisas foram sendo "remendadas" ao longo do tempo sem muito planejamento sobre o que viria a existir. Parte das críticas podem até ser justas, mas, ao mesmo tempo, é claro que os criadores da linguagem não poderiam ter previsto muitas tendências e paradigmas de programação modernos que só vieram a ser tendências anos depois. E, de qualquer forma, é ótimo ver a linguagem em constante evolução, abraçando características bem-sucedidas de outras linguagens sem perder sua essência: a praticidade.

O fato é que quase todos os desenvolvedores *Web* profissionais já tiveram algum contato com PHP, de modo que é fácil encontrar muito código defasado em relação aos recursos mais modernos da linguagem. Neste material, vamos enfrentar tal desafio de conhecer a linguagem de forma aprofundada, compreender as formas mais tradicionais de se trabalhar com ela e, claro, as principais tendências de desenvolvimento moderno com PHP.

Bons estudos!

Erick Eduardo Petrucelli

1 INICIANDO NO ECOSSISTEMA PHP

Olá! Está preparado para se aventurar no ecossistema da linguagem de programação PHP? Então, vamos começar!

Vamos iniciar conhecendo de forma geral o PHP, sua história e suas principais características. Abordaremos um pouco sobre o funcionamento dos servidores *Web* e como o PHP se situa neste contexto. Então, trataremos da escolha e da configuração de um servidor local para desenvolvimento. Em seguida, discutiremos o que fazer para preparar seu computador como um ambiente de desenvolvimento PHP para os estudos e as atividades práticas. Ao final, abordaremos também a configuração dos componentes necessários para permitir a depuração dos códigos-fontes que criaremos, permitindo identificar e corrigir erros com maior produtividade e profissionalismo.

Objetivos

Ao finalizar esta lição, você deverá ser capaz de:

- conhecer a história e as características gerais do PHP;
- entender o funcionamento geral dos servidores *Web*;
- configurar a execução de um servidor *Web* localmente;
- preparar seu computador como um ambiente de desenvolvimento PHP;
- compreender os requisitos para a depuração do código-fonte.

1.1 Conhecendo o PHP

Como dito na apresentação inicial, a linguagem de programação *Hypertext Preprocessor*, comumente chamada apenas de PHP, é uma das mais antigas, famosas e utilizadas linguagens de programação para o desenvolvimento *server-side*.

Server-Side: termo em inglês para "lado do servidor", o qual costuma ser empregado na *Web* com o mesmo significado que *back-end*.

Trata-se de uma linguagem de programação específica, inspirada principalmente nas linguagens preexistentes C e *Pearl*, e recebe posteriormente forte influência também da linguagem Java, sendo constituída principalmente por:

- algumas regras de sintaxe, ou seja, de que maneira os comandos devem ser escritos, metaforicamente similares às regras gramaticais de um idioma;
- um conjunto de palavras reservadas, ou seja, palavras-chave (*keywords*) que denotam alguma ação especial durante a execução da linguagem e que não podem ser utilizadas pelo desenvolvedor para outros fins;
- um núcleo extenso de tipos de dados, funções, objetos e classes, algo que pode ser encarado como conjuntos de códigos que já foram desenvolvidos para as mais diversas necessidades e que estão embutidos na linguagem, os quais você terá à sua disposição para resolver os problemas que desejar.

Saiba mais

A linguagem PHP em si é um projeto livre de código aberto, o que costumamos chamar de *open source*. Representa bem o esteriótipo deste tipo de projeto: criada para atender necessidades não atendidas dos próprios desenvolvedores e refinada ao longo do tempo em prol de sua crescente comunidade. Atualmente, o projeto está hospedado em um repositório público no endereço <https://github.com/php>.

Open Source: refere-se ao conceito de manter público o código-fonte de uma solução, normalmente com vistas a envolver uma comunidade em torno de seu desenvolvimento.

Resumo da história do PHP

O PHP foi criado originalmente no ano de 1994, por Rasmus Lerdorf, como um conjunto simples de *scripts* utilitários escritos em linguagem C para servirem como binários CGI para seu próprio *site* pessoal. Por isso, ele chamou esse conjunto de *scripts* de *Personal Home Page Tools* (ferramentas de página pessoal) e seu objetivo inicial era bem restrito: substituir alguns *scripts* em *Pearl* que ele utilizava para contar o número de visitantes.

CGI: *Common Gateway Interface*, é a tecnologia fundamental para geração de páginas dinâmicas, permitindo a interação do servidor *Web* com outros programas executados neste sistema.

Durante alguns meses, Rasmus reescreveu o PHP *Tools*, como começou a chamar para simplificar, e passou a oferecer uma implementação muito maior e mais rica, capaz inclusive de oferecer conexão e interação com bancos de dados. Assim, finalmente, em 1995, lançou o código-fonte para o público, encorajando outros desenvolvedores a utilizarem e, inclusive, a colaborarem com correção de *bugs* e melhorias diversas.

Após este período inicial, que acabou sendo considerado o equivalente à versão 1.0 da linguagem PHP, tivemos marcos importantes em sua evolução:

- **PHP/FI 2.0**. Por um curto período de 1995, Rasmus desistiu do nome PHP por uma nova implementação denominada FI, por causa de *Form Interpreter* (interpretador de formulários), já que o foco seria oferecer mecanismos para trabalhar com os dados submetidos ao servidor a partir de formulários HTML. Em 1996, após reescrever completamente o projeto, o nome PHP foi reincorporado à sigla FI, tornando-se PHP/FI. Após passar um bom tempo em versão beta, foi oficialmente lançado em 1997. Naquele ponto, não era mais um conjunto de *scripts* utilitários, pois havia ganhado efetivamente todas as características de uma linguagem de programação, com sintaxe e palavras-chave.
- **PHP 3.** Em 1997, Zeev Suraski e Andi Gutmans decidiram iniciar outro projeto de reconstrução da linguagem ao considerarem a atual ineficiente e incompleta para um projeto de

e-commerce. A reconstrução foi abraçada por Rasmus e prosseguida pelo trio. Considerando que estavam criando uma linguagem totalmente nova, decidiram renomear novamente para PHP com uma nova conotação, o acrônimo recursivo PHP: *Hypertext Preprocessor*. Lançada em 1998, esta versão se tornou um marco e é a primeira que se aproxima do PHP moderno, oferecendo diversos dos recursos atuais.

- **PHP 4**. Em 1998, novamente Zeev e Andi começaram a reescrever as coisas, mas, desta vez, focaram no núcleo da linguagem, principalmente em desempenho e aplicações complexas. Este trabalho resultou em um novo motor de execução da linguagem lançado em 1999, nomeado de *Zend Engine* (uma junção do início dos nomes dos dois criadores). Para acompanhar a evolução, em 2000 ocorreu o lançamento da versão 4.0 da linguagem, incluindo suporte a mais servidores, controle de sessão HTTP e *buffer* de saída.
- **PHP 5**. Lançada em 2004, após um longo período em desenvolvimento e vários lançamentos preliminares, esta versão da linguagem veio acompanhada da nova *Zend Engine* 2.0 e, entre várias melhorias e novos recursos, ofereceu um novo modelo reescrito de orientação a objetos mais alinhado às tendências de mercado e inspirado principalmente na linguagem Java.
- **PHP 6**. O trabalho nesta versão iniciou de forma concomitante com o trabalho evolutivo que veio a ser tornar o PHP 5.2 e 5.3, de 2006 e 2009, respectivamente, e almejava principalmente permitir que a linguagem suportasse uma tabela de caracteres mais ampla, a tabela UTF-16. Na prática, esse conjunto amplo de caracteres não é normalmente utilizado na *Web* e as implicações de desempenho da mudança estavam difíceis de serem superadas. Ao invés de uma grande reformulação, mudanças menores foram sendo in-troduzidas nos versionamentos do PHP 5 citados e, dessa forma, a versão 6.0 da linguagem nunca foi efetivamente lançada.
- **PHP 7.** Apesar dos frequentes versionamentos menores, como a versão 5.4 em 2012, a versão 5.5, em 2013, e a versão 5.6, em 2015, o longo período sem uma versão grande e o descrédito causado pelo não lançado 6.0 chegou a minar um pouco a reputação da linguagem. Como um sopro de renovação, em 2015, chegou o PHP 7 e os versionamentos menores têm ocorrido quase que anualmente até o 7.3. Além de um desempenho renovado que surpreendeu a comunidade, trouxe também melhorias em diversas áreas, novas funções e reforço em recursos quanto à orientação a objetos.

HTTP: *Hypertext Transfer Protocol*, é um protocolo fundamental da *Web* que estabelece as regras de comunicação entre um navegador e um servidor *Web*.

Dica

No momento desta publicação, a quantidade de desenvolvedores usando a versão 5 ainda é superior à versão 7. Para dados mais atualizados, dê uma olhada em <https://w3techs.com/technologies/details/pl-php/all/all>. De qualquer forma, não escolha uma versão por que a maioria está usando. Pelo contrário, aproveite a oportunidade para estar à frente da maioria!

Principais características da linguagem

Podemos dizer que o maior intuito do PHP é proporcionar a implementação de soluções com eficiência, tanto quanto à produtividade para criá-las quanto ao desempenho em execução. Entre as principais características dessa linguagem, podemos destacar:

- **Prática**: o PHP surgiu para solucionar problemas específicos de desenvolvedores *Web*. É uma linguagem prática que resulta na criação de aplicações úteis que necessitam de pouco conhecimento. Essa praticidade se confirma quando observamos algumas características, como: não necessita de inclusão de bibliotecas obrigatórias; é capaz de aninhar

chamadas de funções; não exige gerenciar explicitamente a memória; simplifica o trabalho com tipos de dados, oferecendo conversões automáticas por contexto; proporciona se concentrar no objetivo final e não em detalhes de funcionamento.

- **Ampla**: a linguagem apresenta centenas de funções em seu núcleo, além de milhares de bibliotecas à disposição dos desenvolvedores com um quantitativo considerável de funcionalidades possíveis. Não que outras linguagens não permitam obter as mesmas funcionalidades, masm no PHPm muita coisa é nativa ou é baseada em bibliotecas adicionais muito maduras. É possível, por exemplo, gerar imagens nos mais diversos formatos; criar e manipular arquivos Word, Excel, PDF, entre outros; avaliar senhas quanto à dificuldade de adivinhação em comparação a dicionários de senhas padrão ou à composição destas; analisar cadeias de caracteres complexas a partir das bibliotecas de expressões regulares baseadas em Perl e POSIX; autenticar usuários em relação a credenciais armazenadas em arquivos simples, *Active Directory* da Microsoft ou bancos de dados; comunicar usando uma ampla variedade de protocolos (como HTTP, POP3, IMAP, LDAP, DNS); integrar consistentemente a amplas soluções para o processamento de pagamentos, incluindo cartão de crédito.
- **Flexível**: o desenvolvedor está bem servido de diversas alternativas para soluções corriqueiras. Só para exemplificar com um tema, a manipulação de bancos de dados suporta nativamente mais de 25 produtos, incluindo dBase, FilePro, Firebird, FrontBase, IBM DB2, Informix, Ingres, InterBase, MariaDB, Microsoft SQL Server, MongoDB, MySQL, Oracle, PostgreSQL, SQLite e Sybase. Na verdade, as camadas de abstração DBA, ODBC e PDO são flexíveis a ponto de suportar mais sem nem registrar na documentação.
- **Livre**: não há custos diretos com a linguagem, não há restrição de uso, de modificações ou redistribuição. A linguagem PHP é livre de restrições de licenciamento impostas em muitos projetos, pois possui uma licença exclusiva que garante total liberdade (apenas o nome PHP não pode ser utilizado em outras situações e qualquer *software* que o utilize precisa explicitamente incluir a informação que o utiliza em sua própria licença ou documentação). Além disso, como um bom *software* livre, é totalmente aberto a contribuições diretamente em seu repositório oficial.
- **Consolidada**: talvez por apresentar todas as características anteriores, o PHP se tornou a base de diversas soluções muito utilizadas, o que acabou por se tornar uma característica positiva por si só. Alguns exemplos famosos: o sistema de fóruns phpBB, os sistemas de *e-commerce* Magento e PrestaShop, os sistemas de gestão de conteúdo e *blogs* Drupal, Joomla e WordPress, este último é um dos projetos mais famosos e utilizados de toda a Internet!

***Software*:** termo utilizado para designar genericamente conjuntos de instruções que definem como um computador ou dispositivo eletrônico deve funcionar.

Com tantas características animadoras, vamos prosseguir explorando o ambiente em que executaremos normalmente o PHP, os servidores *Web*, antes de efetivamente iniciarmos os conteúdos específicos sobre tal linguagem.

Programando o conhecimento

Qual das opções a seguir não é uma característica da linguagem PHP?

a) Livre.

b) Consolidada.

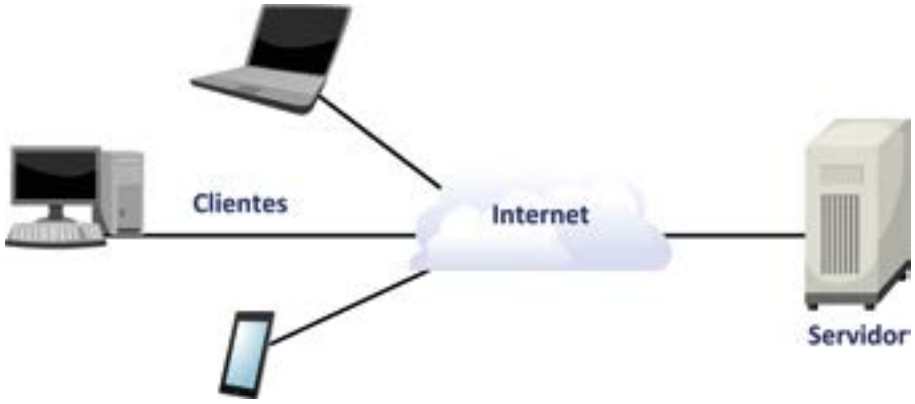
c) Ampla.

d) Proprietária.

Comentário: se você pensou na alternativa "d", parabéns! A linguagem PHP não é uma linguagem proprietária, ou seja, não é detida comercialmente por alguma empresa ou algum indivíduo. As alternativas adicionais apresentam características efetivamente válidas sobre a linguagem.

1.2 Funcionamento dos servidores *Web*

Certamente você já navegou na *internet* diversas vezes, mas, ao decidir navegar em um determinado *site*, por exemplo, o endereço <www.php.net> para ver a página oficial do projeto PHP, os últimos lançamentos e a documentação, como este conteúdo efetivamente chega até seu navegador? Graças aos servidores.



Consideramos um computador como um servidor quando o *hardware* e o *software* são dedicados à execução de tarefas que suportem a entrega de dados solicitados por outros dispositivos, ou seja, outras máquinas que se comunicam eventualmente, agindo como clientes do serviço desejado, em uma relação conhecida como cliente-servidor. No contexto da *internet*, um servidor deve estar conectado de forma constante à rede, possuindo um endereço IP estático e executando constantemente as ferramentas que forem necessárias para servir o que se deseja.

Saiba mais

De forma genérica, eventualmente também chamamos servidor de *host*, que, em sentido literal, significa "hospedeiro", bem adequado quando consideramos que contratamos uma "hospedagem" para um *site*. Ainda em nomenclaturas, a solicitação aos arquivos realizada pelos clientes costuma ser chamada de *request* (requisição), e o resultado devolvido pelo *host* costuma ser chamado de *response* (resposta).

Tipos de servidores na *internet*

Para que a *internet* funcione como estamos acostumados, dependemos de vários tipos de servidores, dentre os quais podemos citar:

- **servidores *Web***: armazenam arquivos que compõem um ou mais *sites* ali hospedados, os quais devem ser disponibilizados aos visitantes que solicitarem. Operam se comunicando através do protocolo HTTP ou HTTPS;

***Hardware*:** termo utilizado para designar as partes físicas de um computador ou dispositivo eletrônico.

IP: *Internet Protocol*, utilizado para o endereçamento durante a comunicação, por isso normalmente se utiliza o termo "endereço IP" para designar o número de identificação de um determinado dispositivo em uma rede.

HTTPS: *Hypertext Transfer Protocol Secure*, é uma variante do protocolo HTTP tradicional, com uma camada adicional de segurança que segue outro protocolo, o SSL/TLS.

- **Servidores** DNS: servidores de apoio à localização de outros servidores, possuem endereços IP estáticos que já ficam registrados nos equipamentos que farão a conexão ou entregues aos clientes pelos provedores de *internet*, durante a conexão inicial. A partir deles, a conexão pode ser direcionada ao IP correto correspondente ao *site* que estiver sendo acessado;
- **Servidores de banco de dados**: normalmente não são acessados diretamente pelos usuários finais, mas sim pelos servidores *Web* que precisem consumir dados antes de entregar as respostas. De fato, não precisam ser obrigatoriamente máquinas separadas, servidores *Web* podem ter os bancos de dados ali mesmo, caso o desempenho seja viável;
- **Servidores CDN**: redes de fornecimento de conteúdos (*Content Delivery Networks*) são conjuntos de servidores interligados cooperando de modo transparente para fornecer conteúdos, normalmente mídias e outros conteúdos estáticos, com um desempenho superior, comumente através de vários servidores espalhados por diversos locais, contendo réplicas fiéis dos dados armazenados, os quais responderão às requisições que estiverem mais próximas geograficamente, reduzindo o tráfego dos dados pela rede;
- **Outros tipos de servidores**: a *internet* ainda se baseia em diversos outros tipos de servidores, como servidores de *e-mail,* através dos protocolos IMAP, SMTP e POP3; servidores de arquivos, através do protocolo FTP; servidores de *games*, os quais podem ter seus próprios protocolos proprietários; servidores *proxy*, que atuam redirecionando conexões, por questões de desempenho ou de privacidade; entre outros tipos.

DNS: *Domain Name System*, é um sistema distribuído de vários servidores que mantém tabelas de mapeamento entre nomes de domínio (os endereços dos sites) e o verdadeiro endereço numérico (IP) do servidor onde estão efetivamente hospedados.

SMTP: *Simple Mail Transfer Protocol*, é o protocolo padronizado para envio de *e-mails* através da Internet com base em texto plano contendo os destinatários e a mensagem a ser transferida.

Evolução dos servidores *Web*

Desde que o protocolo HTTP foi estabelecido, os servidores *Web* têm trabalhado para servir todos os *site*s existentes na *internet*, mas a forma como isso ocorre evoluiu com o tempo, sendo possível classificar os servidores *Web* em gerações.

A **primeira geração** surgiu há aproximadamente 30 anos com o início da *Web*, perdurando até por volta dos anos 2000. Trata-se da entrega de arquivos completamente estáticos aos clientes. Conceitualmente, esses primeiros servidores *Web* eram quase iguais aos servidores de arquivos FTP, com a diferença de que operavam por meio do protocolo HTTP. O ciclo de vida de cada requisição era bem curto, pois era atendida apenas localizando-se o arquivo desejado pelo cliente e enviando em seguida, sem nenhum processamento adicional. Ainda que simples, lembre-se que as máquinas eram mais primitivas e a separação entre servidores e computadores domésticos ainda estava engatinhando. Todo o desempenho estava à mercê do *hardware* da máquina e de sua integridade: bastaria que um cabo fosse acidentalmente desconectado para que a página *Web* ficasse fora do ar.

A **segunda geração** teve início em meados de 1995, sendo que muitos de seus princípios ainda são utilizados hoje. Do ponto de vista de *hardware*, foi quando se iniciou o conceito de dispor os serviços em *data centers*, ambientes projetados para armazenar diversos servidores de forma controlada, idealmente oferecendo redundância: se uma máquina falhasse, uma réplica poderia assumir e manter o serviço no ar. Contudo, a principal revolução da segunda geração foi no *software*: os servidores *Web* passaram a processar os arquivos antes de entregar aos clientes, através de extensões CGI acopladas.

Dessa forma, ao invés de servir apenas conteúdo estático, linguagens de programação passaram a ser utilizadas para realizar processamento prévio, injetando dados em páginas dinâmicas. A história do surgimento do PHP que contamos está atrelada diretamente a essa geração.

Nos anos 2000, vimos surgir a **terceira geração**, cujo modelo se mantém largamente utilizado até hoje, sendo o marco de tal geração o surgimento dos servidores virtuais. A chegada da virtualização permitiu que os administradores de *data centers* se preocupassem apenas com o poder computacional e a integridade do *hardware* dos servidores, permitindo que cada desenvolvedor tivesse acesso à sua própria "máquina virtual" na hospedagem, podendo configurar *softwares* e, eventualmente, bancos de dados.

Isso agregou valor aos serviços prestados por hospedagens, ao mesmo tempo em que permitiu redução de custos e grande popularização. Neste período, hospedagens mais baratas surgiram (e algumas até gratuitas), a maior parte com foco no ecossistema PHP, enquanto hospedagens para outras plataformas eram normalmente mais caras e menos comuns.

Temos ainda uma **quarta geração,** partindo de meados de 2010 até os tempos atuais. Suas principais características são a escalabilidade e a flexibilidade com o fornecimento de recursos de *hardware* dinâmicos, reservando a quantidade ideal de largura de banda, memória e poder de processamento, podendo ajustar as necessidades rapidamente se necessário (normalmente de forma automática). Funcionam como uma evolução da virtualização, na qual toda a infraestrutura é oferecida como serviço, o que comumente chamamos de computação em nuvem (*cloud computing*). A complexidade do gerenciamento reduziu e esse cenário tem se difundido. Vale ressaltar que todas as grandes plataformas de computação em nuvem suportam PHP nativamente.

Algumas pessoas afirmam que estamos iniciando uma quinta geração, enquanto outras afirmam ser uma derivação da quarta. Com a expansão da adoção da computação em nuvem, as principais grandes plataformas começam a oferecer arquiteturas conhecidas como *serverless*, ironicamente significando "menos servidores", apesar de os servidores continuarem existindo normalmente.

Nesse modelo não há absolutamente nenhum tipo de gerenciamento do ambiente por parte do desenvolvedor, apenas a hospedagem e os custos de processar e servir a lógica de *back-end* de forma granular, ou seja, servindo cada função a ser executada de forma independente, para que haja consumo de recursos apenas durante a execução de cada um desses trechos de código, sem precisar ocupar memória e processamento para todo o código da aplicação de uma vez, e com um *front-end*

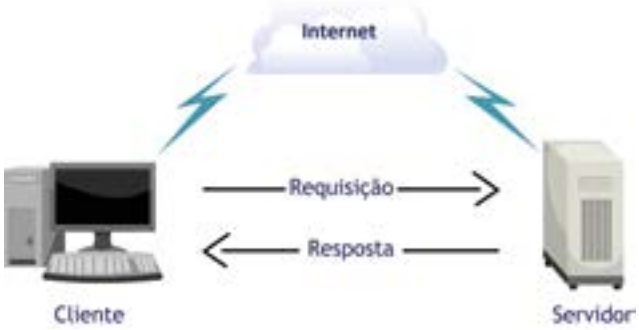
***Front-End*:** designa a camada de uma aplicação diretamente em contato com o usuário final, normalmente a camada de interface. No contexto da *Web*, agrupa as tecnologias que executam no navegador, HTML, CSS e JavaScript.

estático consumindo tais funções independentes, o qual normalmente é servido por CDN. Podemos dizer que tal modelo ainda está engatinhando e a gama de linguagens à disposição ainda é pequena, sendo raro ver o PHP entre elas. Talvez isso mude nos próximos anos.

Funcionamento das requisições HTTP

É importante compreendermos o modelo de requisições da *Web* e como o PHP se encai-xa nesse cenário. Vamos explicar, a seguir, esse processo em 5 passos.

Antes, vamos esclarecer que toda máquina que se propõe a ser um servidor *Web* possui ao menos dois *softwares* fundamentais: um sistema operacional, normalmente Windows ou alguma distribuição Linux, e um aplicativo servidor *Web* propriamente dito, que faz todo o trabalho de comunicação conforme o protocolo HTTP estabelece, ou seja, a partir deste momento, toda vez que estivermos nos referindo ao termo "servidor", estaremos falando desse *software* e não da estrutura computacional como um todo.



1) O passo 1 começa quando um cliente, com seu navegador, envia uma requisição ao servidor, seja um pedido de arquivo ou a submissão de dados através de uma página que já estava previamente em execução. Por exemplo, digamos que você requisitou a página da NT Editora, no endereço <www.nteditora.com.br>. Observe, a seguir, um exemplo de cabeçalho dessa requisição enviada pelo navegador.

```
GET / HTTP/1.1
Host: www.nteditora.com.br
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/71.0.3578.98 Safari/537.36
Accept: text/*,application/xhtml+xml,application/xml;q=0.9,image/*,*/*
Accept-Encoding: gzip, deflate
Accept-Language: pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7
```

Podemos observar, logo no início da instrução, o método de requisição GET, estabelecendo uma solicitação direta de arquivo. Posteriormente, quando estivermos trabalhando com submissão de dados por meio de formulários, normalmente utilizaremos o método POST. O protocolo HTTP estabelece alguns métodos de requisição diferentes, sendo que GET e POST são os mais utilizados. Observe que o navegador preencheu o cabeçalho da requisição também com outras informações relevantes para que o servidor saiba o que fazer, incluindo informações de codificação e de idioma.

Saiba mais

Até informações sobre o agente são enviadas, ou seja, informações sobre o próprio navegador, com detalhes internos de seus motores de execução. Nesse exemplo, a requisição foi enviada a partir da versão 71 do Chrome. É curioso observar que ele também informa o termo Safari, o navegador padrão do macOS. Ocorre que ambos são baseados no mesmo motor de renderização, o WebKit, e, desse ponto de vista, são praticamente idênticos.

2) No Passo 2, o servidor analisa se a requisição é válida, se está no formato correto estipulado pelo protocolo HTTP, e, a partir do endereço que foi requisitado, identifica o arquivo desejado. Se for um arquivo estático, apenas o localiza e envia como resposta. Se for um conteúdo dinâmico, envia as informações para o motor de execução da linguagem em questão, para que faça os processamentos necessários. No caso, o motor de execução PHP é também chamado de interpretador, pois é uma linguagem de *scripts*. Embora possa soar como algo lento, atualmente os interpretadores são muito rápidos, além do *hardware* dos servidores ser bem poderoso e fazer isso facilmente. De qualquer forma, o código interpretado pode ser colocado em cache pelo servidor para que o processamento de interpretação não precise ser repetido a cada nova requisição.

Interpretador: programa que realiza a leitura de um código-fonte em uma linguagem de *script* e o transforma em código executável para o *hardware* em que está localizado.

3) Passo 3, durante o processamento, a linguagem pode executar instruções lógicas, realizar cálculos, acessar outros arquivos, buscar informações em bancos de dados, se comunicar com outros servidores etc.

4) No passo 4, com as informações coletadas, a linguagem gera o corpo da resposta, de acordo com algum dos formatos informados pelo cliente como aceitos. De maneira geral, os navegadores são feitos para carregar e exibir HTML, embora tenham evoluído para exibir vários tipos de imagens e até mesmo executar áudio e vídeo, ou seja, na maioria das vezes, você vai utilizar a linguagem para gerar conteúdo HTML personalizado, com uma estrutura de *tags* pré-estabelecida, conforme o *layout* desejado para a página, mas com o conteúdo que pode mudar dinamicamente conforme os dados e o processamento.

5) O processo termina no passo 5, quando a resposta é enviada ao cliente. Veja, a seguir, uma imagem resumindo todo este fluxo.

Fluxo de execução da requisição à resposta por um servidor *Web* com PHP

Após o final desse fluxo, o arquivo recebido pelo navegador pode conter referências a outros arquivos, como mídias, arquivos de estilo CSS, arquivos JS para execução de lógica no cliente através da linguagem Java*Script*, entre outros que podem ser incorporados a uma página HTML. Todos esses arquivos serão solicitados através de novas requisições através do método GET ao servidor. Por fim, a interação do usuário com a página poderá significar novas requisições ao servidor, solicitando novas páginas ou mesmo submetendo dados da página em questão através do método POST. Dessa forma, um novo processamento ajustará dinamicamente um novo HTML para esse cliente.

Agora que já exploramos em detalhes as características dos servidores *Web* e demonstramos de que maneira o PHP entra nesse cenário, vamos começar a discutir como você pode fazer isso na prática, a partir do próximo tópico!

Programando o conhecimento

A linguagem PHP é uma linguagem de *script*, sendo convertida em código de máquina em tempo real apenas durante a execução. Como também é conhecido esse processo de conversão?

a) Execução.

b) Interpretação.

c) Escriptação.

d) Decodificação.

Comentário: se você pensou na alternativa "b", parabéns! Códigos PHP passam por um processo de interpretação do código-fonte.

1.3 Escolhendo um servidor local

Sendo uma linguagem de *script*, seu trabalho com PHP não passará de código-fonte até o momento da execução. Não dá para abrir diretamente um arquivo PHP em seu navegador preferido e vê-lo funcionando, pois precisamos de um servidor *Web* em execução.

Como já abordamos no tópico anterior, se você quiser que suas criações alcancem o mundo, precisará hospedar seus arquivos em um servidor de verdade. A não ser que queira transformar seu próprio computador em um servidor, negociando um IP fixo com seu provedor de *Internet*, registran-

do esse endereço em servidores DNS e garantindo que sua máquina permaneça ligada o tempo todo, com capacidade de *hardware* e largura de banda para responder todas as requisições! Pois é, dificilmente alguém fará isso.

Durante o desenvolvimento, usaremos um servidor local para testar rapidamente enquanto codificamos, evitando enviar arquivos parciais para um endereço remoto a cada alteração. Vamos discutir algumas opções nas próximas páginas, com foco no sistema operacional Windows (o PHP e a maioria dos servidores executam também em Linux e macOS, mas não teremos como abordar os detalhes específicos de cada plataforma aqui).

Saiba mais

A única diferença efetiva entre um servidor local e um servidor remoto é que o servidor local será instalado em seu próprio computador, para uso durante o desenvolvimento. O *software* utilizado pode ser o mesmo, eventualmente apenas com um pouco mais de cuidado em suas configurações quando realmente for hospedar – principalmente no que for relacionado à segurança.

Servidor embutido do PHP

Desde as origens do PHP como uma linguagem autônoma, sua instalação padrão vem acompanhada de um executável, denominado *php.exe* na versão para Windows, que funciona como CLI, permitindo executar comandos avulsos, ou algum arquivo escolhido, diretamente em um *terminal*, como o *Prompt* de Comando ou o *PowerShell*, por exemplo. Desde a versão 5.4 do PHP, tal ferramenta foi estendida para se comportar como um pequeno servidor *Web* embutido, que pode ser iniciado com apenas um comando no terminal. É uma abordagem simples e leve para aqueles que estiverem interessados apenas em fazer testes rápidos com o PHP, evitando o mínimo possível de instalações adicionais em seu computador. Vamos ver como colocá-lo para funcionar rapidamente.

CLI: *Command-Line Interface*, é um mecanismo exposto por alguns programas para que possam ser manipulados através de comandos textuais.

Certamente, precisamos primeiro baixar o PHP em <php.net/downloads>. Lá, é possível encontrar uma breve lista das últimas versões lançadas e de vários *downloads* relacionados a cada uma delas. A menos que esteja montando um servidor para executar um projeto específico que exige determinada versão, escolha a mais recente.

Importante

Ao entrar na opção de *downloads* do PHP, pode ser assustador ver a quantidade de *links* diferentes disponíveis. Primeiro, escolha entre x64 e x86 de acordo com sua máquina. Atualmente, quase todas são x64, mas, na dúvida, olhe em Sistema, no Painel de Controle do Windows. Sobre a variação entre *Thread Safe* e *Non Thread Safe*, não haverá diferença durante o desenvolvimento com o servidor embutido, apenas se for utilizar outro servidor. Nesse caso, compensa dar uma olhada na documentação oficial do PHP e do servidor escolhido para ter certeza qual é a correta.

Após descompactar o arquivo baixado, por exemplo, em "*C:/php*", você encontrará o executável citado, bem como outros arquivos e pastas. Ali, a pasta "*www*" é onde arquivos para testes podem ser colocados – ela pode ser criada se não existir. Se você estiver ávido para testar o PHP, crie um arquivo "*index.php*" na pasta, com esse trecho simples de código a seguir, usando o "Bloco de notas" mesmo. Parabéns, é seu primeiro código PHP!

```
<?php print "Olá PHP!" ?>
```

Por padrão, o arquivo "*index.php*" será executado pelo seu navegar *Web* quando abrir o endereço que será iniciado pelo servidor embutido. Entretanto, antes de poder iniciá-lo, é preciso ativar um conjunto de configurações padrão do PHP, pois, quando baixamos dire-to do *site* oficial, tais configurações não vêm definidas. Na pasta descompactada do PHP, procure um arquivo chamado "*php.ini-development*" e renomeie para apenas " *php.ini* ".

Pronto, agora que a configuração foi definida para o modo de desenvolvimento, vamos iniciar o servidor embutido acessando a pasta no *terminal* e executando um breve comando. Se você não está acostumado com terminais, segue a tecla SHIFT e clique com o botão direito do *mouse* na pasta "*www*", para ver a opção "*Abrir janela do Prompt de Comando aqui*". No Windows 10, a opção pode ser o *PowerShell*. Então, informe:

```
../php -S localhost:8000
```

Depois disso, pode acessar o endereço <localhost:8000> no navegador, o que disparará a execução do arquivo criado. Se tudo ocorreu bem, você verá a breve mensagem que escrevemos no código aparecer no corpo da página.

E se alguma coisa falhou? Tanta coisa pode dar errado: pode ter escolhido a versão errada do PHP para seu sistema, pode não ter instalado os binários Visual C++ *Redistributable for Visual Studio* que o PHP depende (estão disponíveis também na página de *download* do PHP), pode ter descompactado em algum local com permissões restritas, pode ter errado algum passo na criação do arquivo PHP, pode ter se enganado e aberto o *terminal* na pasta errada, pode ter realizado os passos certos na ordem errada, ou seja, ao mesmo tempo em que a abordagem do servidor embutido é leve e simples, pode causar muita confusão, principalmente em quem está iniciando no PHP. Para facilitar todo o processo de instalação e configuração, inclusive em conjunto com um aplicativo de servidor mais avançado, temos opções mais automatizadas através de pacotes.

Pacotes pré-configurados de instalação

É muito provável que você queira desenvolver testando no mesmo servidor que pretende utilizar em sua hospedagem futura. Entre várias opções existentes no mercado, certamente você ouvirá falar muito sobre o Apache, o mais famoso e consolidado servidor *open source* do mundo, provavelmente o mais utilizado por desenvolvedores PHP.

Saiba mais

Ainda que existam alguns concorrentes de peso, como o proprietário e também muito consolidado *Internet Information Services* (IIS), da Microsoft, e outra opção *open source* que tem crescido em popularidade, o Nginx, o qual normalmente entrega um desempenho superior, o servidor Apache é predominante no ecossistema PHP. Nesse assunto de pacotes de instalação que trazem servidor e PHP pré-configurados, ele é quase unanimidade.

É interessante destacar que, além dos motivos já citados para optar por um pacote pré-configurado de instalação, há outra questão relevante: normalmente também trazem um banco de dados relacional pronto para utilização como parte da instalação, além de fornecer mecanismos simples para iniciar e parar tal serviço – para evitar que seu computador fique com o desempenho afetado por executar um banco de dados à toa, quando não estiver desenvolvendo seu projeto. Vale ressaltar que falaremos da interação do PHP com um banco de dados relacional e, portanto, deixar pré-instalado junto ao servidor local também vai facilitar as coisas no futuro.

Ao longo do tempo, várias soluções de pacotes surgiram e muitas foram abandonadas com o tempo, entre as quais temos atualmente em destaque:

- **XAMPP**: abreviação para X, Apache, MariaDB, PHP e *Pearl*, sendo que o X significa que é multiplataforma (Windows, Linux e macOS), o que o torna uma das soluções mais versáteis e mais utilizadas. Outra curiosidade é em relação à letra M, que, antes de ser MariaDB, representava MySQL, sendo que ainda há resquícios disso por vários lugares, mesmo no *site* oficial, o que ainda confunde muita gente. Durante a instalação, é possível selecionar mais ou menos recursos, por exemplo, não instalar *Pearl*, já que nosso foco é o PHP. Apesar de não estar no nome, a instalação traz ainda o servidor FTP FileZilla, o servidor de *e-mail* Mercury e o servidor Tomcat para códigos Java. Se resolver testá-lo apenas para desenvolvimento PHP, recomendamos que não instale esses recursos adicionais, para não ocupar espaço desnecessário em sua máquina. Vale citar que o XAMPP não possui uma interface muito vasta para configurações, necessitando de algumas interferências diretas em arquivos de configuração. Com um conjunto consistente de recursos, seu *download* tem por volta de 125 MB na versão para Windows com PHP 7.3.1;
- **EasyPHP**: uma das soluções mais antigas para Windows, desde 2000, se mantém bem atualizada, como de costume oferecendo Apache, PHP, MySQL, e também inclui suporte ao servidor Nginx e à linguagem Python, embora não venham instalados inicialmente e precisem de alguns passos a mais para configuração. A ideia é expandirem esse modelo com outras integrações no futuro, sendo os bancos de dados MariaDB e MongoDB listados como os próximos no *site* oficial. Uma vantagem é que o EasyPHP possui uma interface de configuração muito interessante, embutida no próprio servidor como página *Web*, a qual pode ser acessada depois que ele inicia. Se resolver testá-lo, tome o cuidado de entrar nas configurações e alterar a versão do PHP, pois ele vem configurado com a versão 5, embora a 7 também esteja incluída. Com menos recursos pré-instalados, seu *download* padrão tem por volta de 61 MB;
- **WAMP**: abreviação para Windows, Apache, MySQL/MariaDB e PHP, na verdade atualmente é chamado de WampServer. Ao contrário do XAMPP, possui foco no Windows e não oferece o *Pearl*. Também se diferencia por oferecer tanto o MySQL quanto o MariaDB, o que pode ser bem interessante para quem quer mais versatilidade, ou ocupar espaço para quem não quer. O bom é que, através de *menus* simples, permite desativar qual não estiver sendo utilizado. Também permite alternar entre versões PHP e até remover aquela que você realmente não pretenda mais utilizar. Também possui uma interface *Web* padrão com alguns recursos úteis, mas não tão interativa quanto a do EasyPHP. Por fim, como algo útil para brasileiros, ele é o único deles que vem nativamente com tradução para o português, embora algumas coisas continuem em inglês, facilitando compreender os *menus* e as configurações existentes. Com um conjunto consistente de recursos e mais de uma versão para escolher em algumas ferramentas, seu *download* é o mais pesado, por volta de 285 MB na sua versão 3.1.7 para sistemas x64;
- **LAMP** e **MAMP**: como já dito, esse material focará em um ambiente de desenvolvimento no sistema operacional Windows, mas usuários de outros sistemas podem se aventurar nas variantes L (Linux) ou M (macOS).

Por fim, com tantas opções disponíveis, pode ser confuso escolher o caminho a seguir. Na prática, você estará bem servido por qualquer uma dessas opções. Como precisamos de uma padronização neste material, recomendamos que utilize o WAMP, o qual pode ser obtido em <www.wampserver.com/en>, para que possa acompanhar o conteúdo. O processo de instalação é muito simples, daqueles em que só se clica em "*Next*". Inclusive, recomendamos mantê-lo na pasta padrão, em "*c:/wamp64*", considerando que você provavelmente estará utilizando a versão x64.

Programando o conhecimento

Qual é o servidor *Web open source* muito utilizado em projetos PHP, sendo o padrão em todos os pacotes pré-configurados de instalação da atualidade?

a) Apache.

b) IIS.

c) DevServer.

d) XAMPP.

Comentário: se você pensou na alternativa "a", parabéns! O servidor denominado Apache é muito utilizado em todo o mundo, sendo quase uma unanimidade dentro dos pacotes pré-configurados de instalação PHP.

1.4 Configurando o ambiente de desenvolvimento

Chegou a hora de preparar nosso ambiente de desenvolvimento, ou seja, ajustar o nosso computador para desenvolver plenamente com o PHP!

Configurações iniciais do servidor local

Sim, a instalação de um servidor *Web* local faz parte da preparação do ambiente. Então, se você ficou muito ansioso e começou pulando diretamente para cá, sugerimos que volte e passe antes pelo tópico anterior.

Considerando que a instalação do WampServer foi bem-sucedida em seu computador, ao iniciá-lo, você poderá observar um novo ícone na barra de notificações, no canto inferior direito da barra de tarefas. Talvez ele esteja oculto junto a outros ícones, então, arraste-o da área de ícones ocultos diretamente para a área visível da barra. Esse ícone deve iniciar em vermelho e alternar para verde quando os serviços do Apache e dos bancos de dados MySQL e MariaDB estiverem em execução.

***WampServer* inicia com ícone vermelho e se torna verde quando os serviços entram em execução**

De uma forma um pouco estranha, tal ícone oferece *menus* de contexto diferentes se você clicar sobre ele com o botão direito ou com o botão esquerdo do *mouse*.

Com o botão direito do *mouse*, há um *menu* para acessar rapidamente alguns atalhos interessantes, como o endereço <localhost> no navegador – para ver o painel de informações detalhadas do seu WAMP –, a pasta "*www*" na qual você colocará seus projetos, informações e configurações adicionais do Apache, do PHP e dos bancos de dados. Também há atalhos para iniciar, parar e reiniciar os serviços em questão, que é útil se você alterar configurações, exigindo reinicialização dos serviços, ou se quiser parar para aliviar o consumo de recursos de seu computador enquanto faz outras coisas.

Já o botão esquerdo do *mouse* oferece um *menu* de configurações do WAMP, no qual é possível, por exemplo, recarregar todo o programa, escolher o idioma dos *menus* e acessar configurações mais avançadas. Também é por esse *menu* que podemos encerrar completamente a ferramenta, caso não vá mais utilizá-la por algum tempo.

Dica

Se o ícone for de vermelho para amarelo, apenas parte dos serviços foram iniciados. Se permanece vermelho, todos falharam. Ambos os casos podem significar conflitos com outras coisas que já estejam instaladas no computador. Por exemplo, algumas versões do Windows podem vir com o IIS ativado por padrão, sendo necessário localizá-lo no Painel de Controle e desativá-lo. Outra causa comum de problemas é o Skype, por incrível que pareça: algumas versões utilizam a porta 80 por padrão, a mesma que o Apache utiliza para executar o *localhost*. Pare os serviços conflitantes. Se não quiser ou não puder, mude as configurações de portas dos serviços.

Temos uma configuração essencial para fazer: definir a versão mais recente do PHP como a versão padrão para execução. Clique com o botão direito, vá ao *submenu* "*PHP*" e depois ao *submenu* "*Version*". Escolha a maior versão (no momento em que este material é escrito, é a versão 7.3.1). Após a escolha, é natural ver o Apache reiniciando e o ícone do WAMP voltando para amarelo e depois para verde novamente.

Por fim, outra configuração relevante é escolher apenas um dos dois bancos de dados, afinal, não precisamos sobrecarregar o *hardware* do computador mantendo um banco que não será utilizado. Como escolher um deles? Embora este material não seja dedicado a bancos de dados, vamos ver um resumo do que se trata cada um:

- **MySQL**: originalmente criado em 1995 para uso pessoal, foi transformado em um projeto *open source* e evoluiu para uso empresarial, tornando-se o projeto de código aberto mais popular do mundo em pouco tempo. Em 2008, foi adquirido pela Sun Microsystems, a qual foi adquirida pela Oracle em 2010. Como a Oracle possui um dos maiores bancos de dados proprietários do mundo, a comunidade *open source* ficou receosa sobre o futuro do projeto. Ele continua aberto e recebendo atualizações frequentes, sendo ainda uma ótima ferramenta, sempre muito utilizado pela comunidade PHP;
- **MariaDB**: parte dos desenvolvedores do MySQL se incomodaram tanto com a aquisição pela Oracle que criaram um projeto derivado, inicialmente uma réplica idêntica, mas efetivamente livre, sem uma grande empresa por traz. Embora mantenham tal meta, sendo viável alternar entre MySQL e MariaDB sem perceber a diferença, internamente eles têm evoluído por caminhos um pouco diferentes, existindo alguns poucos recursos suportados apenas por esse, bem como algumas diferenças de desempenho. O número de adeptos tem crescido e, sendo o PHP um marco da comunidade *open source*, é natural que muitos migrem para um banco de dados com a mesma filosofia.

Mais uma vez, precisamos de uma padronização para este material, então, também vamos abraçar a filosofia pura do *open source* e adotar o MariaDB. Utilize o botão esquerdo no ícone do WAMP, acesse o *submenu* "*Wamp Settings*" e desmarque a opção "*Allow MySQL*". Se quiser se livrar até mesmo

do espaço em disco que ele ocupa, utilize também o *submenu* "*Delete unused versions*" e remova o MySQL completamente. Se pretende fazer testes com ele no futuro, pode deixá-lo e reativá-lo quando precisar.

Agora que temos nosso servidor executando plenamente, estamos prontos para continuar configurando o ambiente de desenvolvimento.

Editor de códigos-fonte

IDE: *Integrated Development Environment*, é um *software* que reúne ferramentas de apoio ao desenvolvimento de outros *softwares*, normalmente com mais recursos do que editores de texto.

Em nosso rápido primeiro teste com o PHP, foi sugerido utilizar o "Bloco de notas" do Windows mesmo para escrever aquela única linha de código. Ali foi algo simples, mas evidentemente não é o suficiente para desenvolvimento profissional. Programadores profissionais preferem editores dedicados à programação, que ofereçam recursos como coloração de partes do código de acordo com a função, auxílio na manipulação de vários arquivos de um projeto, auto completação de comandos, sugestão de correções etc.

Em diversos ecossistemas de várias linguagens/plataformas, é comum ver os desenvolvedores utilizando alguma IDE. Em cenários nos quais não basta apenas editar os códigos, ter um conjunto de ferramentas integradas, com compilação embutida, além de outros recursos diversos, pode fazer todo o sentido. Há IDEs famosas, como Microsoft Visual Studio, para .NET; Eclipse, NetBeans e IntelliJ, para Java; Android Studio, para An-droid; Xcode, para macOS e iOS; entre várias outras, com diversos focos. Muitas delas abordam mais de uma linguagem/plataforma e poderiam ser utilizadas para desenvolvi-mento PHP sem problemas. Já a IDE denominada PhpStorm é mais focada no PHP, criada com a mesma base da IntelliJ, da empresa JetBrains, oferecendo uma experiência agradável e produtiva, mas com a desvantagem marcante de ser uma ferramenta comercial paga.

Como o foco das IDEs costuma ser abrangente demais, muitas vezes as tornando demasiadamente pesadas, e a PhpStorm não é gratuita, vale a pena considerarmos utilizar um editor de código-fonte ao invés de uma IDE. Eles têm evoluído tanto que essa separação de conceitos se torna cada vez mais difícil, sendo que os recursos oferecidos por alguns editores chegam a igualar algumas IDEs tradicionais.

Importante

Existem vários editores de código, como: Sublime Text, Notepad++, Atom, Brackets e Visual Studio Code – atente-se para não confundir com a IDE denominada apenas Visual Studio. Em se tratando de PHP, muitos desenvolvedores que não estão usando PhpStorm estão usando o Visual Studio Code e com muitas recomendações positivas. Sem dúvidas, vamos adotá-lo neste livro.

Criado pela Microsoft, o Visual Studio Code é um editor moderno, *open source*, leve, multiplataforma (Windows, Linux e macOS), muito bem otimizado para as linguagens *Web*, com suporte nativo

ao PHP, entre várias outras linguagens, e, o melhor, totalmente extensível a partir da instalação de complementos adicionais. Para começar, basta acessar o *site* oficial <code.visualstudio.com> e escolher a opção de *download*. O *site* costuma detectar e oferecer corretamente a versão para seu sistema operacional, mas também permite baixar manualmente outras versões. É um projeto muito ativo e que recebe novas atualizações, ao menos mensalmente, ese atualiza automaticamente com frequência.

Apesar de já vir com suporte ao PHP, é possível melhorá-lo ainda mais a partir da instalação de um complemento. Basta acessar o painel de extensões na barra lateral à esquerda, ou com o atalho Ctrl+Shift+X, buscar pelo termo "PHP", localizar a extensão chamada "*PHP IntelliSense*", provavelmente nos primeiros resultados, e instalar.

Visual Studio Code com o painel de Extensões aberto exibindo a extensão PHP IntelliSense

Após a instalação, é necessário ajustar algumas configurações no seu editor para que a auto completação inteligente de código oferecida pela nova extensão funcione. Comece acessando o *menu* "*File*" > "*Preferences*" > "*Settings*" (ou com o atalho Ctrl+,) no Visual Studio Code. À direita do conteúdo que se abre, procure pelo ícone representado por duas chaves **{ }**, o qual possui o título "*Open Settings (JSON)*" quando estiver com o *mouse* sobre ele. Isso lhe dará acesso direto ao arquivo de configurações do Visual Studio Code para que você possa editá-lo livremente. Supondo que não tenha alteração em nenhuma outra configuração, seu conteúdo ficará preenchido conforme apresentado a seguir.

```
{
  "php.suggest.basic": false,
  "php.validate.executablePath": "c:/wamp64/bin/php/php7.3.1/php.exe"
}
```

JSON: *JavaScript Object Notation* é um formato padronizado e compacto de transferência de dados entre aplicações, inspirado no formato atributo-valor de representação de objetos literais do JavaScript.

Se você tiver ajustado outras configurações, não remova as outras linhas que estarão entre as chaves, apenas acrescente as duas novas linhas exibidas. A primeira configuração, "*php.suggest.basic*", desabilita as sugestões padrão do Visual Studio Code para códigos PHP, evitando sugestões duplicadas, pois a nova extensão exibirá sugestões mais detalhadas. A segunda configuração, "*php.validate.executablePath*", aponta para o caminho em disco onde o interpretador está localizado. Se a versão do seu WAMP não for x64, ou se a versão do PHP não for a 7.3.1, ajuste adequadamente o caminho exibido.

Ótimo, agora temos nosso ambiente executando o servidor local e nosso editor de códigos pronto para desenvolvimento PHP. Entretanto, ainda resta mais um detalhe para configurar, apresentado no próximo tópico, antes de colocarmos a mão na linguagem.

Programando o conhecimento

Qual o endereço padrão a ser acessado no navegador quando visitamos um projeto hospedado em um servidor local executando na porta 80?

a) *wamp*.

b) *host80*.

c) *localhost*.

d) *computer*.

Comentário: se você pensou na alternativa "c", parabéns! O endereço denominado *localhost* é o atalho para o endereço IP do servidor local.

1.5 Habilitando a depuração do código-fonte

Não importa o que você faça, seu código terá erros!

Calma, não estamos jogando uma maldição ou fazendo alguma previsão pessimista sobre você. Mesmo desenvolvedores profissionais muito experientes dificilmente codificam todos os trechos de código sem que nenhum erro aconteça. É algo muito normal. O único problema é não conseguir identificar a causa dos erros, ou pior, deixar passar e entregar um produto ruim. Portanto, investigar e resolver os erros é essencial.

Saiba mais

O termo inglês utilizado para problemas em códigos de programas é *bug*, ou seja, inseto. Nos primórdios, antes das linguagens de programação com instruções textuais, as máquinas eram programadas manualmente, abrindo e fechando circuitos elétricos. Era relativamente comum máquinas apresentarem defeitos quando insetos entravam acidentalmente em algum circuito. Assim, o termo *bug* virou costume e continuou sendo usado.

Antes de começarmos efetivamente com o desenvolvimento PHP, precisamos considerar uma última configuração no ambiente: habilitar a depuração de código.

Instalando o módulo de depuração

O processo conhecido em inglês como *debug,* e, em português, como depuração, evoluiu muito ao longo dos anos. Antes, os desenvolvedores simplesmente tinham à disposição comandos básicos da própria linguagem para escrever passos da execução na tela, como o *print* que já usamos no primeiro exemplo de código PHP, e ficavam apenas observando em qual passo a execução parava, a fim de tentar localizar onde poderia estar um erro. Hoje, podemos parar a execução no ponto desejado, executar linha a linha de forma gra-dual, inspecionar o valor de variáveis, entre outras coisas interessantes.

Ao contrário de outras linguagens que já possuem um módulo *debug*ger embutido, o PHP não traz esse recurso junto à sua instalação padrão, nem mesmo em pacotes pré-configurados de instalação como o WAMP. Necessitamos de um módulo adicional, denominado X*Debug*: um projeto *open source* de extensão para a linguagem em si.

Vamos começar acessando seu *site* oficial em <xdebug.org> e entrando na seção de *download*. Ali começa a confusão, com muitos *link*s para *download* de cada versão disponível. A opção correta depende da arquitetura do sistema, do funcionamento do servidor que estiver executando e da versão da linguagem PHP utilizada.

No caso do Wamp64, baixe a versão cujo *link* termina com os termos "*TS (64bit)*". No momento em que este material foi escrito, seria o *link* "*PHP 7.3 VC15 TS (64 bit)*" o que baixaria o arquivo "php_*xdebug*-2.7.0RC1-7.3-vc15-x86_64.dll". Dependendo do navegador que estiver utilizando, como o Chrome, por exemplo, você pode ser avisado sobre o risco de baixar arquivos DLL, o que não é um problema neste caso, já que sabemos do que se trata. Escolha a opção apropriada no navegador para manter o *download* do arquivo.

Saiba mais

Se você escolheu outro pacote de servidor ao invés do WAMP, ou se não for a versão x64, ou ainda se houver alguma alteração no Apache futuramente, poderá ser necessário escolher outra versão do XDebug. Para não baixar vários arquivos em vão, acesse <localhost/?phpinfo=1>, com o servidor local em execução, copie todo o conteúdo dessa página e cole o que foi copiado na área indicada no endereço <xdebug.org/wizard.php>. Tal página de apoio analisará as informações específicas de sua instalação do PHP e informará qual versão do XDebug sua máquina realmente precisa.

Após o *download*, mova a DLL para "*C:\wamp64\bin\php\php7.3.1\ext*". Novamente, aten-te-se para qualquer alteração que for necessária para chegar até a pasta de destino, caso a versão do WAMP ou do PHP sejam diferentes em seu ambiente. Após colocar o arquivo baixado nessa pasta, renomeie o arquivo apenas para "*php_xdebug.dll*", com o propósito de simplificar as coisas. Agora, acesse o *menu* de contexto do WAMP com o botão direito, abra o *submenu* "PHP" e escolha a opção "*php.ini*" para podermos editar brevemente o arquivo de configurações do PHP. Por padrão, o WAMP deve abrir o "Bloco de notas" para essa edição, mas, como será algo breve, não tem problema.

O mais importante é descer até o final do arquivo e localizar uma linha que se inicia com "*;zend_extension*", removendo esse ponto e vírgula e conferindo o caminho que segue tal configuração, para garantir que se refere ao caminho da DLL que colocamos na pasta. Além disso, para a integração com o Visual Studio Code funcionar, precisaremos habilitar as opções "*xdebug.remote_enable*" e "*xdebug.remote_autostart*", atribuindo o valor 1 para ambas. Se tudo isso parece confuso, dê uma olhada em como devem ficar as últimas linhas de seu arquivo de configuração após os ajustes necessários.

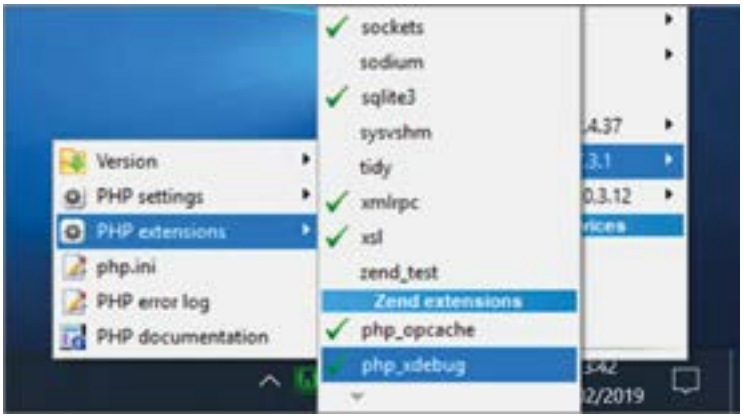
```
[XDebug]
zend_extension="c:/wamp64/bin/php/php7.3.1/ext/php_xdebug.dll"
xdebug.remote_enable = 1
xdebug.remote_autostart = 1
xdebug.profiler_enable = 0
xdebug.profiler_enable_trigger = 0
xdebug.profiler_output_name = cachegrind.out.%t.%p
xdebug.profiler_output_dir ="c:/wamp64/tmp"
xdebug.show_local_vars=0
```

As outras linhas de configuração exibidas já estavam no arquivo, e foram apenas mantidas. Caso sua versão do WAMP traga algumas configurações levemente diferentes, não recomendamos que modifique tudo, apenas ajuste as configurações citadas. E, mais uma vez, vale enfatizar: pode ser necessário ajustar o caminho até a DLL conforme seu ambiente, se estiver utilizando outra versão do WAMP ou do PHP.

Após salvar as edições realizadas, é preciso reiniciar os serviços. Para fazer isso rapidamente, acesse o *menu* do WAMP com o botão esquerdo e escolha "*Refresh*" para reiniciar serviços e o próprio aplicativo. Se os passos foram realizados corretamente, após o WAMP completar a reinicialização, o X*Debug* deve estar habilitado.

Se quiser conferir se o WAMP realmente reconheceu o novo módulo habilitado, acesse o *menu* de contexto com o botão direito, vá para o *submenu "PHP"* e depois para o *submenu* "*PHP extensions*". Há uma lista grande de extensões, percorra para baixo até ver a opção "*php_xdebug*" checada com um marcador verde, como na imagem a seguir.

Extensão X*Debug* marcada como habiliada no *menu* de contexto do WAMP



Instalando a extensão de depuração no Visual Studio Code

Tudo pronto no próprio servidor local, mas falta ajustar alguns detalhes em seu Visual Studio Code para que a depuração seja iniciada quando você precisar.

Importante

Apesar de o Visual Studio Code oferecer recursos de depuração nativa para muitas linguagens, para o PHP dependemos da extensão "*PHP Debug*".

Os passos para instalar uma extensão no Visual Studio Code já foram abordados no tópico anterior, sendo os mesmos aqui, bastando desta vez localizar a "PHP *Debug*" na lista de extensões durante sua pesquisa pelo termo "PHP".

Além da instalação da extensão "PHP *Debug*", realizada apenas uma vez para que fique disponível sempre que você executar o Visual Studio Code, é necessário ajustar algumas configurações de execução específicas para cada projeto que estiver trabalhando, ou seja, os próximos passos serão necessários para cada novo projeto que criar.

Configurando a depuração no projeto

Já dissemos que os projetos devem ser criados como uma nova pasta, dentro da pasta em "*C:\wamp64\www*". Caso ainda não tenha feito isso, crie uma nova pasta ali com o nome que desejar. Neste material, chamamos de "exemplos".

Agora, no Visual Studio Code, utilize o *menu* "*File*" > "*Open Folder*" para localizar a pasta em questão e abri-la no editor. A partir daí, você terá acesso rápido para criar novos arquivos e pastas a partir do próprio editor, o que agiliza o desenvolvimento.

Além disso, para encerrar a configuração, você precisa acessar a opção *Debug* na barra lateral esquerda do editor, ou com o atalho Ctrl+Shift+D, e clicar no ícone de uma engrenagem logo no topo do painel que se abriu. Aparecerá uma caixa no meio da tela para que você selecione qual o ambiente de desenvolvimento desse projeto, sendo que você deve escolher a opção "PHP" na lista. Feito isto, um novo arquivo "*launch.json*" deve ser criado automaticamente para seu projeto, com um conteúdo similar ao seguinte.

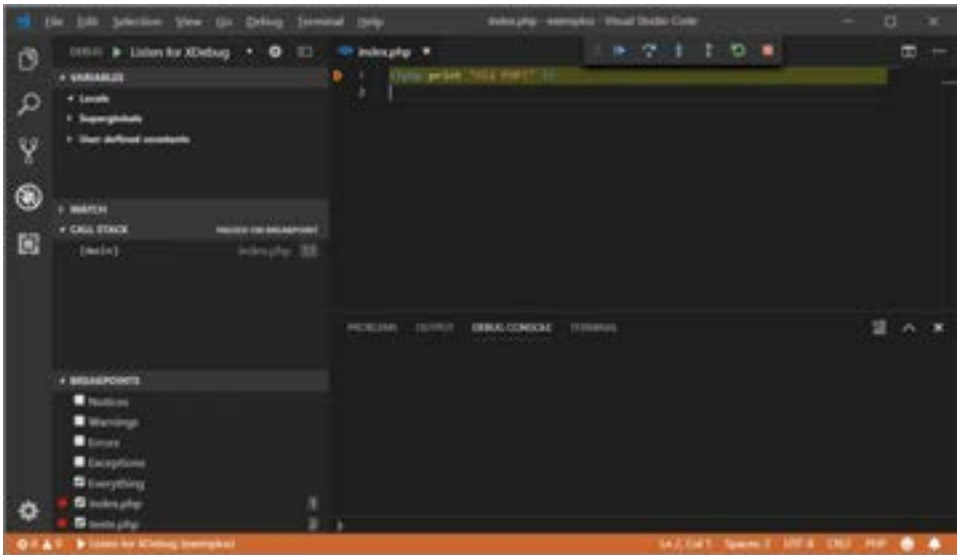
Dependendo da versão do Visual Studio Code, isso pode ser um pouco diferente e existirem outras configurações em outras linhas, mas este trecho "*Listen for* X*Debug*" é o mais importante. Graças a isso, nosso editor terá associado ao projeto um comando de depuração que inicia o X*Debug* em sua porta padrão, a porta 9000, e o deixa esperando por uma ação de depuração, ou escutando, no termo em inglês *listen*, enquanto você navega em alguma página PHP do projeto.

Para iniciar essa escuta de depuração, clique de "*Start Debugging*" no painel de depuração – um ícone verde com o formato triangular de um botão *play* –, ou simplesmente utilize a tecla de atalho *F5* quando quiser iniciar a depuração no projeto. Recomendamos que pare a depuração toda vez que for criar novos arquivos ou alterar o código dos arquivos existentes, reiniciando a depuração com F5 quando estiver pronto para testar.

Agora, só falta garantirmos que tudo está funcionando no ambiente criando um arquivo "*index.php*" na pasta aberta no editor ou copiando o arquivo que já usamos antes. Com o arquivo aberto,

clique ao lado da numeração da única linha existente nesse código. Uma bolinha vermelha deve aparecer, que chamamos de "*breakpoint*", ou seja, o ponto de parada do depurador durante a execução desse arquivo no navegador. Coloque a depuração para executar e abra o endereço do projeto no navegador <localhost/exemplos>. Assim que a interpretação iniciar, a depuração pausará a execução na linha indicada.

Visual Studio Code com a execução pausada durante a depuração de um código PHP



Não vamos nos aprofundar em cada recurso de depuração existente. Você pode passear pelas opções por conta própria e encontrar ajuda na documentação do Visual Studio Code, se precisar. E, obviamente, enquanto temos apenas uma linha de código, a depuração não serve para muita coisa, mas mantenha tudo isso funcionando e terá muito mais agilidade ao identificar possíveis problemas em seus códigos enquanto prossegue nos estudos.

Com tudo isso configurado, estamos prontos para seguir para o próximo capítulo e finalmente começarmos a estudar a linguagem PHP em si.

Programando o conhecimento

Qual o projeto *open source* de um módulo para a linguagem PHP que precisa ser utilizado para permitir a depuração de códigos?

a) PHP IntelliSense.

b) XDebug.

c) PHP Debug.

d) Apache.

Comentário: se você pensou na alternativa "b", parabéns! O projeto XDebug permite habilitar os recursos de depuração na linguagem. As alternativas "a" e "c" se referem a extensões para o Visual Studio Code e não ao módulo para a própria linguagem PHP, enquanto a alternativa "d" se refere ao servidor *Web* utilizado para execução do projeto.

Resumindo

Nesta lição, você conheceu de forma geral a linguagem de programação PHP, sua história e suas principais características. Teve a oportunidade de revisar o funcionamento dos servidores *Web* e compreender como o PHP se situa nesse contexto. Além disso, pôde observar as questões relacionadas à configuração de um servidor local para desenvolvimento do ambiente de desenvolvimento PHP para os estudos e as atividades práticas através do Visual Studio Code e, inclusive, dos componentes necessários para permitir a depuração dos códigos-fontes que criaremos nos capítulos seguintes.

Agora, verifique se você está apto a:

- explicar a história e as características gerais do PHP;
- discutir sobre o funcionamento geral dos servidores *Web*;
- elencar os passos para execução de um servidor *Web* localmente;
- configurar seu computador como um ambiente de desenvolvimento PHP;
- habilitar as ferramentas para depuração do código-fonte.

Exercícios

Parabéns, você finalizou esta lição!

Agora responda às questões ao lado.

Questão 1 - O nome da linguagem de programação PHP é um acrônimo recursivo, ou seja, uma brincadeira com o próprio nome, no qual o próprio acrônimo aparece repetidamente em seu significado. Com isso em mente, qual o nome dessa linguagem de programação?

a) Programming Higher with PHP.

b) PHP: HTML Programming.

c) PHP: Hypertext Preprocessor.

d) Procedural Hypertext Programming.

Questão 2 - A seguir, você encontrará afirmações sobre a linguagem PHP. Analise-as com atenção e julgue os itens em V (verdadeiro ou F (falso).

a) ( ) Mantida por um projeto livre de código aberto, foi criada para atender às necessidades dos próprios desenvolvedores e refinada ao longo do tempo em prol de sua comunidade.

b) ( ) Interpretada no *client-side*, ou seja, depende apenas de um navegador *Web* para execução, assim como ocorre com a linguagem JavaScript.

c) ( ) Inspirada principalmente nas linguagens preexistentes C e Pearl, posteriormente recebe forte influência também da linguagem Java.

d) ( ) Constituída principalmente por regras de sintaxe, um conjunto de palavras reservadas e um extenso núcleo de tipos, funções, classes e objetos.

Questão 3 - A linguagem PHP iniciou como um conjunto simples de *scripts* utilitários escritos em linguagem C como binários CGI, para o *site* pessoal de seu criador. Em dois anos, passou por uma forte reconstrução iniciada por dois outros desenvolvedores, os quais criariam, no ano seguinte, a Zend Engine, o interpretador para execução da linguagem. Quais os nomes desses três desenvolvedores essenciais para a história do PHP?

a) Rasmus Lerdorf, Tim Berners-Lee e Linus Torvalds.

b) Rasmus Lerdorf, Zeev Suraski e Andi Gutmans.

c) Zeev Suraski, Andi Gutmans e Larry Page.

d) Tim Berners-Lee, Larry Page e Zeev Suraski.

Questão 4 - A seguir, você encontrará afirmações sobre diferentes pacotes pré-configurados de instalação do PHP, contendo um servidor *Web*, o interpretador PHP e, eventualmente, bancos de dados e, até mesmo, outras aplicações de apoio. Analise-as com atenção e julgue os itens em V (verdadeiro) ou F (falso).

a) ( ) O WAMP é uma solução específica para Windows em um pacote que traz Apache, PHP e MySQL ou MariaDB, permitindo inclusive alternar entre estes dois SGBDs.

b) ( ) O EasyPHP é um dos mais antigos pacotes multiplataforma, oferecendo Apache, PHP, MySQL e Python, suportando alternar para os servidores Nginx ou Lighttpd.

c) ( ) As soluções LAMP e MAMP iniciaram-se como vertentes, respectivamente, do WAMP para Linux e macOS, mas, atualmente, também se tornaram multiplataforma.

d) ( ) O XAMPP é um pacote que traz Apache, MariaDB, PHP e Pearl, sendo uma famosa solução multiplataforma para Windows, Linux e macOS.

Questão 5 - A seguir, você encontrará afirmações sobre as principais características da linguagem PHP. Analise-as com atenção e julgue os itens em V (verdadeiro) e F (falso).

a) ( ) É uma linguagem prática, não exigindo gerenciamento direto da memória, simplificando o trabalho com tipos de dados e oferecendo conversões automáticas por contexto.

b) ( ) É uma linguagem flexível, servindo o desenvolvedor com diversas alternativas para soluções corriqueiras, por exemplo, suportando diversos bancos de dados.

c) ( ) É uma linguagem livre, sem custos diretos, sem restrições de uso, de modificações ou de redistribuição, aberta à colaboração da comunidade em um repositório público.

d) ( ) É uma linguagem defasada, não recebendo novas versões desde 2005, permanecendo relevante apenas porque há muito código legado espalhado pelo mercado.

Questão 6 - No contexto da arquitetura cliente-servidor que fundamenta a *Web*, os servidores são eventualmente conhecidos também pelo termo em inglês:

a) *hosts*.

b) *requests*.

c) *responses*.

d) *hardwares*.

Questão 7 - Existem várias formas de categorizar servidores. A seguir, apresentamos alguns tipos de servidores recorrentes para o funcionamento corriqueiro da *Web*. Qual das alternativas não é um desses servidores relevantes para a maioria das páginas *Web*?

a) Servidores DNS.

b) Servidores Web.

c) Servidores CDN.

d) Servidores P2P.

Questão 8 - Existem diversos editores de código, alguns focados na *Web*, outros mais genéricos, alguns gratuitos, outros pagos. Não podemos confundir editores de código com IDEs, pois estas costumam ser maiores, normalmente vinculadas a alguma plataforma de programação específica e, em muitos casos, mais pesadas para execução. Entre alguns dos editores de código mais conhecidos, podemos destacar:

I. NetBeans.

II. Notepad++.

III. Visual Studio Code.

IV. PhpStorm.

V. Sublime Text.

É correto afirmar que:

a) As afirmativas I, II e III estão corretas.

b) As afirmativas III e IV estão corretas.

c) As afirmativas II, III e V estão corretas.

d) Todas as afirmativas estão corretas.

Questão 9 - A seguir, você encontrará afirmações sobre cada uma das gerações tipicamente utilizadas para descrever a evolução dos servidores *Web*. Analise-as com atenção e julgue os itens em V (verdadeiro) e F (falso).

a. ( ) A quarta geração foi proposta por volta de 2010, mas não vingou por sua complexidade, sendo difiícil encontrar plataformas desse tipo que suportem PHP.

b. ( ) A terceira geração surgiu nos anos 2000, quando os servidores passaram a processar os arquivos antes de entregar aos clientes, mediante extensões CGI.

c. ( ) A primeira geração surgiu há aproximadamente 30 anos, com o início da *Web*, e se trata da entrega de arquivos completamente estáticos aos clientes.

d. ( ) A segunda geração teve início em meados de 1995, com muitos princípios utilizados ainda hoje, sendo essa a geração que marcou o surgimento do PHP.

Questão 10 - Suponha que uma requisição inicial com o método GET do protocolo HTTP foi recebida pelo servidor *Web*, processada pelo PHP e devolvida na forma de um HTML. Suponha também que essa página retornada possui um formulário com vários campos

que serão submetidos de volta ao servidor em requisições subsequentes, justamente para o mesmo endereço da requisição inicial. Qual método do protocolo HTTP será utilizado para essas requisições posteriores de envio de dados?

a) GET.

b) POST.

c) DELETE.

d) DNS.