PHP

Hypertext Preprocessor

Pedro Cavalheiro - 2015

O que é?

- Linguagem de programação interpretada
- Open source
- Server side
- Procedural ou Orientada a objetos
- Tipagem fraca

Para que serve?

- Scripts server side (coletar dados de formulários, gerar páginas com conteúdo dinâmico, manipular cookies, etc)
- Scripts de linha de comando, sem necessidade de interface gráfica (cron)
- Aplicações desktop (PHP-GTK, não oficial)
- Processamento de imagens, documentos, XML, etc
- Conexões com diversos protocolos (HTTP, IMAP, POP3, LDAP, etc)

Por que aprender?

GitHut	RedMonk	Jobs Tractor	TIOBE Index	Resultado
1. JavaScript	1. JavaScript	1. Java	1. C	1. Java (all)
2. Java	2. Java	2. Objective-C	2. Java	2. JavaScript
3. Python	3. PHP	3. PHP	3. C++	3. PHP
4. CSS	4. Python	4. SQL	4. Objective-C	4. Python
5. PHP	5. C#	5. Java (Android)	5. C#	5. C / C++
6. Ruby	6. C++	6. C#	6. JavaScript	6. C#
7. C++	7. Ruby	7. JavaScript	7. PHP	7. Objective-C
8. C	8. CSS	8. Python	8. Python	8. Ruby
9. Shell	9. C	9. Ruby	9. VisualBasic.NET	9. Visual Basic
10. C#	10. Objective-C	10. C++	10. Visual Basic	

via Sitepoint

Por que aprender?

É fácil.

Onde fica o PHP?

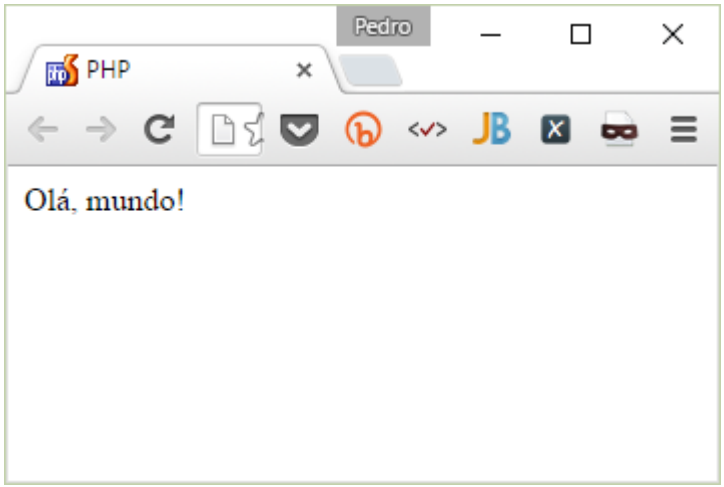
Como o próprio nome diz, o PHP é um pré processador de páginas, ou seja, é executado antes da resposta as requisições feitas pelo navegador, apenas no lado do servidor.

- Requer um interpretador instalado no servidor (Apache + mod_php)
- No Windows, existem pacotes prontos, como XAMPP e WAMP.
- O PHP aceita diversas extensões, acrescentando funcionalidades variadas à linguagem
- A maioria de suas configurações fica no arquivo php.ini

Executando código PHP

É possível inserir código PHP em qualquer lugar da página, através das tags <?php e ?>

```
<body>
    <?php echo 'Olá, mundo!'; ?>
</body>
```



Variáveis

As variáveis no PHP são representadas por um cifrão ($) seguido pelo nome da variável. Os nomes de variável no PHP fazem distinção entre maiúsculas e minúsculas. Um nome de variável válido se inicia com uma letra ou sublinhado, seguido de qualquer número de letras, algarismos ou sublinhados. Mais infos

```php
<?php
$var = 'Bob';
$Var = 'Joe';
echo "$var, $Var";      // exibe "Bob, Joe"

$4site = 'not yet';     // inválido; começa com um número
$_4site = 'not yet';    // válido; começa com um sublinhado
$täyte = 'mansikka';    // válido; 'ä' é um caracter ASCII (extendido) 228
?>
```

Tipos

São quatro tipos escalares: boolean, integer, float, string

Dois tipos compostos: array, object

E finalmente, dois tipos especiais: resource, NULL

Condicional - if...else

```php
<?php
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
?>
```

Mais infos

Condicional - switch

```
switch ($i) {
    case 0:
        echo "i igual a 0";
        break;
    case 1:
        echo "i igual a 1";
        break;
    default:
        echo "i diferente de 0 e 1";
        break;
}
?>
```

Mais infos

For

```php
<?php
/* parâmetros são opcionais */
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
?>
```

Mais infos

Arrays

Um array no PHP é atualmente um mapa ordenado. Um mapa é um tipo que relaciona valores para chaves. Este tipo é otimizado de várias maneiras, então você pode usá-lo como um array real, ou uma lista (vetor), hashtable (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente mais. Mais infos

```php
<?php
$arr = array("foo" => "bar", 12 => true);
echo $arr["foo"]; // bar
echo $arr[12];    // 1

$arr = array("somearray" => array(6 => 5, 13 => 9, "a" => 42));
echo $arr["somearray"][6];    // 5
echo $arr["somearray"][13];   // 9
echo $arr["somearray"]["a"];  // 42
?>
```

Funções

Qualquer código PHP válido pode aparecer dentro de uma função, mesmo outras funções e definições de classes. Mais infos

```php
<?php
function foo ($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Exemplo de função.\n";
    return $valor_retornado;
}
?>
```

Classes

Mais infos

```php
<?php
class SimpleClass extends ParentClass
{
    // declaração de membro - public/private/protected
    public $var = 'um valor padrão';

    // declaração de método - public/private/protected
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

Auxiliar para debug

```
var_dump ($var1, $var2 … $varX );
```

Info

```
<?php
$a = array (1, 2, array ("a", "b", "c"));
var_dump ($a);
?>
```

```
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(2)
  [2]=>
  array(3) {
    [0]=>
    string(1) "a"
    [1]=>
    string(1) "b"
    [2]=>
    string(1) "c"
  }
}
```

```
echo $var;
```

Info

```
<?php
$foo = "foobar";
$bar = "barbaz";

echo "Hello World"; // Hello World
echo $foo; // foobar
echo $foo, $bar; // foobarbarbaz

?>
```

Include

A instrução *include* inclui e avalia um arquivo específico. Mais infos

vars.php

```php
<?php

$color = 'green';
$fruit = 'apple';

?>
```

test.php

```php
<?php
echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

Require

Igual ao include, mas para a execução no caso do arquivo não existir, ou erro ao incluir. Mais infos

vars.php

```php
<?php

$color = 'green';
$fruit = 'apple';

?>
```

test.php

```php
<?php
echo "A $color $fruit"; // A

require 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

Coletando dados

Construindo o formulário

```
<form action="cadastro.php" method="get">
  Nome: <input type="text" name="username"><br>
  Email: <input type="text" name="email"><br>
  <input type="submit" value="Enviar!">
</form>
```

- **Action:** destino para onde os dados do formulário serão enviados
- **Method:** define o método de envio dos dados (GET ou POST). Nunca usar GET para dados importantes (senhas, dados bancarios, etc)
- **Enctype:** define o modo como os dados devem ser codificados, ao enviar o formulário
- **name:** propriedade, nos inputs, que define sua a identificação do lado o servidor. **Obrigatório.**

GET

Solicita dados de uma fonte específica

Dados são enviados numa “query string” (pares chave/valor) presente na URL da requisição:

```
/cadastro.php?nome=Fulano&email=fulano@gmail.com
```

Características:

- Requisições GET podem ser cacheadas
- Requisições GET ficam armazenadas no histórico do navegador
- Requisições GET podem se tornar favoritos
- Requisições GET nunca devem ser usadas para enviar dados importantes
- Requisições GET tem restrições de tamanho (depende do cliente e servidor)
- Requisições GET devem ser usadas apenas para solicitar dados

POST

Envia dados para serem processados em determinado local

Dados são enviados numa “query string” (pares chave/valor) presente no “corpo” da requisição:

```
POST /cadastro.php HTTP/1.1

Host: senac.com.br

nome=Fulano&email=fulano@gmail.com
```

Características:

- Requisições POST nunca são cacheadas
- Requisições POST não ficam armazenadas no histórico do navegador
- Requisições POST não podem se tornar favoritos
- Requisições POST não tem limite de tamanho

Recebendo dados no PHP

Quando um formulário é submetido para um script PHP, qualquer variável do formulário será automaticamente disponível para o script, na variável correspondente, no formato de um array. Mais infos

$_POST

$_GET

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

```php
<?php
        var_dump($_POST);
?>
```

```
array(2) {
  ["nome"]=>
  string(6) "Fulano"
  ["email"]=>
  string(16) "fulano@gmail.com"
}
```

Cookies

Cookie é um mecanismo para guardar dados no navegador remoto e permite o rastreamento ou identificação do retorno de usuários.

DEFINIR

```php
<?php
setcookie("nome_do_cookie", "valor", $tempo);
?>
```

$tempo é opcional para setar a "vida" do cookie

LER

```php
<?php
echo $_COOKIE["nome_do_cookie"];
?>
```

Session

As sessões têm um princípio similar aos cookies, só que o armazenamento do estado é feito pelo servidor Web, e não pelo navegador.

SEMPRE

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

DEFINIR

```php
<?php
$_SESSION["nome"]=$valor;
?>
```

LER

```php
<?php
echo $_SESSION["nome"];
?>
```

REMOVER

```php
<?php
unset($_SESSION["nome"]);
?>
```

Conexão com DBs

O PHP se conecta aos bancos de dados através de ~drivers~. Cada banco de dados possui 1 ou mais drivers disponíveis.
Um driver é uma extensão do PHP, que adiciona algumas funções pré-definidas, possibilitando a execução de todo tipo de comando SQL no banco de dados. Mais infos

O MySQL possui 3:

- mysqli
- mysql***
- PDO

***Não usar, foi descontinuado e será removido nas próximas versões do PHP

Exemplo de consultas

```php
<?php
// mysqli
$mysqli = new mysqli("127.0.0.1", "usuario", "senha", "nome_do_banco");
$result = $mysqli->query("SELECT id, nome FROM pessoas");
$row = $result->fetch_assoc();
echo $row['nome'];

// PDO
$pdo = new PDO('mysql:host=127.0.0.1;dbname=nome_do_banco', 'usuario', 'senha');
$statement = $pdo->query("SELECT id, nome FROM pessoas");
$row = $statement->fetch(PDO::FETCH_ASSOC);
echo$row['nome']);
```

mysqli

```php
<?php
// abre uma conexão com o banco de dados, endereço “127.0.0.1”, usuário “user”, senha “pass” e banco de dados “database”
$mysqli = new mysqli("127.0.0.1", "user", "pass", "database");

// executa o comando SELECT no banco de dados, guardando a resposta na variável $result
$result = $mysqli->query("SELECT id, nome FROM pessoas");

// traduz a linha atual num array onde as colunas (do banco de dados) são as chaves
$row = $result->fetch_assoc();

// escreve o valor da chave (coluna) nome
echo $row['nome'];
```

SQL Injection - Exemplo 1

Uma das maiores vulnerabilidades de sites, a injeção de SQL (SQL Injection) é também, no caso do PHP, uma das mais fáceis de prevenir. Infelizmente, muitos não tomam as devidas precauções e acabam tendo os seus dados comprometidos.

```
// Temos esta consulta simples, onde os dados $username e $password vem de um formulário preenchido pelo usuário
$query = "SELECT * FROM users WHERE username = '$username'";

// Se não houver validação correta, um usuário mal-intencionado poderia colocar algum código SQL no lugar do username
$username = "' OR 1";

// A consulta ficaria assim:
$query = "SELECT * FROM users WHERE username = '' OR 1";

// Como a expressão 'OR 1' sempre resulta em TRUE, a consulta retornaria os dados de todos os usuários no sistema
```

SQL Injection - Exemplo 2

Convenhamos, o exemplo anterior é bobo, mas serve para mostrar a teoria por trás da técnica de injeção de sql.
Vejamos um exemplo mais grave:

```
// começamos com a mesma consulta
$query = "SELECT * FROM users WHERE username = '$username'";

// desta vez, o código inserido é realmente malicioso
$username = "'; DELETE FROM users WHERE 1 OR username = '";

// a consulta final ficaria assim:
$query = "SELECT * FROM users WHERE username = ''; DELETE FROM users WHERE 1 OR username = ''";

// ou seja, se executada, a consulta removeria todos os registros da tabela users
```

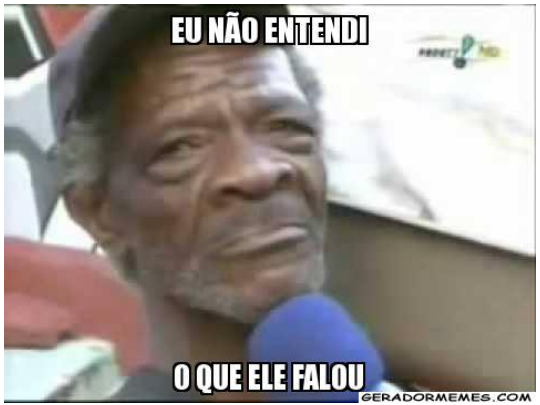
Como prevenir?

Existem diversos modos para prevenir este tipo de situação. O mais comum é através de:

~PREPARED STATEMENTS~

O que são *prepared statements*?

Nada mais são do que consultas “pré-prontas”… A diferença é que em lugar das variáveis, é colocado um *placeholder* (marcador de lugar), que será substituído apenas no momento da consulta, com o tratamento adequado.



Prepared Statements

```php
<?php
// abre conexão com o banco
$mysqli = new mysqli("127.0.0.1", "user", "pass", "database");
// prepara a SQL para receber os parametros. Neste caso, apenas 1.
$con = $mysqli->prepare("SELECT id, nome FROM pessoas WHERE id = ?");
// define o ID do usuário a ser consultado na variável $id
$id = 1;
// atribui a variável $id ao primeiro ?, com o filtro “i” (inteiro). Filtros aceitos: i - inteiro, d - double, s - string, b - blob
$con->bind_param("i", $id);
// executa a query já com o parâmetro incluído
$con->execute();
// define que os resultados encontrados serão gravados nas variáveis $id e $nome, obedecendo a ordem das colunas consultadas
$con->bind_result($id, $nome);
//grava o resultado encontrado nas variáveis
$con->fetch();
//lalala
echo $nome;
```

Capturando erros

```php
<?php
$mysqli = new mysqli("127.0.0.1", "user", "pass", "database");
/* testa sucesso da conexão */
if (mysqli_connect_errno()) {
    echo "Conexão falhou! Mensagem: " . mysqli_connect_error();
    exit;
}
if (!$mysqli->query("SELECT * FROM users")) {
    echo "Erro na consulta! Mensagem: " . $mysqli->error;
}
/* fechando a conexão */
$mysqli->close();
?>
```

Uso comum

A sintaxe base apresentada pode ser aplicada para qualquer operação realizada no banco de dados

SELECT

```
$mysqli->query("SELECT id, nome FROM pessoas");
```

INSERT

```
$mysqli->prepare("INSERT INTO pessoas (id, nome) VALUES (?, ?)");
```

UPDATE

```
$mysqli->prepare("UPDATE pessoas SET nome = ? WHERE id = ?");
```

DELETE

```
$mysqli->prepare("DELETE FROM pessoas WHERE id = ?");
```

Documentação mysqli

http://php.net/manual/pt_BR/class.mysqli.php