

**CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA**

**Apostila de Desenvolvimento Web
2024**

**Professor:
Rafael Guimarães Rodrigues**

Sumário

1.	Conteúdo Programático da Disciplina	4
2.	A Linguagem PHP	4
3.	Como usar o PHP	5
4.	Arquitetura Cliente Servidor	9
5.	Delimitadores, comentários e o primeiro arquivo	11
6.	Variáveis	12
7.	Tipos de Dados suportados	13
8.	Apóstrofos, Aspas e Caracteres de Controle	14
9.	Interpretação de variáveis numa string	15
10.	Concatenação de strings	15
11.	Impressão de strings	15
12.	Impressão com término da execução	16
13.	Conversão de tipo	16
14.	Verificação de tipo de uma variável	19
15.	Exibindo a estrutura de uma variável	19
16.	Verificando se uma variável possui um valor	20
17.	Constantes	21
18.	Operadores de atribuição	22
19.	Operadores pós fixos	23
20.	Operadores prefixos	23
21.	Cuidados com os pós fixos	23
22.	Operadores aritméticos	23
23.	Operadores de comparação e Operadores lógicos	24
24.	Operador ternário	25
25.	Escopo das variáveis	26
26.	Variáveis globais	26
27.	If, else if, else	28
28.	switch	29
29.	for	32
30.	while	33
31.	Do-while	34
32.	Interferindo no loop com break e continue	35
33.	Funções	36

34. Passagem de parâmetros por valor e por referência	37
35. Parâmetros com valores predefinidos (default)	37
36. Funções com argumentos variáveis	38
37. Manipulação de Arrays.....	39
38. Manipulação de Strings	51
39. Inclusão de arquivos.....	55
40.Funções – Um pouco mais.....	56
41.Funções anônimas e callback	59
42.Requisição e Resposta (Revisão)	60
43.Envio de Formulário por HTML.....	61
44.Envio de dados na requisição	63
45.Redirecionamento	64
46.Upload de Arquivos	65
47. Outras funções para arrays	70
48. Um pouco sobre funções de data e hora	72
49.Funções recursivas.....	73
50.Hash e funções de hash	78
51.Listas	80
52.Configuração regional com strftime e setlocale().....	81
53.DateTime.....	82
54.PDO.....	82
55.Realizando CRUD com PDO	88
56.PDO – query e os tipos de fetch	93
57.PDO – query e foreach	98
58.PDO – SQL INJECTION - PROBLEMA	104
59.PDO – Evitando SQL INJECTION – PREPARE e EXECUTE	109
60.CRUD via PDO (HTML, CSS e PHP).....	112
61.CRUD com requisições AJAX/AJAJ	121

1. Conteúdo Programático da Disciplina

O que vamos aprender?

Neste curso aprenderemos a desenvolver aplicações web com a linguagem PHP, acesso a banco de dados, além de integração com tecnologias aprendidas em outras disciplinas.

Por que PHP?

O PHP vem se consolidando, ao longo dos anos, como uma das linguagens de programação orientadas a objetos que mais crescem no mundo. A linguagem PHP é amplamente difundida e representa uma fatia considerável das aplicações Web na atualidade.

2. A Linguagem PHP

Histórico

A linguagem PHP foi concebida em 1994 por Rasmus Lerdorf para ser utilizada exclusivamente na Web (rodar em um servidor web). Originalmente a sigla representava Personal Home Pages. Posteriormente, PHP: Hypertext Processor.

A página oficial do PHP é <http://www.php.net>. Nesta página você encontra um manual completo, em português, sobre a linguagem. Atualmente, em 2023, o PHP está em sua versão 8.0.2.

O que distingue o PHP de algo como o Javascript no lado do cliente é que o código é executado no servidor, enviando para o lado cliente o código HTML ou informações. O cliente recebe os resultados da execução desse script e a utiliza para renderizar dinamicamente algo a ser exibido no browser. Esse algo pode ser uma página HTML inteira ou parte dela, por exemplo.

Paradigmas de programação

O PHP implementa todas as estruturas de programação, manipulação de dados, suporte para a implementação das estruturas de dados, banco de dados, e, além disso, atualmente conta com um respeitável suporte para a programação orientada a objetos.

O PHP nasceu procedural, mas vem implementando o paradigma da OO já há alguns anos e vai se aproximando cada vez mais do rigor necessário a esse paradigma. Desde a versão 7.4 é possível “tipar” desde parâmetros de funções até atributos de classes (veremos mais adiante).

Aplicações desktop

O PHP provavelmente não é a melhor linguagem para criação de aplicações desktop com interfaces gráficas, mas se você conhece bem o PHP e gostaria de usar alguns dos seus recursos avançados nas suas aplicações do lado do cliente, você pode usar o PHP-GTK para escrever programas assim. Você também tem a possibilidade de escrever aplicações multiplataforma desse jeito. O PHP-GTK é uma extensão do PHP, não disponibilizada na distribuição oficial.

Plataformas

O PHP pode ser utilizado na maioria dos sistemas operacionais, incluindo Linux, Microsoft Windows, Mac OS X, RISC OS e provavelmente outros. O PHP também é suportado pela maioria dos servidores web atualmente. Isso certamente inclui o Apache.

Acesso a banco de dados

Assim como outras linguagens web, o PHP possui suporte a uma ampla variedade de banco de dados. Escrever uma página web consultando um banco de dados é incrivelmente simples usando uma das extensões específicas de um SGBD relacional como o MySql ou usando uma camada de abstração como o PDO que veremos mais adiante.

Flexibilidade

O PHP é conhecido por sua flexibilidade. Com o PHP é possível escolher o sistema operacional, o banco de dados e o servidor web. Do mesmo modo, você pode escolher o paradigma procedural (programação estruturada) ou o paradigma da programação orientada a objetos. Sua flexibilidade vai ficar bem evidente ao longo do curso.

Características

O PHP é uma linguagem de script, interpretada e que roda geralmente no lado servidor independente de plataforma. Trata-se de uma linguagem de código aberto (open source) originalmente criada para produzir páginas web. Possui tipagem dinâmica (falaremos mais adiante). Você pode escrever programas estruturados ou orientados a objetos com PHP. Sua sintaxe se assemelha às linguagens C e C++.

3. Como usar o PHP.

Para desenvolver aplicações web com PHP você precisa de 3 coisas: o próprio PHP, um servidor web e um cliente (browser) web. Você provavelmente já tem um browser e dependendo da configuração do seu sistema operacional, você pode também ter um servidor web (Ex.: Apache no Linux). Você também pode alugar um servidor web. Dessa maneira, você não precisa configurar nada por conta própria, apenas escrever os seus scripts PHP, enviar via FTP para o servidor que você alugou e ver os resultados no seu browser.

Uma outra opção mais simples para os estudos é instalar uma ferramenta que já possua estes requisitos e que seja de fácil instalação e manuseio. Dentre as mais conhecidas estão o XAMPP (Windows/Linux/MacOS/Solaris), o LAMP (Linux) e o WAMP (Windows).

O que usaremos nas aulas?

Em nossas aulas usaremos o XAMPP (<https://www.apachefriends.org/download.html>). Para instalar o XAMPP basta fazer o

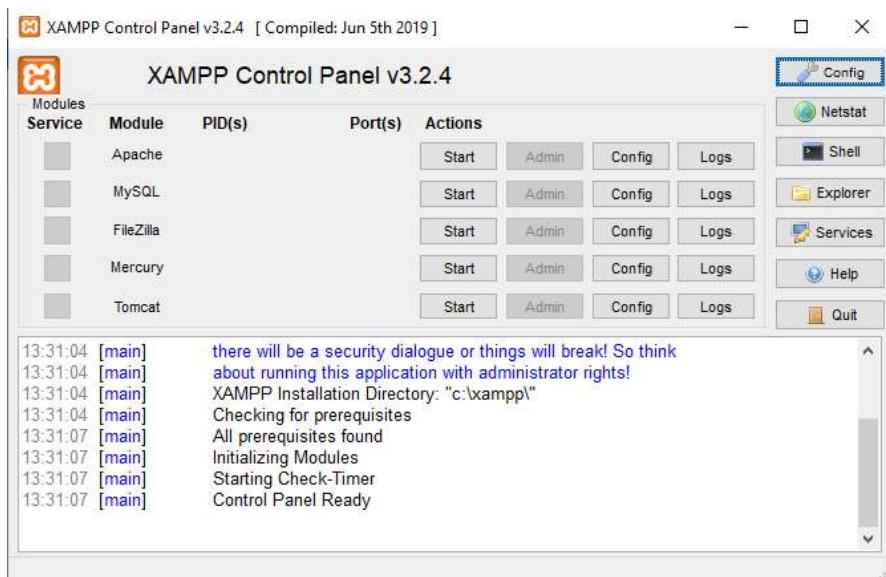
download na página oficial, de acordo com sua versão do Windows e proceder com a instalação que é muito simples e intuitiva. Escolha a versão que vem com o PHP 8.0.2.

Ao instalar o XAMPP você terá em sua máquina o servidor HTTP Apache, o PHP (como um módulo para o Apache) e o banco de dados MySQL (MariaDB). Recomendo fortemente que assistam a vídeo aula abaixo que explica detalhadamente a instalação e a ferramenta:

<https://www.youtube.com/watch?v=kk6G-8F8LA0>

Como utilizar o XAMPP?

Para ter acesso aos serviços do XAMPP, basta digitar a palavra “XAMPP” na busca do Windows e escolher a opção XAMPP Control Panel. Em seguida aparecerá a janela abaixo:



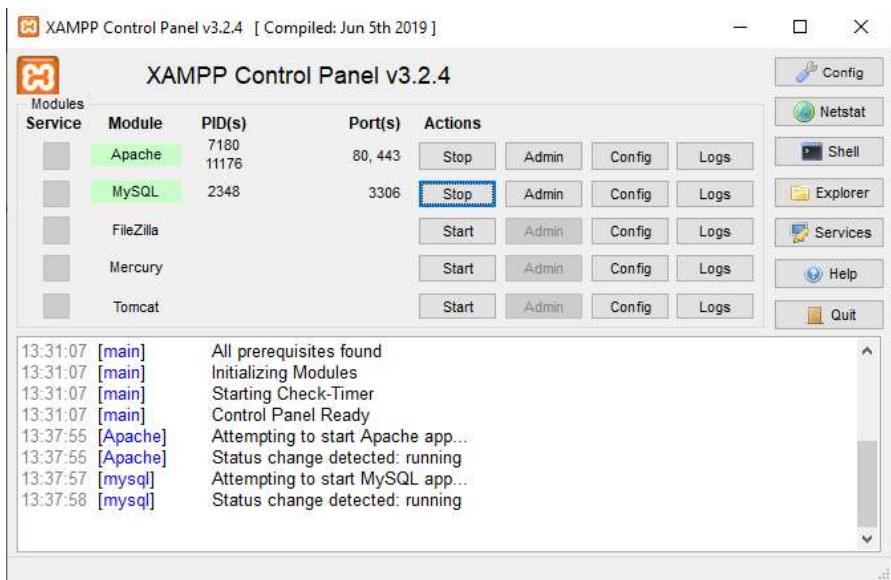
Para que seus scripts PHP sejam interpretados é necessário que o servidor Apache esteja rodando como um serviço. Nesse caso você tem duas opções: Se quiser que esse serviço seja iniciado automaticamente junto com o Windows, você deve marcar a caixinha ao lado esquerdo da palavra Apache. Para evitar problemas nesse início dos estudos, recomendo essa opção tanto para o Apache quanto para o MySQL.

A outra opção consiste em abrir o painel de controle do XAMPP e clicar em start no serviço que se deseja utilizar (Apache e MySQL). Essa é a maneira mais correta de se trabalhar. No entanto, o aluno iniciante por muitas vezes se esquece de executar esse procedimento e fica “sem saber” o porquê de seus scripts não estarem funcionando.

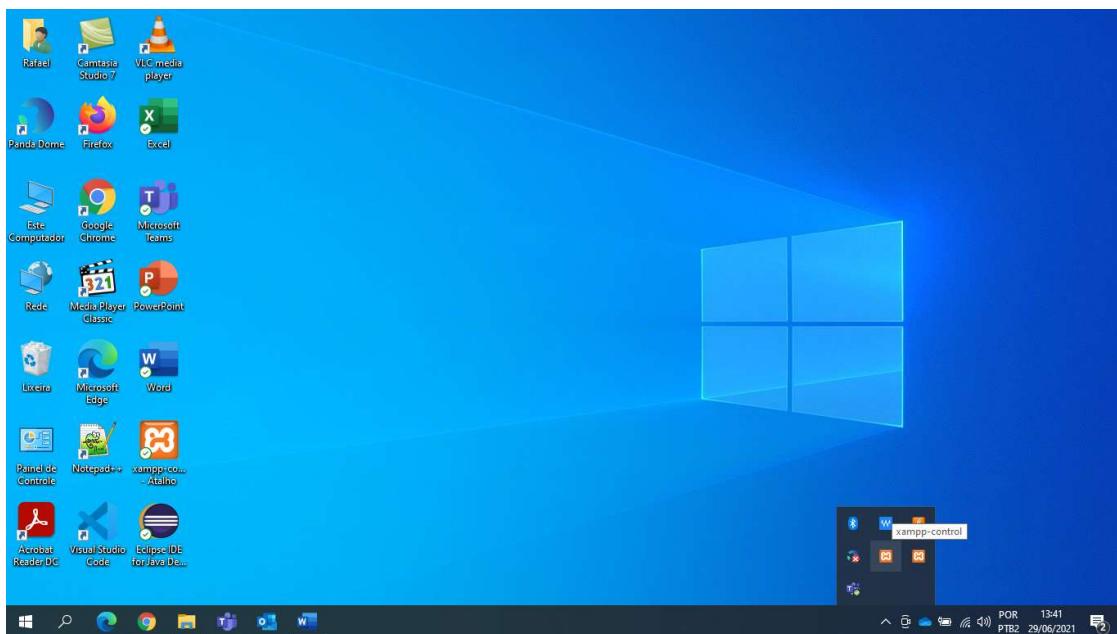
Com os serviços funcionando, o painel de controle exibe ambos na cor verde e com a indicação das portas (80 p/ o Apache e 3306 p/ o MySQL).

Na verdade, na figura abaixo você vai ver que o Apache reserva duas portas (80 e 443). A porta 80 é o padrão (http) enquanto a porta 443 é a porta que implementa aspectos de segurança.

Você pode observar que sites de bancos, por exemplo, usam o https. Um exemplo é o endereço do site do banco do brasil (<https://www.bb.com.br/>). No entanto, não vamos nos preocupar com essa parte agora, mesmo porque isso é competência de quem cuida da infraestrutura de um servidor web e não do desenvolvedor.



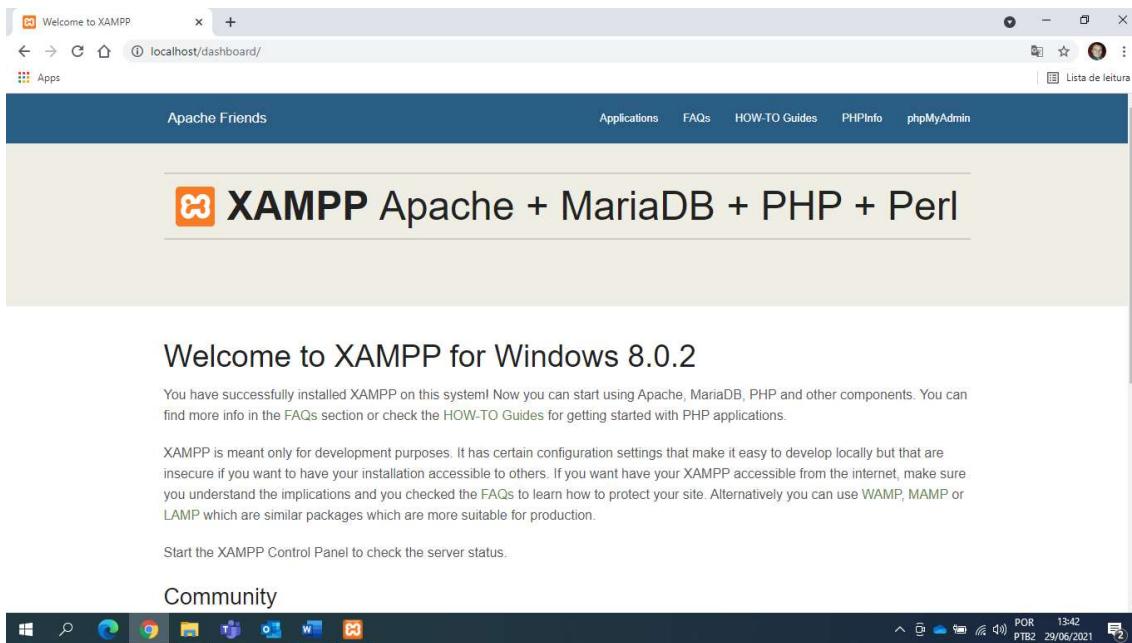
Uma vez que tudo está funcionando corretamente, basta fechar o painel de controle do XAMPP que fica acessível na barra de ferramentas conforme imagem abaixo.



Se encontrar dificuldades para iniciar o apache, significa que o Windows já está usando sua porta padrão (80). Programas como o Skype e o servidor Windows IIS costumam utilizar essa porta. Nesse último caso, siga as instruções do link abaixo:
<https://terminaldeinformacao.com/2018/12/12/como-resolver-problema-da-porta-80-ao-subir-apache-xampp-windows-10/>

Testando a instalação

Para testar se o XAMPP está instalado, acesse um navegador e entre no endereço <http://localhost/>. Se for carregada a página do XAMPP, o programa foi instalado com sucesso!



Onde devo colocar meus arquivos PHP?

No caso do WAMP, seus arquivos ou projetos PHP devem ser colocados na pasta www do referido programa. Se você estivesse usando o XAMPP, por exemplo, a pasta seria htdocs. Somente arquivos colocados salvos dentro da pasta correta serão executados pelo servidor PHP. Isso também vale para arquivos .html. Veja alguns exemplos:

Endereço Físico	Endereço no Navegador
C:\xampp\htdocs\teste.php	http://localhost/teste.php
C:\xampp\htdocs\projeto1\teste.php	http://localhost/projeto1/teste.php
C:\xampp\htdocs\cefet\alunos\cadastro.php	http://localhost/cefet/alunos/cadastro.php
C:\xampp\htdocs\cefet\index.php	http://localhost/cefet/
C:\xampp\htdocs\cefet\teste.html	http://localhost/cefet/teste.html

Por que usar o Apache?

Sempre que você digita um endereço em seu browser(navegador), o mesmo busca pelo recurso solicitado em um servidor web. Mesmo quando você contrata um serviço de hospedagem na web, o servidor em questão geralmente é o Apache.

No nosso caso, quando digitamos o endereço <http://localhost/projeto1/teste.php>, o navegador procura dentro da pasta htdocs do XAMPP o diretório projeto1 e dentro desse diretório o arquivo teste.php. Logo, nosso localhost (servidor local) é justamente a pasta htdocs do XAMPP. Lá é o ambiente propício p/ que nossos scripts php sejam interpretados.

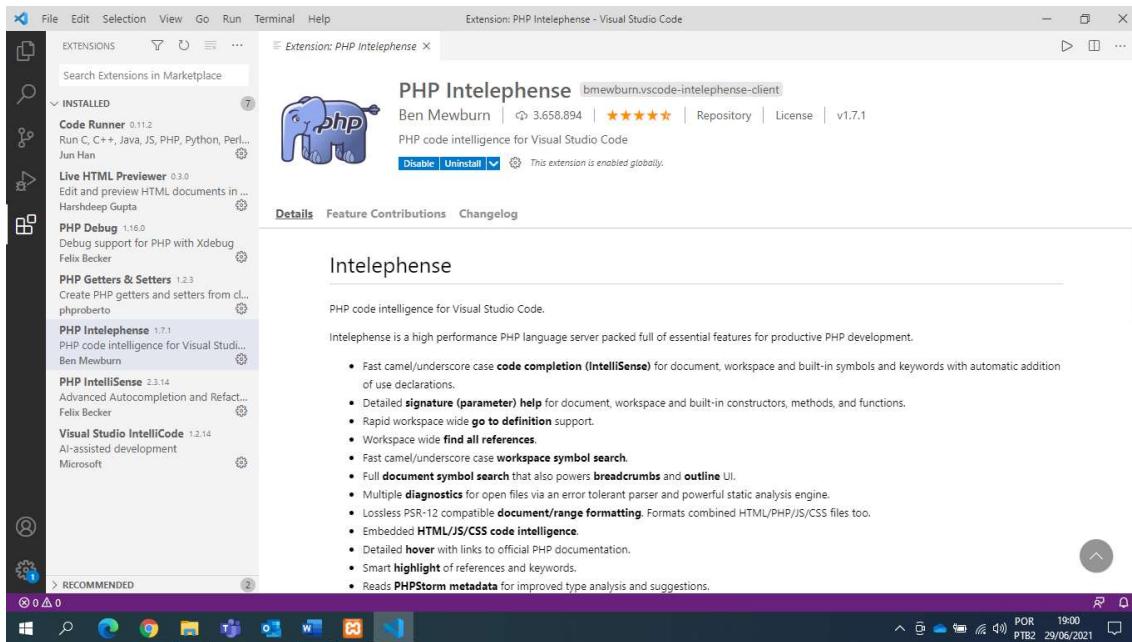
Ao longo do curso você vai perceber que é lá que colocamos também nossos arquivos .html, .css e .js entre outros. Tudo fica no servidor!

Dentro da pasta C:\xampp\apache\bin você pode encontrar o arquivo httpd.conf. Esse arquivo contém as configurações do servidor Apache. Da mesma forma, dentro da pasta C:\xampp\php você vai encontrar o arquivo php.ini com as configurações do PHP. Obviamente você não deve mexer nesses arquivos por enquanto.

Que ferramenta utilizaremos para escrever nossos scripts PHP?

Não usaremos nenhuma IDE visto que o foco nesse momento é o aprendizado da linguagem. As IDEs muitas vezes facilitam tanto a ponto de atrapalhar o desenvolvimento do aluno. No nosso caso utilizaremos um Editor de textos próprio para o desenvolvimento e que ajuda demais na produtividade.

O Editor escolhido para nossas aulas foi o Visual Studio Code que pode ser baixado no endereço <https://code.visualstudio.com/download>. Essa ferramenta auxilia a completar códigos, sinaliza a sintaxe com cores e fontes de acordo com o tema escolhido entre outras facilidades. Instalando as extensões corretas, o VSCode indica até mesmo erros de compilação. Veja abaixo (lado esquerdo) algumas extensões que você DEVE instalar.

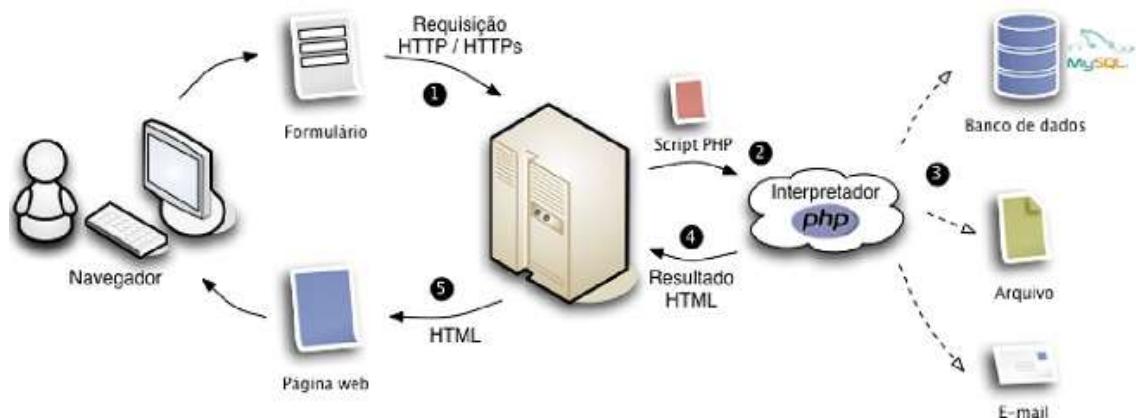


Recomendo fortemente que assistam a vídeo aula abaixo explicando a ferramenta, instalação e suas características mais relevantes:
<https://www.youtube.com/watch?v=xv27VHgqF7Y&list=PLZun-PtGBZCtks8RtUbrmim5o2cS7GkVn>

4. Arquitetura Cliente Servidor

Sempre que trabalhamos desenvolvimento web, estamos, na verdade, trabalhando na arquitetura conhecida como cliente/servidor. Nesse caso, nós somos o cliente e o servidor é quem recebe nossas demandas/pedidos/requisições (também conhecido como request).

O cliente, na verdade, é nosso browser e o servidor pode ser um servidor local como o Apache da ferramenta XAMPP ou um servidor remoto. Não importa. Observe atentamente a figura abaixo e em seguida faremos algumas considerações.



Toda vez que o cliente digita um endereço no browser, na verdade ele está solicitando um recurso ao servidor e esperando uma resposta. Nesse caso, ele está enviando uma requisição ao servidor. Essa requisição é conhecida como request. A resposta esperada pelo cliente e enviada pelo servidor é conhecida como response. Cliente e servidor se entendem por que falam a mesma “língua” ou o mesmo “idioma” que, nesse caso é o protocolo HTTP ou o HTTPS.

Tecnologias lado cliente e tecnologias lado servidor

Lado cliente

HTML, CSS e JavaScript

Essas tecnologias são interpretadas pelo navegador. Vamos supor que você crie numa pasta dev do seu raiz (c:/dev) um arquivo chamado teste.html, que utiliza um arquivo teste.js e um arquivo estilos.css.

Se você abrir esse arquivo teste.html no navegador, tudo vai funcionar normalmente mesmo sem ter um servidor sendo apontado ou rodando em sua máquina. Isso é possível por que essas tecnologias são interpretadas pelo browser.

O mesmo não pode ser feito em relação a um arquivo .php, .aspnet etc. Esses arquivos precisam estar em um ambiente em que eles possam ser interpretados. Esse ambiente é um servidor web configurado para tal. Essas tecnologias NÃO são compreendidas pelo navegador!

Lado servidor

PHP, Aspnet, Java, Python, Perl etc.

O banco de dados (No nosso caso o MySQL) também fica no servidor. Pode ser o mesmo servidor ou um servidor a parte.

Por que eu preciso colocar meus arquivos HTML, CSS e JavaScript em um servidor web ou na pasta htdocs do XAMPP?

Quando você digita um endereço como www.localhost/minha-aplicacao/teste.html, na verdade você está obtendo uma cópia do arquivo teste.html que está na pasta htdocs/minha-aplicacao do XAMPP. Essa cópia é recebida pelo navegador em forma de texto e a página em questão é renderizada. Se o arquivo teste.html

referencia arquivos .js e .css, esses também serão trazidos em cópia para o cliente e a página será renderizada de acordo com as instruções javaScript contidas no arquivo.js e os estilos definidos no arquivo.css.

Quando você digita um endereço como www.localhost/minha-aplicacao/teste.php, ocorre algo parecido. O arquivo .php devolve HTML ou informações (Um arquivo XML ou JSON por exemplo) para que um HTML seja renderizado pelo browser. Lembre-se! O browser não comprehende PHP. Ele só comprehende HTML, CSS e JavaScript.

Em todo esse esquema desenhado acima, o PHP é quem interpreta as requisições do cliente e trabalha para devolver uma resposta HTML que conteíble tudo o que foi solicitado, incluindo informações que estejam em um banco de dados por exemplo. Se a linguagem do lado servidor fosse java, um ou mais arquivos java fariam o mesmo tipo de trabalho.

5. Delimitadores, comentários e o primeiro arquivo.

O interpretador do servidor identifica um código PHP por meio de 2 delimitadores que indicam o início e o fim do programa ou trecho de programa. Todo código PHP, dentro de seu próprio arquivo ou dentro de outro, deve ser iniciado por **<?php** (ou **<?**) e terminado por **?>**. Veja um exemplo:

```
<html>
<head><title>Delimitadores PHP</title></head>
<body>
    <h1>Exemplo de uso de delimitadores PHP - Exemplo 1</h1>
    <?php
        //Inserção de código PHP - Exemplo 1
        echo "Olá Mundo!";
    ?>

    <h1>Exemplo de uso de delimitadores PHP - Exemplo 2</h1>
    <?
        //Inserção de código PHP - Exemplo 2
        echo "Sem usar a palavra php";
    ?>
</body>
</html>
```

Observe que podemos fazer pequenas inserções de código PHP no meio do código HTML. Para que isso funcione você deve salvar o arquivo com a extensão .php em vez de .html.

Apenas para ilustrar, no exemplo acima utilizamos uma função PHP para impressão, chamada **echo**.

Exercício de Fixação:

- 1) Escreva o código acima, salve em um arquivo chamado ola.php dentro da pasta aula01 e teste em seu navegador acessando o endereço www.localhost/aula01/ola.php. Não esqueça que a pasta aula01 e seu conteúdo devem estar dentro da pasta htdocs do XAMPP!

Comentários

Existem 2 tipos de comentários em PHP. Comentário de uma linha e comentário de bloco. O PHP suporta comentários no estilo 'C', 'C++' e shell do Unix shell (estilo Perl). Veja exemplos:

```
<?php
    echo 'Isto é um teste'; // Estilo de comentário de uma linha em c++
    /* Este é um comentário de múltiplas linhas
       ainda outra linha de comentário */
    echo 'Isto é ainda outro teste';
    echo 'Um teste final'; # Este é um comentário de uma linha no estilo shell
?>
```

6. Variáveis.

Definição de nome

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**.

Os nomes de variável seguem as mesmas regras como outros rótulos no PHP. Um nome de variável válido se inicia com uma letra ou sublinhado, seguido de qualquer número de letras, algarismos ou sublinhados. Em uma expressão regular isto poderia ser representado desta forma: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

Uma boa prática de programação é usar o padrão camel case para definição de nomes de variáveis compostos. Exemplo: \$minhaBicicleta, \$anoDoMeuNascimento. Utilizem sempre nomes significativos e que possam indicar o tipo de valor que está sendo armazenado numa variável. Se você está guardando o preço de custo de um produto, faz sentido que o nome da variável seja \$precoDeCusto e não \$xyz, por exemplo.

Case sensitive

A linguagem PHP é “case sensitive”, uma letra maiúscula é diferente de uma letra minúscula. Por este motivo, tradicionalmente os programadores utilizam letras minúsculas para compor os nomes das variáveis (exceto nos nomes compostos) e letras maiúsculas para compor o nome das constantes.

Veja alguns exemplos com variáveis:

```
<?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 (extended) 228
?>
```

7. Tipos de Dados suportados.

Alguns programadores costumam dizer que o PHP não define tipo para as variáveis ou que o PHP não tem tipo de dados, mas existem tipos sim. Na verdade, o PHP usa uma checagem dinâmica de tipos, por isso, uma única variável pode receber valores de tipos diferentes em momentos diferentes da execução do script. Por esse motivo, não é necessário declarar as variáveis. Consequentemente não é necessário declarar o tipo delas para poder utilizá-las.

O interpretador PHP (no servidor) detectará o tipo de variável por meio da verificação do conteúdo durante toda a execução do programa. Dentro desse conceito, o PHP suporta os seguintes tipos de dados:

- Tipos básicos: **inteiros**, **booleanos**, **numéricos** e **string**;
- Tipos compostos: **array** e **object**;
- Tipos especiais: **resource** (referência para um recurso (arquivo, BD etc.) e **NULL** (representa ausência de valor. Só admite a constante NULL);
- Pseudo-tipos: **mixed** (Aceita vários tipos (básicos, compostos e especiais)), **number** (Número que pode ser integer ou float) e **callback** (referências para funções ou métodos).

Simulação do tipo booleano

O PHP possui um tipo booleano especificado, mas pode avaliar expressões e retornar verdadeiro ou falso simulado por meio do tipo inteiro. Neste caso, o valor zero (0) representa o estado falso e qualquer valor diferente de zero representa o estado verdadeiro.

Veja um exemplo de atribuições e tipos de dados:

```
<?php
//NUMEROS INTEIROS (integer ou long)
$idade = 40; //integer
$minhaIdade = 40; //integer no estilo camel case
$num = -123; //integer (decimal negativo)
$num = 0173; //integer (123 na base octal)
$num = 0x7B; //integer (123 na base hexadecimal)

//NUMEROS REAIS (float ou double)
$precoDaCamisaDoFlamengo = 1.99; //real positivo estilo camel case
$valor = -7.90; // real negativo - O mesmo que -7,90
.

//BOLEANOS (true ou false)
$maiorDeIdade = true; //boolean estilo camel case
$x = false; //boolean
$dezEMaiorQueNove = ( 10 > 9 ); //boolean (resultado de uma expressão booleana)

//CADEIA DE CARACTERES (string)
$nome = 'Fulano'; //string (com apóstrofo)
$nome = "Fulano"; //string (com aspas)
//DICA: Acostume-se a usar apóstrofo (aspas simples) para trabalhar com strings
?>
```

Exercício de fixação:

- 1) Escreva o código acima e teste algumas saídas no navegador usando a função **echo**. Não precisa usar parênteses na função echo.

8. Apóstrofos, Aspas e Caracteres de Controle

Uma **string** com apóstrofo (aspas simples) não expande (exibe) o valor de uma variável e nem interpreta caracteres de controle como o “\n” por exemplo. Já uma string com aspas, expande o valor de uma variável e interpreta o caractere de controle. Exemplo:

```
$nome = 'Rafael';
echo 'Olá, $nome!'; //SAÍDA: Olá, $nome!
echo "Olá, $nome!"; //SAÍDA: OLÁ, Rafael!

//TESTE PARA CONSOLE
echo 'Um \nDois'; //Não pula linha
echo "Um \nDois"; //Pula linha
```

Isso acontece porque em strings com apóstrofo, o interpretador não procura por variáveis em seu interior. Logo, se sua string não tiver variáveis, use apóstrofo. Assim o interpretador não perderá tempo analisando a string. O exemplo com o “\n” só teria validade em um terminal do servidor. Navegadores não reconhecem o “\n”.

Exercício de fixação:

- 1) Escreva o código acima e teste algumas saídas no navegador.

Caracteres de controle

Caracteres de controle são caracteres constantes predefinidos pela linguagem e utilizados especificamente para realizar uma ação não prevista pela **string** impressa. Veja a tabela de caracteres de controle da impressão:

Sintaxe	Resultado
\n	Nova Linha
\r	Retorno de carro
\t	Tabulação horizontal
\\\	Imprime a barra (\)
\\$	Imprime o símbolo \$
\'	Imprime um apóstrofo
\"	Imprime uma aspa

Como foi dito anteriormente, a funcionalidade destas constantes depende de o navegador reconhecer-las ou não. O “\n”, por exemplo, não é reconhecido pelos navegadores, sendo necessária a utilização da tag
 para realizar uma quebra de linha. Veja mais um exemplo:

```
$salario = 2000.00;
echo "O salário é R\$ ".\$salario;

//RESULTADO: O salário é R$ 2000.00
```

Exercício de fixação:

- 2) Escreva o código acima e teste algumas saídas no navegador.

9. Interpretação de variáveis numa string

Veja alguns exemplos autoexplicativos e observe os comentários representando a saída:

```
$cerveja = 'Antartica';

echo "Rafael estava com sede e bebeu doze $cervejas";
//SAÍDA: Rafael estava com sede e bebeu doze $cervejas (INCORRETO)

echo "Rafael estava com sede e bebeu doze ${cerveja}s";
//SAÍDA: Rafael estava com sede e bebeu doze Antarticas (CORRETO)

echo "Rafael estava com sede e bebeu doze $cerveja's";
//SAÍDA: Rafael estava com sede e bebeu doze Antartica's (CORRETO, MAS INDESEJADO)
```

Dica: Recomenda-se usar chaves.

Exercícios de fixação:

- 1) Teste o código acima em seu navegador.

10. Concatenação de strings

Para concatenar strings usa-se o ponto. Veja o exemplo abaixo:

```
$a = 'João';
$b = 'Olá, ' . $a;
$c = '<b>' . $b. '</b>';
//Acrescenta a tag <br /> ao final.
$c .= '<br />';
```

11. Impressão de strings

Para impressão de strings podemos usar as funções **echo** e **print**. Ambas não necessitam de parênteses.

- **echo**
 - Não retorna nada;
 - Pode imprimir várias strings, separadas por vírgula, desde que não seja usado parênteses.
- **print**
 - Retorna sempre 1 (um);
 - Imprime somente uma string.

Veja alguns exemplos:

```
$usuario = 'Juca';
//Ao usar aspas, o interpretador identifica as variáveis e exibe seu valor
echo "Olá, $usuario <br/>";

//Usando echo com vírgulas
echo "Idade: ',20, ' Altura: ',1.80, '<br />'; //Numeros são convertidos para strings
//Usando echo com ponto (concatenação)
echo 'Idade: ' .20. ' Altura: ' .1.80. '<br />'; //Numeros são convertidos para strings e concatenados

print "Olá, $usuario <br/>";//A variável interpretada e seu valor será exibido
print 'Idade: ' .20. ' Altura: ' .1.80. '<br />'; //Numeros são convertidos para strings e concatenados
```

No exemplo acima, o segundo echo não faz muito sentido uma vez que estamos tratando de valores literais. O mesmo vale para o terceiro echo que em versões mais recentes da linguagem pode nem funcionar.

Uma sugestão mais eficaz de teste segue abaixo:

```
$idade = 20;
$altura = 1.81;
echo "Idade: ".$idade." Altura: ".$altura." <br>";
echo "Idade: {$idade} Altura: {$altura} <br>";
//ou
echo "Idade: 20 Altura: 1.81 <br>";
```

Exercícios de fixação:

- 1) Teste os códigos acima.

12.Impressão com término da execução

A função **die** é usada para terminar a execução do script exibindo uma mensagem.
Sintaxe:

```
void die ( [string $message] );
```

Exemplo:

```
die ( 'Acesso negado.' );
```

Na verdade, a função **die** é idêntica à função **exit**, que possui as seguintes sintaxes:

```
void exit ( [string $message] );
void exit ( int $status );
```

Seu uso com um valor do tipo **int**, não realiza impressão e apenas indica o código do erro com que deseja terminar.

Exercícios de fixação:

- 1) Teste usar um echo depois de uma função die ou exit.

13.Conversão de tipo

A transformação de tipos no PHP pode ser feita de 3 maneiras:

- Automaticamente (cast automático ou coerção);

- Explícitamente pelo usuário, utilizando o typecast;
- Utilizando a função settype().

Automaticamente

Quando ocorrem determinadas operações entre dois valores de tipos diferentes, o PHP converte o conteúdo de um deles automaticamente para adequar o resultado sem, porém, alterar o valor contido no operando. O tipo no qual os valores dos operandos serão convertidos tem a ordem de precedência mostrada na tabela abaixo:

Ordem	Um dos operandos é:	O outro é transformado em:
1	Real (float)	Real (float)
2	Inteiro	Inteiro

Uma string é convertida em um número real ou inteiro quando começa por um número de acordo com a análise realizada pelo interpretador e, no caso de começar com letra, será convertida em zero. Veja exemplos:

```
$num = 1 + '10.5';           //float (11.5)
$num = 1 + '-1.3e3';         //float (-1299)
$num = 1 + 'teste-1.3e3';    //int (1)
$num = 1 + 'teste3';         //int (1)
$num = 1 + '10 porquinhos'; //int (11)
$num = 4 + '10.2 porquinhos'; //float (14.2)
$num = '10.0 vacas ' + 1;   //float (11.0)
$num = '10.0 vacas ' + 1.0; //float (11.0)
```

O interpretador descobre e aceita sinais precedendo números, pontos que representam as casas decimais e a letra e ou E como exponencial. Veja o exemplo:

```
$valor = 3e2; //Valor recebe 300, que é 3 x 10 elevado ao quadrado
echo $valor; //Imprime 300
```

Exercícios de fixação:

- 1) Escreva os códigos acima alternando as atribuições com a função echo para verificar o resultado das conversões automáticas.

Explicitamente pelo usuário, usando o typecast

O usuário deve explicitar o tipo de dado de forma semelhante às linguagens Java e C, escrevendo o novo tipo entre parênteses antes do valor a ser convertido. Veja alguns exemplos:

```
$texto = '10';
$num = (integer) $texto;
$texto = (string) $num;

$valor = 3.2; //cria a variável do tipo real
echo $valor; //Vai imprimir 3.2
$valor = (int) $valor; //converte para inteiro
echo $valor; //Vai imprimir 3

$b = 0;
$num = (bool) $b; // $num recebe false
$num = (bool) 45; // $num recebe verdadeiro
$num = (bool) -5; // $num recebe verdadeiro
```

Exercícios de fixação:

- 1) Teste as conversões acima utilizando a função echo para verificar se os valores foram preservados após a conversão.
- 2) Faça testes com as funções floatval, intval etc

As palavras-chave aceitas para compor o cast são as listadas na tabela abaixo:

Palavra-chave	Converte para
int ou integer	Inteiro
real, double ou float	Real com ponto flutuante
bool ou boolean	Boleano
string	Cadeia de caracteres (string)
array	Vetor indexado (array)
object	Objeto
binary	String binária
unset	NULL

Usando a função settype

A função settype() converte uma variável no tipo especificado. Veja os exemplos:

```
$num = 200;

settype($num, 'int');
settype($num, 'integer');
settype($num, 'bool');
settype($num, 'boolean');
settype($num, 'string');
settype($num, 'binary');
settype($num, 'array');
settype($num, 'object');
```

Exercícios de fixação:

- 1) Escreva o código acima intercalando com a função echo para verificar se houve mudanças na conversão.

14.Verificação de tipo de uma variável

Como já foi analisado anteriormente, a tipagem utilizada pelo PHP é dinâmica e, portanto, nem sempre é possível saber qual é o tipo de uma variável em determinado instante. Existem 2 maneiras de verificar o tipo das variáveis:

- Através da função `gettype()`;
- Através de funções que testam o tipo da variável.

Função `gettype()`

A função `gettype()` retorna o tipo da variável testada indicado por uma das seguintes strings: “integer”, “double”, “string”, “object”, “unknown type”. Veja exemplos:

```
echo gettype( 7 ); // 'int'  
echo gettype( 7.0 ); // 'float'  
echo gettype( 'sete' ); // 'string'  
echo gettype( NULL ); // 'NULL'
```

Exercícios de fixação:

- 1) Teste o código acima.

Funções que testam o tipo da variável

Algumas funções retornam verdadeiro se a variável for do tipo perguntado e falso em caso contrário. São elas:

`is_int(mixed $var), is_integer(mixed $var), is_long(mixed $var), is_real(mixed $var), is_float(mixed $var), is_bool(mixed $var), is_string(mixed $var), is_array(mixed $var), is_object(mixed $var), is_numeric(mixed $var), is_resource(mixed $var), is_null(mixed $var).`

Veja exemplos:

```
$valor = 55;  
  
if( is_int( $valor ) ){  
    echo "Inteiro";  
} else{  
    echo "Não inteiro";  
}
```

Essa estrutura de comparação também é conhecida como seleção.

Exercícios de fixação:

- 1) Faça alguns testes com as funções citadas acima.

15.Exibindo a estrutura de uma variável

Através da função **var_dump** podemos exibir a estrutura de uma variável.
Sintaxe: void var_dump(mixed \$expression, [, mixed \$...]);
Veja um exemplo:

```
$a = 1.99;  
$b = false;  
  
var_dump( $a, $b );  
//SAÍDA:  
//float 1.99  
//boolean false
```

Exercícios de fixação:

- 1) Faça testes com var_dump.

16.Verificando se uma variável possui um valor

Existem dois tipos de funções para verificar se uma variável foi criada ou inicializada (usualmente conhecida como ligada): a função **isset()** e a função **empty()**.

isset()

A função **isset()** retornará **verdadeiro** se a variável existir (estiver ligada, ainda que com uma string vazia ou o valor zero) e **falso** caso contrário.

Veja o exemplo de um campo de um <form> não preenchido. O script recebe o campo não inicializado. O isset vai verificar se esse campo veio trafegando pela rede independentemente de estar preenchido ou não.

```
if( isset( $_POST['cidade'] ) ){ //verifica se a variável está "ligada"  
    echo "O campo cidade foi preenchido";  
}else{  
    echo "O campo cidade não foi preenchido";  
}
```

Veja um exemplo com a variável \$b:

```
//if( null == $b ) gera erro de variável não definida  
if( isset( $b ) ) echo '$b existe.';  
else echo '$b não existe ou é nula.'; // << resultado  
  
$b = null;  
  
if( isset( $b ) ) echo '$b existe.';  
else echo '$b não existe ou é nula.'; // << resultado
```

Exercícios de fixação:

- 1) Faça os testes acima com a função isset().

empty()

A função **empty()** retornará **verdadeiro** se a variável não contiver nenhum valor (não estiver ligada), possuir valor 0 (zero) ou uma string vazia. Caso contrário, retornará **falso**. Veja exemplos:

```
$a = 0;  
  
if( empty( $a ) ) echo '$a está vazia.'; // << resultado  
else echo '$a está preenchida.\n';  
  
$a = '';  
  
if( empty( $a ) ) echo '$a está vazia.'; // << resultado  
else echo '$a está preenchida.\n';  
  
$a = null;  
  
if( empty( $a ) ) echo '$a está vazia.'; // << resultado  
else echo '$a está preenchida.\n';  
  
$a = ' '; // espaço é um caractere  
  
if( empty( $a ) ) echo '$a está vazia.';  
else echo '$a está preenchida.\n'; // << resultado
```

Exercícios de fixação:

- 1) Faça os testes acima.

17. Constantes

Por definição, uma vez associado um valor a uma constante, o mesmo não poderá mais ser alterado. Uma constante só pode ter valores escalares, sendo proibidos arrays e objetos. Constantes tem escopo global.

Para definir as constantes, utiliza-se a função **define()**, que espera receber dois parâmetros: o nome e o valor da constante. Após sua execução, define() retornará verdadeiro se for bem sucedida e falso caso contrário. Veja um exemplo:

```
define( 'PI' , 3.1416 ); //float  
define( 'TITULO' , 'Comprimento da circunferência' ); //string  
$raio = 3;  
$circunferencia = 2 * PI * $raio;  
echo TITULO . " : " . $circunferencia;  
//SAÍDA: Comprimento da circunferência : 18.8496
```

Exercícios de fixação:

- 1) Teste o código acima.

Constantes predefinidas

O ambiente do servidor PHP possui algumas constantes predefinidas indicando, por exemplo, a versão do PHP, o Sistema Operacional do servidor, o arquivo em execução etc. Veja exemplos:

- PHP_EOL: Fim de linha;
- PHP_VERSION: Versão do PHP;
- PHP_INT_MAX: O maior valor inteiro.

É possível ver todas as constantes predefinidas através do comando abaixo:

```
echo "Usando a função get_defined_constants(): <br/>";
print_r( get_defined_constants() );
```

O resultado fica um pouco confuso.

Pressionando as teclas CTRL + U, a exibição fica mais palatável. Veja:

18. Operadores de atribuição

Existe um operador básico de atribuição e diversos derivados dele, mas todos sempre retornam o valor atribuído.

No caso dos operadores derivados de atribuição, a operação é feita antes da atribuição, sendo atribuído o resultado da operação ao primeiro operando especificado à esquerda da atribuição. A atribuição é sempre por valor, não por referência (veremos adiante). Veja exemplos:

```
$x = 10;  
$x += 5; // 15  
$x *= 2; //30  
$x /= 3; //10  
$x -= 5; //5  
$x %= 2; //1  
  
$s = 'Olá';  
$s .= ' Mundo!'; //Olá Mundo
```

Exercícios de fixação:

- 1) Teste o código acima.

19.Operadores pós fixos

```
$i = 1;  
$i++; //Incrementa em 1 e $i passa a valer 2  
$i--; //Decrementa em 1 e $i passa a valer 1
```

20.Operadores prefixos

```
$i = 1;  
++$i; //Incrementa em 1 e $i passa a valer 2  
--$i; //Decrementa em 1 e $i passa a valer 1
```

21.Cuidados com os pós fixos

Nas operações de atribuição é necessário ter um certo cuidado com o operador pós fixo já que ele faz a atribuição antes de incrementar/decrementar seu valor. Isso pode gerar resultados indesejados. Observe atentamente o exemplo abaixo:

```
$a = 5;  
$b = $a++; //CUIDADO: $b recebe $a (5) e só depois $a passa a valer 6  
  
$a = 5;  
$b = ++ $a; // $a é incrementado antes de ser atribuído a $b. Ambos passam a valer 6
```

Exercícios de fixação:

- 1) Teste o código acima

22.Operadores aritméticos

Os operadores aritméticos são aplicados somente aos tipos numéricos (integer ou float). Se forem de outro tipo, terão seus valores convertidos automaticamente ou

explicitamente (para o tipo numérico) de acordo com as regras descritas no item 12, antes da realização da operação. Segue uma tabela com os operadores aritméticos:

Operador	Ação
-	Subtração, também menos unário
+	Adição, também mais unário
*	Multiplicação
/	Divisão
%	Modulo da Divisão (resto da divisão)
**	Exponenciação (5^2 é representado como $5^{**} 2$)

Veja um exemplo com o operador %:

```
$a = 5;  
$restoDaDivisaoPorDois = $a % 2; // (5 / 2) = 2 e o resto é 1  
echo $restoDaDivisaoPorDois; // Vai imprimir 1
```

Exercícios de fixação:

- 1) Teste o código acima.

23. Operadores de comparação e Operadores lógicos

Operadores de comparação

Os operadores de comparação são também conhecidos como **operadores relacionais**. O resultado de uma comparação é **sempre um valor booleano**. Qualquer valor diferente de zero é considerado **verdadeiro** e zero, **falso**. Veja alguns exemplos:

```
if( $a == $b ) echo '$a e $b são iguais.';  
if( $a != $b ) echo '$a e $b são diferentes.';  
if( $a > $b ) echo '$a é maior que $b.';  
if( $a < $b ) echo '$a é menor que $b.';  
if( $a >= $b ) echo '$a é maior ou igual a $b.';  
if( $a <= $b ) echo '$a é menor ou igual a $b.';  
  
if( $a === $b ) echo '$a e $b possuem valores e tipos iguais.';  
if( $a !== $b ) echo '$a e $b possuem valores e tipos diferentes.';
```

Em tempo: Na última linha, considere ‘\$a e \$b possuem valores e/ou tipos diferentes.’.

Exercícios de fixação:

- 1) Crie um exemplo, defina valores para \$a e \$b e faça testes com os operadores mostrados acima.

Operadores lógicos

São muito usados para operar números inteiros representando valores booleanos, mas se aplicam a qualquer tipo.

Veja a tabela:

Operadores Lógicos		
Exemplo	Nome	Resultado
\$a and \$b	E	Verdadeiro (TRUE) se tanto \$a quanto \$b são verdadeiros.
\$a or \$b	OU	Verdadeiro se \$a ou \$b são verdadeiros.
\$a xor \$b	XOR	Verdadeiro se \$a ou \$b são verdadeiros, mas não ambos.
! \$a	NÃO	Verdadeiro se \$a não é verdadeiro.
\$a && \$b	E	Verdadeiro se tanto \$a quanto \$b são verdadeiros.
\$a \$b	OU	Verdadeiro se \$a ou \$b são verdadeiros.

A razão para as duas variantes dos operandos "and" e "or" é que eles operam com precedências diferentes.

Observe os exemplos e entenda a diferença de usar **&&** ou and e de usar **||** ou or:

```
<?php

// foo() nunca será chamada como estes operadores são short-circuit
$a = (false && foo());
$b = (true || foo());
$c = (false and foo());
$d = (true or foo());

// "||" tem maior precedência que "or"
$e = false || true; // $e will be assigned to (false || true) which is true
$f = false or true; // $f will be assigned to false
var_dump($e, $f);

// "&&" tem maior precedência que "and"
$g = true && false; // $g will be assigned to (true && false) which is false
$h = true and false; // $h will be assigned to true
var_dump($g, $h);
?>
```

O exemplo acima produzirá a seguinte saída:

```
bool(true)
bool(false)
bool(false)
bool(true)
```

Exercícios de fixação:

- 1) Teste o código acima.

24. Operador ternário

O operador ternário trabalha da seguinte forma:

expressão ? valor_caso_verdadeiro : valor_caso_falso;

Geralmente este operador é utilizado para operações de atribuição com base em resultado de alguma expressão booleana.

Veja um exemplo:

```
$c = ( $a > $b ) ? 'Maior' : 'Menor ou igual';

$tratamento = ( 'M' == $sexo ) ? 'Sr.' : 'Sr*';
```

Exercícios de fixação:

- 1) Teste o código acima.

25.Escopo das variáveis

O escopo de uma variável é o contexto onde ela foi definida. A maior parte das variáveis do PHP tem somente escopo local.

Entretanto, com as funções definidas pelo usuário, um escopo local é introduzido. Qualquer variável utilizada dentro da função é, por default, limitada dentro do escopo local da função. Observe atentamente o exemplo:

```
function teste(){
    $z = 5;
    echo "Durante a execução da função: $z<br />";
}

$z = 3;
echo "Antes da execução da função: $z<br />"; // VAI IMPRIMIR: 3
teste(); // VAI IMPRIMIR: 5
echo "Depois da execução da função: $z<br />"; // VAI IMPRIMIR: 3
```

26.Variáveis globais

As variáveis globais são declaradas fora das funções e reconhecidas pelo programa inteiro; assim podem ser utilizadas em qualquer ponto do código do programa, inclusive dentro das funções.

Caso existam uma variável global e uma local com o mesmo nome, a variável com o escopo local é acessada prioritariamente ali. Veja um exemplo:

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

function Teste()
{
    echo $a; /* referencia uma variável do escopo local (não definida) */
}

Teste();
?>
```

Este script não produz nenhuma saída porque a instrução echo refere-se a uma versão local da variável \$a, e ela não tem nenhum valor assimilado nesse escopo.

Essa é uma pequena diferença da linguagem C quando variáveis globais são automaticamente disponíveis para funções sem sobreescriver uma eventual definição local. Isto causa problemas quando as pessoas mudam inadvertidamente uma variável global.

No PHP, as variáveis globais precisam ser declaradas globais dentro de uma função se ela vai ser utilizada naquela função.

O modificador de escopo ‘global’

Uma variável de escopo global pode ser utilizada no interior de uma função (escopo local); porém, é necessário explicar esta intenção com a aplicação do modificador global, que pode conter diversas variáveis, separadas por vírgulas. Veja o exemplo abaixo:

```
function testaGlobal(){
    global $valor, $res;
    $valor +=1;
    $res +=1;
    echo 'Valor (dentro da função): '.$valor.', Resultado (dentro da função): '.$res.'  
<br/>';
}

$valor = 100;
$res = 10 * $valor;
echo 'Valor (antes da função): '.$valor.', Resultado (antes da função): '.$res.'  
<br/>';
testaGlobal();
echo 'Valor (depois da função): '.$valor.', Resultado (depois da função): '.$res.'  
<br/>';
```

Saída produzida:

```
Valor (antes da função): 100, Resultado (antes da função): 1000
Valor (dentro da função): 101, Resultado (dentro da função): 1001
Valor (depois da função): 101, Resultado (depois da função): 1001
```

Exercícios de fixação:

- 1) Teste o código acima.

Uma outra maneira de acessar as variáveis de escopo global dentro de uma função é utilizando um array predefinido pelo PHP cujo nome é **\$GLOBALS[]**. Veja um exemplo de sua utilização:

```
function testaGlobal(){
    $GLOBALS['valor'] +=1;
    $GLOBALS['res'] +=1;
    echo 'Valor (dentro da função): '.$GLOBALS['valor'].', Resultado (dentro da função): '.$GLOBALS['res'].'<br/>';
}

$valor = 100;
$res = 10 * $valor;
echo 'Valor (antes da função): '.$valor.', Resultado (antes da função): '.$res.'  
<br/>';
testaGlobal();
echo 'Valor (depois da função): '.$valor.', Resultado (depois da função): '.$res.'  
<br/>';
```

A forma anterior certamente é mais prática!

Exercícios de fixação:

- 1) Teste o código acima.

27.If, else if, else

Veja um exemplo desta tradicional estrutura:

```
<?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";
}
?>
```

Pode haver vários elseifs dentro do mesmo if. A primeira expressão elseif (se houver) que retornar TRUE será executada.

No PHP, você pode escrever 'else if' (em duas palavras) que o comportamento será idêntico a 'elseif' (em uma única palavra).

O significado sintático é um pouco diferente (se você está familiarizado com C, o comportamento é o mesmo) mas no fundo é que ambos terão exatamente o mesmo comportamento.

O elseif só é executado se o if precedente ou qualquer elseif retornar FALSE, e o elseif atual retornar TRUE.

Exercícios de fixação:

- 1) Teste o código acima.

**Uma outra sintaxe usando os dois pontos (:) (Apenas a título de informação.
Abordagem muito específica e incomum)**

```
<?php

/* Incorrect Method: */
if($a > $b):
    echo $a." is greater than ".$b;
else if($a == $b): // Will not compile.
    echo "The above line causes a parse error.";
endif;

/* Correct Method: */
if($a > $b):
    echo $a." is greater than ".$b;
elseif($a == $b): // Note the combination of the words.
    echo $a." equals ".$b;
else:
    echo $a." is neither greater than or equal to ".$b;
endif;

?>
```

Note que elseif e else if só serão considerados exatamente iguais se usados com chaves como no 1º exemplo. Quando usando com dois pontos (:) para definir as condições if/elseif, como no exemplo acima, você não pode separar else if em duas palavras, ou o PHP irá falhar com um erro de interpretação.

Veja mais um exemplo alternativo:

```
<?php
if ($a == 5):
    echo "a equals 5";
    echo "...";
elseif ($a == 6):
    echo "a equals 6";
    echo "!!!";
else:
    echo "a is neither 5 nor 6";
endif;
?>
```

28.switch

Assim como em outras linguagens, o PHP também tem a estrutura switch que consiste em um comando de seleção com múltiplas opções.

Esta estrutura testa sucessivamente o valor de uma expressão, comparando o resultado dela com uma lista de constantes inteiras ou de caracteres.

Quando o valor coincide, os comandos associados àquela constante são executados. Para que os testes não continuem, utiliza-se o comando break para interromper a sequência de verificação. Veja o exemplo de um if/else substituído por um switch sem prejuízos:

```
<?php
if ($i == 0) {
    echo "i equals 0";
} elseif ($i == 1) {
    echo "i equals 1";
} elseif ($i == 2) {
    echo "i equals 2";
}

switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
}
?>
```

Um exemplo com strings

```
<?php
switch ($i) {
    case "apple":
        echo "i is apple";
        break;
    case "bar":
        echo "i is bar";
        break;
    case "cake":
        echo "i is cake";
        break;
}
?>
```

Observe atentamente o exemplo abaixo:

```
<?php
switch ($i) {
case 0:
case 1:
case 2:
    echo "i is less than 3 but not negative";
    break;
case 3:
    echo "i is 3";
}
?>
```

Este exemplo contempla as opções 0,1 e 2. Em qualquer dos casos o trecho de código contido no case 2 será executado!

default

Quando nenhuma das opções é válida, você pode ainda usar a declaração default, que corresponde a “nenhuma das alternativas anteriores”. Veja:

```
<?php
switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
    default:
        echo "i is not equal to 0, 1 or 2";
}
?>
```

29.for

O loop for é o loop mais complexo no PHP. Ele tem comportamento semelhante ao C. A sintaxe do loop for é:

```
for (expr1; expr2; expr3)
    statement
```

A primeira expressão (expr1) é executada incondicionalmente somente no começo do loop.

No começo de cada iteração a expr2 é avaliada. Se a avaliação resultar em TRUE, o loop continua e as instruções aninhadas são executadas. Se a avaliação resultar em FALSE, a execução do loop termina.

No final de cada iteração, a expr3 é executada.

Cada uma das expressões pode ser vazia ou conter múltiplas outras expressões separadas por vírgulas. Na expr2, todas as expressões separadas por vírgula são avaliadas, mas o resultado é obtido da última parte.

Se a expr2 estiver vazia significa que o loop deve ser executado indefinidamente (O PHP considera implicitamente como TRUE, igual ao C). Isto pode não ser tão inútil quanto você pensa, pois muitas vezes você deseja que o loop termine usando a instrução break em vez de usar a expressão verdade do for.

Analise os seguintes exemplos. Todos exibem números de 1 até 10:

```
/* exemplo 1 */

for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

/* exemplo 2 */

for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}

/* exemplo 3 */

$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}

/* exemplo 4 */

for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
```

É claro que o primeiro exemplo aparenta ser o melhor (ou talvez o quarto), mas você pode achar que usar expressões vazias no loop for seja vantajoso em algumas ocasiões.

Exercícios de fixação:

- 1) Teste os códigos acima
- 2) Nesse último for, faça com que ele imprima \$j.

30.while

O mesmo exemplo do for, que exibe números de 1 a 10, com o clássico while:

```
<?php
/* example 1 */

$i = 1;
while ($i <= 10) {
    echo $i++; /* the printed value would be
                  $i before the increment
                  (post-increment) */
}

/* example 2 */

$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
?>
```

Repare que no exemplo 1 estamos imprimindo e incrementando na mesma linha sem prejuízos à execução do código.

Exercícios de fixação:

- 1) Teste o código acima

31. Do-while

O loop do-while é muito similar ao loop while, exceto pela expressão de verificação que está no final de cada iteração em vez de estar no começo.

A maior diferença para o loop normal while é que a primeira iteração do loop do-while sempre é executada (a expressão de verificação somente é executada no final da iteração), considerando que no loop while não é necessariamente executada (a expressão de verificação é executada no começo de cada iteração e se o resultado for FALSE logo no começo, a execução do loop é abortada imediatamente).

Só há uma sintaxe para o loop do-while:

```
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```

O loop acima será executado somente uma vez, pois após a primeira iteração, quando a expressão de verificação for executada, ela resultará em FALSE (\$i não é maior que 0) e o loop será encerrado.

Exercícios de fixação:

- 1) Teste os códigos acima.

32. Interferindo no loop com break e continue

Break

O comando **break** tem dois propósitos: terminar um comando switch ou forçar a terminação imediata de um laço de repetição, ignorando o teste condicional normal do laço. Exemplo:

```
$i = 0;  
while (++$i) {  
    switch ($i) {  
        case 5:  
            echo "At 5<br />\n";  
            break 1; /* Exit only the switch. */  
        case 10:  
            echo "At 10; quitting<br />\n";  
            break 2; /* Exit the switch and the while. */  
        default:  
            break;  
    }  
}
```

Exercícios de fixação:

- 1) Teste o código acima

Continue

O comando **continue** trabalha de forma parecida com o break. A diferença é que em vez de interromper a iteração, o comando continue força o fluxo do programa a passar para a próxima iteração, pulando qualquer comando que exista entre ele e o final do bloco de código do laço. Lembrando que o continue é mais indicado para o laço for. Exemplo:

```
1 <?php
2     $i=0;
3     for($i=0;$i<=100;$i++){
4         if($i>=20 && $i<=90)
5             continue;
6         echo $i."<br>";
7     }
8 ?>
```

Exercícios de fixação:

- 1) Teste os códigos acima.

33. Funções

Uma função é um trecho de programa delimitado, que executa uma ação específica. Em geral, isola-se o trecho de código em uma função quando o mesmo tiver que ser utilizado mais de uma vez, ou ainda, por questões de portabilidade.

Retorno

Se você desejar que a função retorne algum valor, basta utilizar a palavra `return` associada ao valor que deseja retornar. Uma função pode retornar um simples valor, o conteúdo de uma variável ou até mesmo o resultado de uma expressão.

Uma função pode ter quantos comandos `return` forem necessários. No entanto, a função será interrompida ao encontrar o primeiro.

Veja um exemplo básico:

```
2 #declaração de função
3 function digaOi():void{
4     echo 'Oi<br>';
5 }
6
7 #declaração de função com retorno
8 function soma(int $a,int $b):int{
9     return ($a+$b);
10 }
11
12 #chamada das funções
13 digaOi();
14 $x = 20;
15 $y = 30;
16 $resultado = soma($x,$y);
17 echo "O total da soma de $x e $y é $resultado<br>";
```

IMPORTANTE: Conceitualmente, uma função que não retorna nada (`function digaOi` por exemplo) é uma sub-rotina e não verdadeiramente uma função. Para ser função precisa ter `return`.

Exercícios de fixação:

- 1) Teste o código acima.

34. Passagem de parâmetros por valor e por referência

Parâmetros

Os parâmetros são informações que a função precisa para realizar determinada tarefa. Esses parâmetros são passados pelo código que chama a função e dentro dela se comportam como variáveis locais.

Passagem de parâmetros por valor

Na passagem por valor é possível passar uma constante ou o valor (conteúdo) de uma variável. Neste segundo caso, o código da função vai utilizar essa importância sem que o conteúdo original da variável seja alterado. Nossa exemplo acima mostra esta passagem por valor dos conteúdos de \$x e \$y para os parâmetros (variáveis locais) \$a e \$b da função soma.

Passagem de parâmetros por referência

Passar um parâmetro por referência significa passar o endereço de memória daquela variável. Diferente do que acontece com a passagem por valor, onde ocorre uma cópia de conteúdo, na passagem por referência a variável pode sair da função com seu valor alterado.

Para fazer com que uma função receba parâmetros por referência você deve indicar na declaração da função que se espera receber um endereço e não um valor. Isso faz com que a passagem deste(s) parâmetro(s) seja sempre por referência. Nesse caso, utiliza-se o modificador “&”. Veja um exemplo:

```
2 #passagem por referência (declaração da função)
3 function incrementar(int &$num):void{
4     $num++;
5 }
6 #chamada da função
7 $x=10;
8 echo 'A variavel $x vale' . $x . '<br>';
9 incrementar($x); // $x passa a valer 11
10 echo 'A variavel $x agora vale' . $x . '<br>';
```

Exercícios de fixação:

- 1) Teste os códigos acima.

35. Parâmetros com valores predefinidos (default)

É possível predefinir valores para os argumentos das funções. No caso de a função ser chamada sem que o valor de um argumento seja especificado, estes valores predefinidos serão assumidos. Nestes casos, a passagem do parâmetro na chamada da função torna-se opcional. Veja exemplo:

```
12  function teste(string $valor="pré-definido"):void{
13  |     echo "$valor<br>";
14  }
15  teste(); //vai imprimir pré-definido
16  teste("outro valor"); //vai imprimir outro valor
```

Quando a função tem mais de um parâmetro, o que tem valor padrão deve ser declarado por último, como no exemplo abaixo:

```
18  function teste2(string $figura, string $cor = "azul"):void{
19  |     echo "A figura $figura tem a cor $cor<br>";
20  }
21  teste2("círculo");
22  teste2("círculo", "verde");
```

Exercícios de fixação:

- 1) Teste os códigos acima.

36. Funções com argumentos variáveis

No PHP é possível você passar e recuperar uma quantidade indefinida de argumentos para uma função. Para isso utilizamos uma função nativa chamada **func_get_args**. Veja o exemplo abaixo:

```
25  #declaração de função com argumentos variáveis
26  function minha_funcao():void{
27  |     $meusArgumentos = func_get_args();
28  |     var_dump($meusArgumentos);
29  |     echo "<br>";
30  }
31  #chamada de função com argumentos variáveis
32  minha_funcao("Olá", "mundo");
33  minha_funcao("Rafael", 5, 8.9, false);
34  minha_funcao("teste", 7.4, 9.7);
35  minha_funcao();
```

Saída no navegador:

```
array(2) { [0]=> string(4) "Olá" [1]=> string(5) "mundo" }
array(4) { [0]=> string(6) "Rafael" [1]=> int(5) [2]=> float(8.9) [3]=> bool(false) }
array(3) { [0]=> string(5) "teste" [1]=> float(7.4) [2]=> float(9.7) }
array(0) { }
```

Exercícios de fixação:

- 1) Teste o código acima.

Considerando as possibilidades e a existência da função nativa **func_num_args**, podemos escrever uma função soma tradicional e outra versão que possa receber uma quantidade de argumentos além do previsto. Veja:

```
38  function soma(float $a, float $b):float{
39  |     return ($a+$b);
40 }
41
42  function somav2(float $a, float $b):float{
43  |     $numArgs = func_num_args();
44  |     $args = func_get_args();
45  |     $resultado = 0.0;
46  |     for($i=0;$i<$numArgs;$i++)
47  |         $resultado += $args[$i];
48  |     return $resultado;
49 }
50
51 $x = 20;
52 $y = 50;
53
54 echo 'O resultado da soma de $x e $y é '.soma($x,$y)."<br>";
55 echo 'O resultado da soma de $x, $y e outros números é '.somav2($x,$y,7.8,90.6)."<br>";
```

Exercícios de fixação:

- 2) Teste o código acima.

37. Manipulação de Arrays

Definição de array

Arrays são arranjos de dados, também conhecidos como: cadeia, vetor ou sequência. O array, em PHP, é um mapa ordenado. Lembrando que um mapa é um tipo que retorna **valores para chaves**.

Acesso e Tipos de Dados

O valor de uma posição do array deve ser acessado por meio de seus índices, que podem ser valores de qualquer tipo e não somente inteiros.

Mesmo que os índices sejam todos do tipo inteiro, eles não precisam formar uma sequência contínua.

Quando um array é criado sem especificar os índices, eles são criados automaticamente a partir do zero.

Como a checagem dos tipos no PHP é dinâmica, valores de tipos diferentes podem ser usados como índices de array, assim como os valores atribuídos que também podem ser de diversos tipos.

Atribuição de valores para as chaves

Valores são atribuídos para suas respectivas chaves pelo operador =>. Exemplo:

```
//Declaração ( chave => valor )
$meuArray = array(
    'nome' => 'Juca',
    'idade' => 20,
    10 => 'Chocolate',
    11 => 500,
    'time' => 'Fluminense'
);

echo $meuArray[ 'nome' ].'<br />'; //Juca
echo $meuArray[ 'idade' ].'<br />'; //20
echo $meuArray[ 10 ].'<br />'; //Chocolate
echo $meuArray[ 11 ].'<br />'; //500
echo $meuArray[ 'time' ].'<br />'; //Fluminense
```

Chaves Automáticas

Se as chaves não forem especificadas na declaração, é feita uma numeração automática, iniciando em zero (0). Exemplo:

```
$novoArray = array( 100, 200, 300 );

$a = $novoArray[ 0 ]; //100
$b = $novoArray[ 1 ]; //200
$c = $novoArray[ 2 ]; //300
```

Se a numeração do primeiro item do array for especificada, os demais serão seus sucessores. Exemplo:

```
$trimestre = array(
    1 => 'Janeiro',
    'Fevereiro',
    'Marco'
);

echo $trimestre[ 1 ].'<br/>'; // 'Janeiro'
echo $trimestre[ 2 ].'<br/>'; // 'Fevereiro'
echo $trimestre[ 3 ].'<br/>'; // 'Marco'
```

Exercícios de fixação:

- 1) Teste os códigos acima

Atribuição

Ao usar um array com uma **chave** que ainda não havia sido definida, esta chave é criada automaticamente. Se o array ainda não tinha sido definido, ele também será criado automaticamente. Exemplo:

```
$exemplo[ 3 ] = 100;
$exemplo[ 4 ] = 'Texto';

$time[ 'Thiago' ] = 'Botafogo';
$time[ 'Rafael' ] = 'Fluminense';

echo $exemplo[ 3 ]."<br/>";
echo $exemplo[ 4 ]."<br/>";
echo "Time do Rafael: $time[Rafael]<br/>";
echo "Time do Thiago: $time[Thiago]<br/>";
```

Exercícios de fixação:

- 1) Teste o código acima.

Equivalência de declaração

A declaração

```
$ingredientes = array(
    'fruta' => 'macã',
    'legume' => 'batata'
);
```

equivale a:

```
$ingredientes[ 'fruta' ] = 'macã';
$ingredientes[ 'legume' ] = 'batata';
```

Checkando a existência ou nulidade da chave

Para checar a existência ou nulidade de chave, podemos usar a função `isset` vista anteriormente. Veja exemplos:

```
$a = array( 'nome' => 'Fulano', 'celular' => null );

var_dump( isset( $a['nome'] ) );           // bool(true)
var_dump( isset( $a['celular'] ) );         // bool(false)
var_dump( isset( $a['endereco'] ) );         // bool(false)
```

Exercícios de fixação:

- 1) Teste o código acima.

Removendo um elemento do array

Para remover um elemento do array, basta usar a função `unset` passando a chave do elemento desejado. Veja exemplo:

```
$meuArray = array( 'nome' => 'Eulano', 'celular' => '9999-0909', 'idade' => 30 );  
  
var_dump( $meuArray ); //Imprime todo o array  
  
unset( $meuArray[ 'nome' ] ); //remove o elemento de chave 'nome'  
  
var_dump( $meuArray );
```

Para remover o array todo, basta passar a variável sem a chave. Veja exemplo:

```
$meuArray = array( 'nome' => 'Eulano', 'celular' => '9999-0909', 'idade' => 30 );  
  
var_dump( $meuArray ); //Imprime todo o array  
  
unset( $meuArray ); //remove todo o array  
  
var_dump( $meuArray ); //Não vai imprimir nada
```

Exercícios de fixação:

- 1) Teste os códigos acima.

Imprimindo a estrutura de um array

Existem duas funções em PHP que facilitam a impressão de arrays, **print_r(\$array)** e **var_dump(\$array)**. Veja exemplos:

```
]<?php  
$cores = array( 'Azul', 'Verde', 'Amarelo', 'Branco' );  
echo "Saída com print_r: <br/><pre>";  
print_r( $cores );  
echo "</pre>";  
echo "<br/>Saída com var_dump (mais detalhado): <br/><pre>";  
var_dump( $cores );  
echo "</pre>";  
?>
```

Observe a saída produzida por cada um:

Saída com print_r:

```
Array
(
    [0] => Azul
    [1] => Verde
    [2] => Amarelo
    [3] => Branco
)
```

Saída com var_dump (mais detalhado):

```
array (size=4)
  0 => string 'Azul' (length=4)
  1 => string 'Verde' (Length=5)
  2 => string 'Amarelo' (length=7)
  3 => string 'Branco' (Length=6)
```

Exercícios de fixação:

- 1) Teste os códigos acima.

Imprimindo os elementos com foreach

Você pode usar o laço **foreach** para percorrer todo o array imprimindo apenas seus elementos. Veja exemplo:

```
|foreach($cores as $c) {
    echo "A cor é $c <br/>";
}
```

O resultado produzido será:

```
A cor é Azul
A cor é Verde
A cor é Amarelo
A cor é Branco
```

Imprimindo as chaves e os elementos do array

Você pode usar o mesmo laço foreach para imprimir de forma mais completa (chaves e elementos). Veja exemplo:

```
$cores = array( 'Azul', 'Verde', 'Amarelo', 'Branco' );

foreach($cores as $chave => $valor) {
    echo "$chave = $valor <br/>";
}
```

O resultado produzido será:

0 = Azul
1 = Verde
2 = Amarelo
3 = Branco

Imprimindo um elemento em uma string

Sabendo qual é a chave, você pode acessar seu valor diretamente. Veja:

```
$meuArray = array(  
    'fruta' => 'Maçã',  
    'legume' => 'Batata'  
,  
  
echo "Fruta favorita: ${meuArray[ 'fruta' ]}<br />";  
echo "Legume favorito: ${meuArray[ 'legume' ]}<br />";
```

Fruta favorita: Maçã
Legume favorito: Batata

Exercícios de fixação:

- 1) Teste todos os códigos acima, desde o 1º exemplo com o foreach.

Ordenando elementos

Para ordenar os elementos de um array, basta usar a função sort. Veja exemplo:

```
$cores = array( 'Azul', 'Verde', 'Amarelo', 'Branco' );  
  
sort( $cores );  
  
echo "<pre>";  
print_r( $cores );  
echo "</pre>";
```

Saída produzida:

```
Array  
(  
    [0] => Amarelo  
    [1] => Azul  
    [2] => Branco  
    [3] => Verde  
)
```

Contando e somando elementos

Para contar elementos podemos fazer uso das funções count e sizeof. Ambas produzem a mesma saída. Veja o exemplo:

```
$cores = array( 'Azul', 'Verde', 'Amarelo', 'Branco' );  
  
echo 'Total de cores: '.count( $cores )."<br/>"; //4  
echo 'Total de cores: '.sizeof( $cores )."<br/>"; //4
```

Para somar elementos usamos a função array_sum. Veja exemplo:

```
$numeros = array( 10, 20, 30 );  
echo array_sum( $numeros )."<br/>"; //60
```

Exercícios de fixação:

- 1) Teste todos os códigos acima com as funções sort, count, size e array_sum.

Verificando a existência de um elemento no array

Para verificar se um elemento existe em determinado array, utiliza-se a função in_array. Veja exemplo:

```
$cores = array( 'Azul', 'Verde', 'Amarelo', 'Branco' );  
  
echo in_array( 'Verde', $cores ); //1  
echo in_array( 'Grená', $cores ); //Não vai imprimir nada
```

Inserindo elementos no array

Para inserir novos elementos no array, você pode utilizar a função array_push. Veja exemplo:

```
$cores = array( 'Azul', 'Verde', 'Amarelo', 'Branco' );  
  
array_push( $cores, 'Vermelho' );  
array_push( $cores, 'Cinza', 'Preto' );  
  
echo "Saída após array_push: <br/><pre>";  
print_r( $cores );  
echo "</pre>";
```

O comando acima equivale a:

```
$cores = array( 'Azul', 'Verde', 'Amarelo', 'Branco' );  
  
$cores[] = 'Vermelho';  
$cores[] = 'Cinza';  
$cores[] = 'Preto';  
  
echo "Saída após as inclusões com []: <br/><pre>";
```

Veja o resultado obtido em ambos os casos:

Saída após array_push:

```
Array
(
    [0] => Azul
    [1] => Verde
    [2] => Amarelo
    [3] => Branco
    [4] => Vermelho
    [5] => Cinza
    [6] => Preto
)
```

Saída após as inclusões com []:

```
Array
(
    [0] => Azul
    [1] => Verde
    [2] => Amarelo
    [3] => Branco
    [4] => Vermelho
    [5] => Cinza
    [6] => Preto
)
```

Retornando os elementos indexados numericamente

Para retornar os elementos indexados numericamente, utiliza-se a função array_values. Veja exemplo:

```
$meuArray = array(
    'fruta' => 'Maçã',
    'legume' => 'Batata',
    'cerveja' => 'Antártica'
);

echo "<br /><pre>";
print_r( array_values( $meuArray ) );
echo "</pre><br/>";
```

O resultado produzido será:

```
Array
(
    [0] => Maçã
    [1] => Batata
    [2] => Antártica
)
```

Exercícios de fixação:

- 1) Teste todos os códigos acima com as funções in_array, array_push e array_values.

Passagem por valor e por referência utilizando array

Veja um código interessante que envolve passagem por valor e por referência utilizando array:

```
function exibir($infos) {
    foreach($infos as $chave => $valor)
        echo "$chave : $valor<br/>";
}

function exibirPorReferencia(&$infos) {
    if( gettype($infos['salario']) === 'double' )
        $infos['salario']+=$valor;
    echo "{$infos['nome']} : {$infos['salario']}<br/>";
}

function exibirPorReferenciaComFor(&$infos) {
    //Dentro do foreach você precisa ratificar a referência ao endereço de memória
    foreach($infos as $chave => &$valor){
        if(gettype($valor) === 'double')
            $valor+=400;
        echo "|$chave : $valor<br/>";
    }
}

$informacoes = array(
    'nome' => 'Rafael',
    'salario' => 2600.50
);
exibir($informacoes);
echo "Depois de executar a função exibir<br/>";
var_dump($informacoes);echo "<br/><br/>";

exibirPorReferencia($informacoes);
echo "Depois de executar a função exibirPorReferencia<br/>";
print_r($informacoes);echo "<br/><br/>";

exibirPorReferenciaComFor($informacoes);
echo "Depois de executar a função exibirPorReferenciaComFor<br/>";
print_r($informacoes);echo "<br/><br/>";
```

Exercícios de fixação:

- 1) Teste o código acima. Teve dúvidas? Procure o professor!

Aprendendo um pouco mais....

Mesmo não lidando com funções, é possível fazer alterações por referência em determinadas estruturas. Vejam o código abaixo.

```
$informacoes = array(
    'nome' => 'Rafael',
    'salario' => 2600.50
);

echo "Antes do foreach... <br/>";
print_r($informacoes);echo "<br/>";
foreach($informacoes as $chave => &$valor){
    if(gettype($valor) === 'double')
        $valor+=1000;
    echo "$chave : $valor<br/>";
}
echo "Depois do foreach... <br/>";
print_r($informacoes);
```

Exercícios de fixação:

- 1) Teste o código acima. Teve dúvidas? Procure o professor!

Arrays bidimensionais (Matriz)

Tanto no PHP quanto em outras linguagens existem os array multidimensionais. Para não complicar, trataremos apenas de arrays de duas dimensões, conhecidos como bidimensionais ou matrizes.

Imagine que queiramos guardar a seguinte estrutura:

	0	1	2
0	“Maçã”	“Banana”	“Laranja”
1	“Inhame”	“Batata”	“Cenoura”
2	“Couve”	“Mostarda”	“Rúcula”

Veja como seria a criação e o acesso a essa estrutura.

```
49 //Criando um array de produtos
50 $produtos = array();
51 //Criando um array de frutas e preenchendo de uma forma específica
52 $frutas = array();
53 $frutas[0] = "Maçã";
54 $frutas[] = "Banana";
55 //Criando um array completo
56 $legumes = array("Inhame","Batata");
57 //Colocando os arrays $frutas e $legumes nas posições 0 e 1 de $produtos
58 array_push($produtos, $frutas, $legumes);
59 array_push($produtos, array("Couve","Mostarda","Rúcula"));
60 //Adicionando um legume na linha 1 e coluna 2 de produtos
61 $produtos[1][2] = "Cenoura";
62 //Adicionando uma 3ª fruta ao array
63 $frutas[2] = "Laranja";
64 //Recolocando o array de frutas
65 $produtos[0] = $frutas;
66 print_r($produtos);
67 //Acessando uma posição específica do array
68 echo $produtos[2][1];
```

Exercícios de fixação:

- 1) Teste o código acima. Teve dúvidas? Procure o professor!

Se você rodar o script o resultado será:

```
Array ([0] => Array ([0] => Maçã [1] => Banana [2] => Laranja) [1] => Array ([0] => Inhame [1] => Batata [2] => Cenoura) [2] => Array ([0] => Couve [1] => Mostarda [2] => Rúcula)) Mostarda
```

Se você pressionar as teclas CTRL + U (boa prática) o resultado será exibido dessa forma:

```
1 Array
2 (
3     [0] => Array
4         (
5             [0] => Maçã
6             [1] => Banana
7             [2] => Laranja
8         )
9
10    [1] => Array
11        (
12            [0] => Inhame
13            [1] => Batata
14            [2] => Cenoura
15        )
16
17    [2] => Array
18        (
19            [0] => Couve
20            [1] => Mostarda
21            [2] => Rúcula
22        )
23
24 )
25 Mostarda
```

Exercícios de fixação:

- 1) Teste o código acima. Teve dúvidas? Procure o professor!

Utilizando strings com índices para as linhas.

Suponha que queiramos agora ter a seguinte estrutura:

	0	1	2
‘frutas’	“Maçã”	“Banana”	“Laranja”
‘legumes’	“Inhame”	“Batata”	“Cenoura”
‘verduras’	“Couve”	“Mostarda”	“Rúcula”

Veja como seria a criação e o acesso a essa estrutura.

```
73 $frutas = array();
74 $frutas[0] = "Maçã";
75 $frutas[] = "Banana";
76
77 $legumes = array("Inhame", "Batata", "cenoura");
78
79 //Nesse momento $produtos já é criado e definido como array
80 $produtos['frutas'] = $frutas;
81 //Para a posição/linha ‘legumes’, passamos o array $legumes
82 $produtos['legumes'] = array("Inhame", "Batata", "cenoura");
83
84 $verduras = array("Couve", "Mostarda", "Rúcula");
85 //Para a posição/linha ‘verduras’, passamos o array $verduras
86 $produtos['verduras'] = $verduras;
87 //Criando e fazendo atribuições para a posição/linha ‘laticinios’ do array $produtos
88 $produtos['laticinios'][0] = "Manteiga";
89 $produtos['laticinios'][1] = "Queijo";
90 array_push($produtos['laticinios'], "Presunto");
91 //Adiciona “Laranja” na posição 2 do array $frutas
92 $frutas[2] = "Laranja";
93 //Para a posição/linha ‘frutas’, passamos novamente o array $frutas
94 $produtos['frutas'] = $frutas;
95
96 print_r($produtos);
97 //Acessando posições específicas do array/matriz $produtos
98 echo $produtos['laticinios'][1]."<br/>";
99 echo $produtos['frutas'][2]."<br/>";
```

Veja como ficaria a impressão após um CRTL+U.

```
1 Array
2 (
3     [frutas] => Array
4         (
5             [0] => Maçã
6             [1] => Banana
7             [2] => Laranja
8         )
9
10    [legumes] => Array
11        (
12            [0] => Inhame
13            [1] => Batata
14            [2] => cenoura
15        )
16
17    [verduras] => Array
18        (
19            [0] => Couve
20            [1] => Mostarda
21            [2] => Rúcula
22        )
23
24    [laticinios] => Array
25        (
26            [0] => Manteiga
27            [1] => Queijo
28            [2] => Presunto
29        )
30
31 )
32 Queijo<br/>Laranja<br/>
```

Exercícios de fixação:

- 1) Teste o código acima. Teve dúvidas? Procure o professor!

Se quiser, pode imprimir todo o conteúdo da matriz \$produtos em forma de uma lista HTML. Veja:

```
102 echo "#####Lista de compras#####<hr>";
103 //Imprimindo a matriz $produtos na forma de lista com marcações HTML
104 foreach($produtos as $chave => $valor){
105     echo "<ul><strong>$chave</strong>" ;
106     //Ordenando cada array contido em $valor contido em cada chave ('frutas', 'legumes' etc)
107     sort($valor);
108     foreach($valor as $valor)// Não preciso das chaves que seriam 0,1 e 2
109     |   echo "<li>".$valor."</li>";
110     echo "</ul>";
111 }
```

O resultado será o seguinte:

```
#####
#Lista de compras#####

```

frutas
• Banana
• Laranja
• Maçã

legumes
• Batata
• Inhame
• cenoura

verduras
• Couve
• Mostarda
• Rúcula

laticinios
• Manteiga
• Presunto
• Queijo

Exercícios de fixação:

- 1) Teste o código acima. Teve dúvidas? Procure o professor!

Quando a matriz possui elementos numerados o for pode ser a melhor opção. Veja:

```
50 //Criando um array de produtos
51 $produtos = array();
52 //Criando um array de frutas e preenchendo de uma forma específica
53 $frutas = array();
54 $frutas[0] = "Maçã";
55 $frutas[] = "Banana";
56 //Criando um array completo
57 $legumes = array("Inhame","Batata");
58 //Colocando os arrays $frutas e $legumes nas posições 0 e 1 de $produtos
59 array_push($produtos, $frutas, $legumes);
60 array_push($produtos, array("Couve","Mostarda","Rúcula"));
61 //Adicionando um legume na linha 1 e coluna 2 de produtos
62 $produtos[1][2] = "Cenoura";
63 //Adicionando uma 3ª fruta ao array
64 $frutas[2] = "Laranja";
65 //Recolocando o array de frutas
66 $produtos[0] = $frutas;
67
68 echo "#####Lista de compras#####<hr>";
69 //Imprimindo a matriz $produtos na forma de lista com marcações HTML
70 for($linha = 0 ; $linha < count($produtos) ; $linha++){
71     echo "<strong> Linha ".$linha." da matriz</strong>";
72     for($coluna = 0; $coluna < sizeof($produtos[$linha]) ; $coluna++)
73         echo "<li> coluna $coluna (posição [$linha][$coluna]): ".$produtos[$linha][$coluna]."</li>";
74     echo "</ul>";
75 }
```

O resultado será exibido da seguinte forma:

```
#####Lista de compras#####
```

Linha 0 da matriz

- coluna 0 (posição [0][0]): Maçã
- coluna 1 (posição [0][1]): Banana
- coluna 2 (posição [0][2]): Laranja

Linha 1 da matriz

- coluna 0 (posição [1][0]): Inhame
- coluna 1 (posição [1][1]): Batata
- coluna 2 (posição [1][2]): Cenoura

Linha 2 da matriz

- coluna 0 (posição [2][0]): Couve
- coluna 1 (posição [2][1]): Mostarda
- coluna 2 (posição [2][2]): Rúcula

Exercícios de fixação:

- 1) Teste o código acima. Teve dúvidas? Procure o professor!

38. Manipulação de Strings

O PHP possui muitas funções para manipulação de strings, dentre as quais podemos destacar:

- **empty** → Retorna true se a string estiver vazia;
- **strlen** → Retorna o tamanho de uma string;
- **strpos** → Retorna a posição da primeira ocorrência de uma string;
- **substr** → Retorna parte de uma string;

- **explode** → Divide uma string em outras strings, usando um separador;
- **implode** → Junta elementos de um array em uma string, geralmente utilizando um ‘concatenador’;
- **str_split** → Converte uma string para um array;
- **str_replace** → Substitui todas as ocorrências de determinada string por uma outra string de substituição;
- **str_pad** → Preenche uma string de certo tamanho com outra string;
- **ltrim** → Retira espaços e tabs do início da string;
- **rtrim** → Retira espaços e tabs do final da string;
- **trim** → Retira espaços e tabs do início e do final da string;

Veremos a seguir, exemplos com cada uma destas funções:

empty, strlen, strpos e substr

```
//empty - Retorna true se a string estiver vazia;

echo empty( '' ); //1
var_dump( empty( 'Fulano' ) ); //boolean false

//strlen - Retorna o tamanho de uma string;

echo strlen( 'Rafael' ); //6

//strpos - Retorna a posição da primeira ocorrência de uma string;

$Texto = 'Olá Mundo';

var_dump( strpos( $Texto, 'X' ) ); // false
var_dump( strpos( $Texto, 'O' ) ); // 0
var_dump( strpos( $Texto, 'M' ) ); // 4
var_dump( strpos( $Texto, 'M', 5 ) ); // false
var_dump( strpos( $Texto, 'M', 4 ) ); // 4

//substr - Retorna parte de uma string;

$Texto = 'Olá Mundo';

echo substr( $Texto, 1 ).'<br/>'; // 'lá Mundo';
echo substr( $Texto, 0, 3 ).'<br/>'; // 'Olá';
echo substr( $Texto, 4, 5 ).'<br/>'; // 'Mundo';
```

Exercícios de Fixação:

- 1) Teste todos os códigos acima

explode

```
//explode - Divide uma string em outras strings, usando um separador;

echo "<pre>";
print_r( explode( '/', '06/11/1974' ) );
echo "</pre><br/>";

echo "<pre>";
print_r( explode( ':', '09:48:23' ) );
echo "</pre><br/>";
```

Resultado:

```
Array
(
    [0] => 06
    [1] => 11
    [2] => 1974
)
```

```
Array
(
    [0] => 09
    [1] => 48
    [2] => 23
)
```

Exercícios de Fixação:

- 2) Teste todos os códigos acima

implode

```
//implode - Junta elementos de um array em uma string, geralmente utilizando um 'concatenador';

echo "<pre>";
print_r( implode( '/', array( '06', '11', '1974' ) ) );
echo "</pre><br/>";

echo "<pre>";
print_r( implode( ':', array( '09', '48', '23' ) ) );
echo "</pre><br/>";
```

Resultado:

06/11/1974

09:48:23

Exercícios de Fixação:

- 3) Teste todos os códigos acima

str_split

```
//str_split - Converte uma string para um array;  
  
echo "<pre>";  
print_r( str_split( 'Fluminense' ) );  
echo "</pre><br/>";  
  
echo "<pre>";  
print_r( str_split( 'Fluminense', 2 ) );  
echo "</pre><br/>";
```

Resultado:

```
Array  
(  
    [0] => F  
    [1] => l  
    [2] => u  
    [3] => m  
    [4] => i  
    [5] => n  
    [6] => e  
    [7] => n  
    [8] => s  
    [9] => e  
)
```

```
Array  
(  
    [0] => Fl  
    [1] => um  
    [2] => in  
    [3] => en  
    [4] => se  
)
```

Exercícios de Fixação:

- 4) Teste todos os códigos acima

str_replace

```
//str_replace - Substitui todas as ocorrências de determinada string por uma outra string de substituição;  
  
$texto = 'Olá Mundo';  
  
echo str_replace( 'Mundo', 'Amigos', $texto ).'<br/>'; // 'Olá Amigos'  
  
$texto = 'Bombeiro';  
  
echo str_replace( 'o', 'a', $texto ).'<br/>'; // 'Bambeira'  
echo $qtdeTrocas.'<br/>'; // 2
```

Exercícios de Fixação:

- 5) Teste todos os códigos acima

str_pad

```
//str_pad - Preenche uma string de certo tamanho com outra string;

echo str_pad( 'Juca', 10 ).'<br/>'; // 'Juca      '
echo str_pad( 'Juca', 10, '-' ).'<br/>'; // 'Juca-----'
echo str_pad( 'Juca', 10, '-', STR_PAD_LEFT ).'<br/>'; // '-----Juca'
echo str_pad( 'Juca', 10, '-', STR_PAD_BOTH ).'<br/>'; // '--Juca--'
```

Exercícios de Fixação:

- 6) Teste todos os códigos acima

ltrim, rtrim e trim

```
//ltrim - Retira espaços e tabs do inicio da string;

echo ltrim( ' Juca' ).'<br/>'; // 'Juca'
echo ltrim( '**Juca', '*' ).'<br/>'; // 'Juca'

//rtrim - Retira espaços e tabs do final da string;

echo rtrim( 'Juca ' ).'<br/>'; // 'Juca'
echo rtrim( 'Juca*', '*' ).'<br/>'; // 'Juca'

//trim - Retira espaços e tabs do inicio e do final da string;

echo trim( ' Juca ' ).'<br/>'; // 'Juca'
echo trim( '**Juca**', '*' ).'<br/>'; // 'Juca'
```

Exercícios de Fixação:

- 7) Teste todos os códigos acima

Pesquise sobre outras funções para manipulação de strings como strtoupper, strtolower, ucwords e ucfirst.

39.Inclusão de arquivos

Tipos de inclusão de arquivos:

- **include** → Inclui a biblioteca/arquivo no local indicado. Se o arquivo/biblioteca não existir, a execução do script continua;
- **require** → Inclui a biblioteca/arquivo no local indicado. Se o arquivo/biblioteca não existir ou se contiver erros, a execução do script é interrompida;
- **include_once** → Idem à include, porém só inclui o arquivo/biblioteca uma única vez;
- **require_once** → Idem à require, porém só inclui o arquivo/biblioteca uma única vez;

Alguns exemplos de inclusão:

```
include( 'arquivo.php' );
include( 'pasta/arquivo.php' );
include_once( '../pasta/arquivo.php' );

require_once( 'arquivo.php' );
require( 'pasta/arquivo.php' );
```

Exercícios de Fixação:

- 1) Teste todos os códigos acima com arquivos reais.

Pesquise sobre os perigos do include de arquivos remotos.

A partir da versão 7 do PHP, erros fatais viraram exceções que podem ser tratadas. Falaremos disso mais adiante.

40. Funções – Um pouco mais

Neste tópico falaremos um pouco sobre os vários tipos de funções que existem no PHP. Quando falamos de funções atualmente é preciso deixar claro que o assunto não se esgota facilmente e deverá ser retomado nos tópicos posteriores.

Funções anônimas

No trecho de código apresentado abaixo, mostramos uma função comum e em seguida o mesmo exemplo aplicando uma função anônima.

```
2     $precoDeCusto = 450.0;
3     $margemDeLucro = 25.0;
4
5     //Função comum
6     function calcularPrecoDevenda($custo, $margem){
7         return $custo += ( ($custo * $margem) / 100);
8     }
9     var_dump( calcularPrecoDevenda($precoDeCusto, $margemDeLucro) );
10
11    //Usando uma função anônima (lambda)
12    $precoDeVenda = function($custo, $margem){
13        return $custo += ( ($custo * $margem) / 100);
14    };
15    //Vai imprimir a função --> Uma função anônima/lambda cujo tipo é um Closure
16    echo "<br>";var_dump($precoDeVenda);
17    //Vai imprimir o resultado da chamada da função
18    echo "<br>";var_dump($precoDeVenda($precoDeCusto, $margemDeLucro));
19
```

Em geral, funções anônimas podem ser atribuídas a uma variável sem necessidade de serem nomeadas e também podem ser utilizadas como callback, ou seja, passadas como argumento para uma outra função. Veremos a parte do callback mais adiante.

Quando usamos o var_dump() para verificar o tipo de uma função anônima, o tipo retornado sempre é um closure. No entanto, há dois tipos de função anônima. **O exemplo acima é o que conhecemos conceitualmente como lambda.** Trata-se de uma função normal, com argumentos, retorno etc. A diferença é que ela não tem um nome.

Como seria uma função anônima conhecida como closure e qual seria a diferença básica?

A diferença abaixo é que uma função anônima conceitualmente conhecida como closure pode utilizar variáveis externas ao seu escopo. Veja o exemplo observando a partir da linha 19.

```
2     $precoDeCusto = 450.0;
3     $margemDeLucro = 25.0;
4     //Função comum
5     function calcularPrecoDevenda($custo, $margem){
6         return $custo += ( ($custo * $margem) / 100);
7     }
8     var_dump( calcularPrecoDevenda($precoDeCusto, $margemDeLucro) );
9
10    //Usando uma função anônima (lambda)
11    $precoDeVenda = function($custo, $margem){
12        return $custo += ( ($custo * $margem) / 100);
13    };
14    //Vai imprimir a função --> Uma função anônima/lambda cujo tipo é um Clousure
15    echo "<br>";var_dump($precoDeVenda);
16    //Vai imprimir o resultado da chamada da função
17    echo "<br>";var_dump($precoDeVenda($precoDeCusto, $margemDeLucro));
18
19    //Usando uma função anônima (closure)
20    $precoDeVendaClosure = function($custo) use($margemDeLucro){
21        return $custo += ( ($custo * $margemDeLucro) / 100);
22    };
23    //Vai imprimir a função --> Uma função anônima/closure cujo tipo é um Clousure
24    echo "<br>";var_dump($precoDeVendaClosure);
25    //Vai imprimir o resultado da chamada da função recebendo apenas uma variável como argumento
26    echo "<br>";var_dump($precoDeVendaClosure($precoDeCusto));
27    //Perceba que só passamos um argumento. A variável externa $margemDeLucro foi utilizada
28    echo "<br>";var_dump($precoDeVendaClosure(500.0)); //Passando um literal
29    //Perceba que só passamos um valor literal como argumento. A variável externa $margemDeLucro foi utilizada
```

Arrow Functions

As chamadas arrow functions visam proporcionar um código mais enxuto ao desenvolvedor. No entanto, pode ser que o mesmo sinta dificuldades em relação à legibilidade. Veja o mesmo exemplo das funções anteriores na forma de arrow function. Lembrando que arrow functions também são funções anônimas e que, portanto, também podem ser utilizadas como callback (argumento para outras funções).

```
31    //Usando Arrow function
32
33    $precoDeVenda2 = fn($custo, $margem) => $custo += ( ($custo * $margem) / 100);
34    //Vai imprimir a arrow function que também é um Clousure
35    var_dump($precoDeVenda2);
36    //Vai imprimir o resultado da chamada da função
37    echo "<br>";
38    var_dump($precoDeVenda2($precoDeCusto, $margemDeLucro));
```

O que mudou?

- Redução da palavra **function** por **fn**;
- O par de chaves {} foi substituído por uma arrow =>;
- O return foi suprimido.

Funções tipadas

A possibilidade de tipar argumentos e o retorno de funções surgiu a partir da versão 7 do PHP. Isso veio para aproximar ainda mais a linguagem do paradigma orientado a objetos e a possibilidade de escrever códigos mais rígidos, com regras mais bem definidas. Veja o código abaixo e observe atentamente os comentários.

```
45 //Funções tipadas
46 function somar(int $n1, int $n2):int{
47     return ($n1+$n2);
48 }
49 $numero1 = 25;
50 $numero2 = 20;
51
52 echo "<br>";var_dump(somar($numero1, $numero2)); //45
53 echo "<br>";var_dump(somar($numero1, 25.75)); //Vai imprimir 50 e não 50.75!!!!
54 //Quando passamos um float esperando um int, perde-se a parte não inteira (0.75)
55 //Passando um valor incompatível
56 echo "<br>";var_dump(somar($numero1, "Rafael")); //Vai dar erro fatal.
57 //A função não espera uma string
58
59 $somar = function(int $n1, int $n2):int{
60     return ($n1+$n2);
61 };
62 echo "<br>";var_dump($somar($numero1, $numero2));
```

Funções utilizando o spread operator

Assim como a já citada func_get_args(), o operador spred trás a possibilidade de deixar em aberto a quantidade de argumentos recebidos por uma função. Veja o exemplo de código abaixo com atenção aos comentários.

```
64 //Spread operator
65 function somar2(int ...$numeros){
66     //return array_sum($numeros); //SOLUÇÃO SIMPLES!!
67
68     //Solução menos inteligente
69     $total = 0.0;
70     foreach($numeros as $n)
71         $total += $n;
72     return $total;
73 }
74
75 echo "<br>";var_dump(somar2($numero1, $numero2, 38, 47, 26));
```

IMPORTANTE: O spread operator também pode ser utilizado em funções anônimas!

Usando o tipo de retorno para fazer conversão de tipos

Veja um exemplo:

```
64 //Spread operator
65 function somar2(int ...$numeros):string{
66     //return array_sum($numeros); //SOLUÇÃO SIMPLES!!
67
68     //Solução menos inteligente
69     $total = 0.0;
70     foreach($numeros as $n)
71         $total += $n;
72     return $total;
73 }
74
75 echo "<br>";var_dump(somar2($numero1, $numero2, 38, 47, 26));
--
```

Observe que agora em vez de retornar 156, a função vai retornar o tipo convertido para string que é “156”.

O mesmo pode ser feito em relação a outros tipos compatíveis.

De float p/ int com perda de valor.

De int para float sem grandes problemas.

De boolean para int, enfim... teste as possibilidades!

Exercícios de fixação:

- 1) Teste todos os exemplos acima. São muitos, mas vocês precisam praticar.

41.Funções anônimas e callback

Nos capítulos anteriores falamos bastante sobre as funções anônimas e exemplos de uso. Também foi dito que as mesmas servem para serem passadas como callback (argumentos) para outras funções. Como exemplo, deixo o trecho de código abaixo e peço que atentem para os comentários.

```
3 //Criando uma função anônima (lambda)
4 $media = function(float ...$notas):float{
5     return array_sum($notas)/sizeof($notas);
6 };
7 //declarando uma variável e atribuindo valor a ela
8 $n4= 7.7;
9 //Invocando a função anônima
10 echo "O valor da média é ".$media(5.9,6.0,7.1,$n4)."<br>";
11
12 //Declarando uma função que recebe outra função como 1º argumento e valores do tipo float como argumentos seguintes
13 $resultado = function(callable $fnMedia, float ...$notas):string{
14     //Os 3 pontos (...) são para que o tipo seja mantido e não entendido como um único valor do tipo array
15     $valorMedia = $fnMedia(... $notas);
16     //Utilizando um operador ternário para fazer a verificação
17     return ($valorMedia>=6.0)? "Aprovado": "Reprovado";
18 };
19
20 echo "O resultado é ".$resultado($media, 5.9,6.0,7.1,$n4)."<br>";
```

Se tiver dúvidas para entender o código, peça auxílio ao professor!

Exercícios de fixação:

- 1) Teste o exemplo acima.

42. Requisição e Resposta (Revisão)

O protocolo HTTP

Nas aplicações web a comunicação entre cliente (browser) e servidor acontece via protocolo HTTP.

O protocolo HTTP (Protocolo de Transferência de Hiper Texto) é executado na camada de Aplicação (Modelo OSI) e usa, por padrão, a porta 80.

Esta comunicação segue o modelo cliente-servidor, que implementa o paradigma "requisição e resposta". Veja a figura:



O protocolo HTTP dispõe de diversos **comandos** (métodos). São eles:

- **GET**: Solicita um recurso (ex: arquivo).
- **HEAD**: Solicita informações sobre um recurso.
- **POST**: Envia dados para serem processados para o recurso desejado.
- **PUT**: Envia um recurso.
- **DELETE**: Apaga um recurso.
- **TRACE**: Mostra a rota da requisição entre servidores e eventuais mudanças realizadas.
- **OPTIONS**: Recupera os métodos HTTP que o servidor aceita.
- **CONNECT**: Serve usualmente para estabelecer uma conexão segura (ex: SSL).

Os comandos GET e POST são os mais utilizados.

Como acontece a navegação?

- Quando digitamos uma URL no navegador (*browser*), ele irá realizar um GET.
- O arquivo baixado será colocado numa pasta (de "arquivos temporários de internet") e depois será processado.
- Se o navegador souber como exibi-lo, ou tiver um *plugin* que saiba, irá fazê-lo.

Apache e PHP

O processo do Apache, httpd, monitora as requisições na porta 80 (ou outra configurada) e as processa.

Ao processar uma página PHP, o Apache a envia ao seu módulo PHP, que a processa e devolve o resultado ao Apache, que por sua vez, envia ao requisitante.

Processamento de HTML no cliente

1. O cliente irá baixar o arquivo requisitado e renderizá-lo, através de seu navegador.
2. Neste processo de renderização, será lido o código em linguagem HTML.
3. O código será transformado num modelo orientado a objetos chamado *Document Object Model* (DOM) e os objetos visíveis serão desenhados.
4. O navegador também processará código JavaScript, que consegue acessar e modificar o DOM, adicionando dinamismo e interatividade à página.

Observações

1. Repare que em nenhum momento o cliente processa código PHP (somente o servidor).
2. Como veremos posteriormente, usaremos PHP para gerar código HTML a ser enviado para o cliente.

43. Envio de Formulário por HTML

Mesmo trabalhando com PHP no lado servidor, podemos submeter um formulário através de uma simples página HTML. Veja um exemplo clássico:

```
<html>
<head>
    <title>Exemplo de envio de dados</title>
</head>
<body>
    <form id='f' name='f' method='post' action='destino.php'>
        <label for='nome'>Nome:</label>
        <input id='nome' name='nome' type='text' />
        <label for='email'>Email:</label>
        <input id='email' name='email' type='text' />
        <input id='enviar' name='enviar' type='submit'
            value='Enviar' />
    </form>
</body>
</html>
```

Como obter os dados com PHP?

Para obter os dados enviados via método POST, há a variável global **\$_POST**. Ela é um *array* que contém todos os nomes (propriedade **name**) dos campos enviados.

Similarmente, há a variável **\$_GET**, para obter dados enviados pelo método GET. Veremos mais adiante.

Considerações importantes

1. Os campos do formulário devem ter o atributo **id**, para respeitar as regras do XHTML (W3C). O id é importante quando utilizamos JavaScript. Aliás, o padrão é usar JavaScript. Por enquanto o foco continua no HTML e no PHP.
2. Também devem ter o atributo **name**, para poder ter seus valores obtidos pelo PHP. O PHP só reconhece o name.

3. Portanto, acostume-se a *sempre* usar **id** e **name** nos campos de formulário.

Veja como ficaria o arquivo destino.php, que vai receber e processar as informações enviadas pelo cliente:

```
<?php
// Arquivo destino.php
echo 'Nome: ', $_POST[ 'nome' ], '<br />';
echo 'Email: ', $_POST[ 'email' ], '<br />';
?>
```

Exercícios de fixação:

- 1) Teste o código acima.

Verificação dos campos

- Campos não enviados não estarão no *array*.
- Se tentarmos acessar uma chave inexistente, será gerado um erro.
- Logo, é importante verificar se a chave existe.
 - Podemos usar a função **empty** para isso.
- A função **isset** não é boa opção porque entende “” como variável existente.

Veja como ficaria a versão correta do arquivo destino.php:

```
if(! empty($_POST['nome']))
    echo 'Nome: ', $_POST['nome'], '<br/>';
if(! empty($_POST['email']))
    echo 'E-mail: ', $_POST['email'], '<br/>';
```

Exercícios de fixação:

- 1) Teste a mudança acima.

Sugestão

- Sempre verifique se **todos** os campos esperados foram enviados pelo cliente em sua requisição **POST**.
 - Caso não tenham sido enviados, pode-se terminar a execução do script informando que o campo não foi enviado.
- No caso do **GET**, verifique os parâmetros obrigatórios.

Veja como ficaria o código aprimorado do arquivo destino.php:

```
$campos= array('email', 'nome');
foreach($campos as $c){
    if(empty($_POST[$c])){
        die("Campo $c não enviado");
    }
}
```

Exercícios de fixação:

- 1) Teste a mudança acima.

Uso do método GET

- O método GET **não** deve ser usado quando for preciso enviar senhas ou quaisquer outras informações importantes.
- Os dados enviados de um formulário através do método GET (method='get') serão **enviados junto à requisição da página** e, portanto, serão visíveis a todos.
- A URL da página será exibida na barra de endereços do navegador junto aos dados. Ex: <http://localhost/destino.php?nome=rafael&email=rafael@cefet.com>

44. Envio de dados na requisição

Uso de parâmetros na URL

Podemos enviar dados junto à requisição por uma página. Basta seguir as seguintes regras:

- O primeiro dado deve vir após um símbolo de interrogação (?) e seu valor após o símbolo de igualdade (=).
- Cada dado adicional deve vir precedido de um e-comercial (&).

Exemplos:

<http://dvds.com?id=12345>
<http://livros.com?isbn=12345678901>
<http://livros.com?autor=102&ano=1995>
<http://revistas.com?nome=Veja&ano=1990&mes=1>

Obtendo os dados enviados via GET

Em PHP, podemos obter os dados que são enviados via GET pela variável `$_GET` (de modo similar ao visto para POST).

Exemplo: <http://revistas.com?nome=Veja&ano=1990&mes=1>

O index.php poderia obter os dados como:

```
<?php  
  
if ( isset( $_GET[ 'nome' ] ) ) echo $_GET[ 'nome' ];  
if ( isset( $_GET[ 'ano' ] ) ) echo $_GET[ 'ano' ];  
if ( isset( $_GET[ 'mes' ] ) ) echo $_GET[ 'mes' ];  
  
?>
```

Exercícios de fixação:

- 1) Crie uma página ex1.php que imprima o valor dos dados tipo e código recebidos via GET, como nos exemplos abaixo:

- <http://localhost/ex1.php?tipo=livro&codigo=123>
 - <http://localhost/ex1.php?tipo=dvd&codigo=19>
- 2) Crie uma página HTML contato.html que contenha um formulário com os campos nome e telefone e que envie estes dados, via POST, para o arquivo ex2.php, que deve imprimir o valor dos dados recebidos.

45. Redirecionamento

Podemos enviar o usuário para outra página quando preciso. Isto pode ser feito com o uso da função **header**, que serve para muitos outros usos.

Exemplo:

```
header('Location: www.site.com');
```

Outro uso comum é o redirecionamento para outra seção do próprio site, como quando há um parâmetro na URL que indique isto. Ex:

Suponha que a página aceite um parâmetro "p" que redirecione para a página com o nome informado. Usar "index.php?p=eletronicos" fará redirecionar para a página de produtos eletrônicos. Veja como ficaria o arquivo index.php:

```
<?php
// index.php
if ( isset( $_GET[ 'p' ] ) ) {
    $pagina = $_GET[ 'p' ] . '.php';
    header( "Location: $pagina" );
}
?>
```

Considerações sobre segurança

Se for redirecionar para uma página que contiver algum valor fornecido pelo usuário, como no exemplo anterior, **sempre** verifique esse valor.

Há grande possibilidade de um usuário mal-intencionado tentar exibir alguma página a que ele não possui acesso, mas é acessível pelo servidor através do método **header**

Veja um exemplo de código:

```
<?php
// index.php
if ( isset( $_GET[ 'p' ] ) ) {
    $pagina = $_GET[ 'p' ];
    $possuiPonto = !( strpos( $pagina, '.' ) === false );
    $possuiBarra = !( strpos( $pagina, '/' ) === false );
    if ( $possuiPonto || $possuiBarra )
        die( 'Acesso negado.' );
    $pagina .= '.php';
    if ( ! file_exists( $pagina ) )
        die( 'Página inexistente.' );
    header( "Location: $pagina" );
}
?>
```

Exercícios de fixação:

- 1) Teste o código acima.
- 2) Crie uma página pesquisa.html com um formulário contendo um campo busca e um botão OK. Ao clicar em OK, o formulário deve enviar a busca para o arquivo ex3.php. Este último deve redirecionar o usuário para o site <http://www.google.com.br/#hl=pt-BR&q=>
 - a. onde, no parâmetro q, deve ser usado o valor obtido do campo busca.

A página que processa a informação deve ficar mais ou menos assim:

```
<?php
if ( isset( $_GET[ 'p' ] ) ) {
    $pagina = $_GET[ 'p' ];

    $pagina = 'http://www.google.com.br/#hl=pt-BR&q=' . $pagina;
    header( "Location: $pagina" );
}

?>
```

46.Upload de Arquivos

Como funciona

1. No formulário, declaramos as entradas (*inputs*) com o tipo **file**.
2. Quando o formulário é enviado, os arquivos vão para a pasta **tmp** do PHP, como arquivos temporários, e, portanto, recebem um nome temporário.
3. Se os arquivos não forem tratados na requisição em que são enviados, eles são apagados do servidor.
4. Logo, é preciso tratá-los:
 1. Obter o conteúdo de cada arquivo temporário;
 2. Salvar cada conteúdo em outro local, com o nome correto;

No formulário HTML

Veja como ficaria o arquivo upload.html:

```
<html> <head> <title>Exemplo de Upload de Arquivos</title> </head>
<body>
<h1> Exemplo de Upload de Diversos Arquivos </h1>
<!-- Tipo de codificação, "enctype", deve ser indicado ! -->
<form enctype='multipart/form-data' id='f' action='destino.php' method='post' >
<!-- Tipo de formato MIME aceito, "accept", deve ser indicado caso se queira aceitar somente alguns formatos -->

<label for='arquivo1' >Documento:</label>
<input type='file' id='arquivo1' name='arquivo1'
accept='application/msword,application/pdf' /><br />

<label for='arquivo2' >Imagem:</label>
<input type='file' id='arquivo2' name='arquivo2'
accept='image/*' /> <br />

<input id='enviar' name='enviar' type='submit' value='Enviar' />
</form>
</body>
</html>
```

Alguns MIMEs

Exemplo	Formato
image/*	Qualquer imagem
video/*	Qualquer vídeo
audio/*	Qualquer áudio
application/zip	Arquivo .ZIP
application/pdf	Arquivo .PDF
application/msword	Arquivo .DOC, .DOT
application/powerpoint	Arquivo .PPT
application/excel	Arquivo .XLS

Lista completa de MIMEs:

- Veja a lista não oficial em: <http://reference.sitepoint.com/html/mime-types-full>
→ Veja a lista oficial em: <http://www.iana.org/assignments/media-types>

Variável \$_FILES

1. \$_FILES é a matriz com os dados dos arquivos enviados.
2. Funciona de forma semelhante à \$_POST.
 - a. Ex.: \$_POST['arquivo1']
3. Porém, possui informações de cada arquivo enviado:
 - ['name'] : Nome
 - ['type'] : Tipo
 - ['size'] : Tamanho
 - ['tmp_name'] : Nome do arquivo temporário
 - ['error'] : Erro ocorrido

Ex.: echo \$_FILES['arquivo1']['name'];

Listando os arquivos recebidos

Veja como ficaria o arquivo destino.php:

```
<?php
$campos = array( 'arquivo1', 'arquivo2' );
foreach ( $campos as $c ) {
    echo $c,
    '<br />Nome: ',           $_FILES[ "$c" ][ 'name' ],
    '<br />Tipo: ',           $_FILES[ "$c" ][ 'type' ],
    '<br />Tamanho (bytes): ', $_FILES[ "$c" ][ 'size' ],
    '<br />Nome Temp.: ',     $_FILES[ "$c" ][ 'tmp_name' ],
    '<br />Erro: ',           $_FILES[ "$c" ][ 'error' ],
    '<br /><br />';
}
// Arquivos são apagados ao final !
?>
```

Exercícios de fixação:

- 1) Escreva os códigos dos arquivos upload.html, destino.php e faça os testes.

Problema: Limite

O arquivo com as configurações do PHP (php.ini) determina, por default, algumas limitações. São elas:

- Limite de tamanho de arquivos é 2M
- Limite de arquivos enviados por vez é 20
- Limite de tamanho do POST é 8M

Para fazer as modificações necessárias para aumentar estes limites, procure as referidas linhas e faça as alterações:

```
post_max_size = 8M
...
upload_max_filesize = 2M
max_file_uploads = 20
```

Soluções

1. Ajustar o **php.ini** manualmente.
 - Deve ter permissão de acesso.
2. Ajustando a configuração durante a execução:

```
$max = 1024 * 1024 * 100; // 100 MB
ini_set( 'post_max_size', $max );
ini_set( 'upload_max_filesize', $max );
ini_set( 'max_file_uploads', 50 );
    ○ Deve ter permissão, senão as configurações não funcionam.
```

- Com **ini_set**, as configurações originais serão restauradas após a execução do script.

3. Ajustar o **.htaccess** (recomendado)

```
RewriteEngine On  
# Para PHP 5, Apache 1 e 2  
<IfModule mod_php5.c>  
    php_value post_max_size 100M  
    php_value upload_max_filesize 100M  
    php_value max_file_uploads 50  
</IfModule>
```

Obs.: Crie um arquivo com nome **.htaccess** na pasta raiz do site.

Recebendo diversos arquivos (lado cliente)

Para facilitar o recebimento de vários arquivos no PHP, defina o nome (*name*) da entrada (*input*) como um array PHP:

```
<label for='arquivo1'>Arquivo 1:</label>  
<input type='file' id='arquivo1' name='arquivos[]' /> <br />  
  
<label for='arquivo2'>Arquivo 2:</label>  
<input type='file' id='arquivo2' name='arquivos[]' /> <br />
```

Veja como ficaria o arquivo upload2.html

```
<html> <head> <title>Exemplo de Upload de Diversos Arquivos</title> </head>  
<body>  
    <h1> Exemplo de Upload de Diversos Arquivos </h1>  
    <!-- Tipo de codificação, "enctype", deve ser indicado -->  
    <!-- Método de envio deve ser post -->  
    <form enctype='multipart/form-data' id='f' action='destino2.php' method='post'>  
  
        <label for='arquivo1'>Arquivo 1:</label>  
        <input type='file' id='arquivo1' name='arquivos[]' /> <br />  
  
        <label for='arquivo2'>Arquivo 2:</label>  
        <input type='file' id='arquivo2' name='arquivos[]' /> <br />  
  
        <input id='enviar' name='enviar' type='submit' value='Enviar' />  
    </form>  
</body>  
</html>
```

Recebendo diversos arquivos (lado servidor)

Veja como ficaria o arquivo destino2.php

```
<?php
$nomes = $_FILES[ 'arquivos' ][ 'name' ];
$tipos = $_FILES[ 'arquivos' ][ 'type' ];
$stamanhos = $_FILES[ 'arquivos' ][ 'size' ];
$nomesTemporarios = $_FILES[ 'arquivos' ][ 'tmp_name' ];
$erros = $_FILES[ 'arquivos' ][ 'error' ];

$max = count( $nomes );
for ( $i = 0; $i < $max; ++$i ) {
    echo '<br />Nome: ', $nomes[ $i ],
          '<br />Tipo: ', $tipos[ $i ],
          '<br />Tamanho (bytes): ', $stamanhos[ $i ],
          '<br />Nome Temp.: ', $nomesTemporarios[ $i ],
          '<br />Erro: ', $erros[ $i ],
          '<br /><br />';
}
// Arquivos são apagados ao final !
?>
```

Exercícios de Fixação:

- 1) Escreva os códigos de upload2.html e destino2.php. Faça os testes!

Movendo um arquivo temporário

move1.php:

```
<?php
$diretorioUpload = 'up/'; // Diretório "up" dentro da pasta atual
$arquivoDestino = $diretorioUpload . $_FILES[ 'arquivo1' ][ 'name' ];
$arquivoOrigem = $_FILES[ 'arquivo1' ][ 'tmp_name' ];
if ( move_uploaded_file( $arquivoOrigem, $arquivoDestino ) ) {
    echo 'Sucesso!';
} else {
    echo 'Ocorreu um erro.';
}
?>
```

Movendo vários arquivos temporários

moveVarios.php

```
<?php
$nomes = $_FILES[ 'arquivos' ][ 'name' ];
$nomesTemporarios = $_FILES[ 'arquivos' ][ 'tmp_name' ];

$diretorioUpload = 'up/'; // Diretório "up" dentro da pasta atual
$max = count( $nomes );
for ( $i = 0; $i < $max; ++$i ) {
    $origem = $nomesTemporarios[ $i ];
    $destino = $diretorioUpload . $nomes[ $i ];
    echo '<br />Movendo: "' . $origem . '" para "' . $destino . '"... ';
    if ( move_uploaded_file( $origem, $destino ) ) {
        echo 'OK';
    } else {
        echo 'ERRO';
    }
}
?>
```

Exercícios de fixação.

- 1) Escreva e teste os dois códigos acima.

47. Outras funções para arrays

Nos capítulos anteriores já mencionamos e mostramos exemplos de diversas funções para manipulação de arrays, dentre as quais podemos destacar sort, sizeof, count, in_array, array_sum, array_push entre outras. No entanto, à medida que vamos adquirindo novos conhecimentos sobre a linguagem, a abordagem de outras funções começa a fazer mais sentido. Por isso falaremos um pouco mais sobre algumas funções para manipulação de arrays neste capítulo.

array_map

Imagine que precisamos, em algum momento, realizar algum tipo de operação a cada elemento de um array e que essa operação possa retornar um novo array modificado por essa operação. Aposto que já pensaram em usar um for para resolver a questão. No caso do array_map isso não se faz necessário visto que ele pode receber como argumento uma função anônima (ou até mesmo nomeada) e um array onde será aplicada essa função para cada elemento. Veja um exemplo bem simples onde geramos um novo array com todos os elementos elevados ao cubo.

```
22 //array_map com função nomeada
23 function cubo(int $n):int{
24     return ($n * $n * $n);
25 }
26
27 $array_inteiros = array(1,2,3,4,5);
28 echo "<br>"; print_r($array_inteiros);
29 $array_inteiros_ao_cubo = array_map('cubo', $array_inteiros);
30 echo "<br>"; print_r($array_inteiros_ao_cubo);
31
32 //array_map com função anônima
33 $array_inteiros_ao_cubo = array_map(function(int $n):int{
34     return ($n * $n * $n);
35 }, $array_inteiros);
36 echo "<br>"; print_r($array_inteiros_ao_cubo);
```

Perceba que apresentamos duas maneiras de se utilizar array_map. Uma com função nomeada e outra com função anônima.

Outro exemplo. Dessa vez, além da função que monta e retorna um array de mensagens traduzidas, passamos dois arrays como argumento.

```
38 //array_map com função anônima guardada em uma variável e dois arrays como argumento
39 $array_numbers = array("One", "Two", "Three", "Four", "Five");
40 $mostrarEmIngles = function(int $numero, string $numeroEmIngles):string{
41     return "O número $numero em inglês é $numeroEmIngles";
42 };
43
44 $arrayTextosTraduzidos = array_map($mostrarEmIngles, $array_inteiros, $array_numbers);
45
46 echo "<br>"; print_r($arrayTextosTraduzidos);
47
48 //array_map com função anônima e dois arrays como argumento
49 $arrayTextosTraduzidos = array_map(function(int $numero, string $numeroEmIngles):string{
50     return "O número $numero em inglês é $numeroEmIngles";
51 }, $array_inteiros, $array_numbers);
52
53 echo "<br>"; print_r($arrayTextosTraduzidos);
```

Dessa vez utilizamos uma função anônima guardada em uma variável e depois utilizamos uma função anônima passada diretamente como argumento.

Neste exemplo abaixo, utilizamos dois arrays para gerar um array com chave e valor recebidos como argumento. Um dos arrays vai fornecer as chaves e o outro array vai fornecer os valores. A função anônima cuida de montar esse novo array.

```
55 //array_map com função anônima gerando um novo array com chave e valor recebidos como argumento
56 $arrayNumerosEmIngles = array_map(function(int $numero, string $numeroEmIngles):array{
57     return array($numero => $numeroEmIngles);
58 }, $array_inteiros, $array_numbers);
59
60 echo "<br>"; print_r($arrayNumerosEmIngles);
```

Perceba que a função array_map pode ser utilizada em inúmeras situações. Basta surgir a necessidade ou utilizar a criatividade!

array_filter

Serve para eliminar elementos de um array com base em alguma expressão boleana.
Veja um exemplo.

```
59     #array_filter
60     $numeros = [1,2,3,4,5];
61     $numerosPares = array_filter($numeros, function(int $n):int{return ($n%2)==0;});
62     $numerosPares = array_filter($numeros, fn(int $n):int=>($n%2)==0);
63     #imprimindo a estrutura
64     echo "<pre>";
65     print_r($numerosPares);
66     echo "</pre>";
```

Exercícios de fixação:

- 1) Teste todos os exemplos acima.
- 2) Faça todos os exercícios fornecidos pelo professor.
- 3) Pesquise sobre outras funções para manipulação de arrays. No link abaixo você pode encontrar muitas delas:

<https://www.joemaster.com.br/manual/php/ref.array.html>

48. Um pouco sobre funções de data e hora

Para trabalhar com data e hora no PHP temos várias funções e recursos disponíveis.

Principais funções e suas características:

- **date()** → Retorna data e hora atual no formato especificado. Essa função não possui o setlocale que seria a opção de traduzir para um determinado idioma. A função date() utiliza o idioma do servidor.
- **time()** → Retorna o timestamp que é a quantidade de segundos decorridos desde 01/01/1970. O timestamp representa uma data e hora em segundos e por isso serve para representar uma data sendo utilizada como argumento da função date().
- **strtotime()** → Retorna o timestamp referente a uma data passada como argumento num determinado formato. Além de uma data há outras opções que podem servir como argumento.

Vejamos alguns exemplos de código e sua saída:

```
datahora.php > ...
1  ?php
2  //Definindo uma constante para pular linha
3  const PULA_LINHA = "<br/>";
4
5  #####FUNÇÃO date()#####
6  //A função date não possui o setlocale. O idioma será sempre o idioma do servidor
7  $data = date("d/m/Y H:i:s");
8  //Pesquise mais sobre as letras da função date em PHP Manual https://www.php.net/manual/pt\_BR/function.date.php
9  echo "Data e hora atual: $data".PULA_LINHA;
10 echo PULA_LINHA;
11
```

← → C ⌂ ⓘ localhost/deswebl/datahora.php

Data e hora atual: 11/11/2021 20:40:15

Mais informações sobre as opções de letras em:
https://www.php.net/manual/pt_BR/function.date.php

```
12 #####FUNÇÃO time() = Timestamp #####
13 //Timestamp é a quantidade de segundos decorridos desde 01/01/1970
14 $ts = time();
15 echo "Timestamp atual: $ts".PULA_LINHA;
16 echo PULA_LINHA;
17 //Usando um timestamp p/ representar uma data
18 $dataFixa = date("d/m/Y H:i:s", $ts);
19 echo "Data e hora fixados pelo timestamp $ts: $dataFixa".PULA_LINHA;
20 echo PULA_LINHA;
21
```

Timestamp atual: 1636659615

Data e hora fixados pelo timestamp 1636659615: 11/11/2021 20:40:15

```
22 #####FUNÇÃO strtotime()#####
23 //Você pode obter um timestamp a partir de uma data e utilizá-lo depois
24 //P formato é aaaa-mm-dd que equivale a 06/11/2021
25 $tsDiaEpi = strtotime("2021-11-06");
26 $meuAniversarioEm2021 = date("l, d/m/Y", $tsDiaEpi);
27 echo "Dia épico em 2021 com dia da semana (letra l): $meuAniversarioEm2021".PULA_LINHA;
28 echo PULA_LINHA;
```

Dia épico em 2021 com dia da semana (letra l): Saturday, 06/11/2021

```
30 //Outras opções para strtotime()
31 echo "Outras opções autoexplicativas para o timestamp:".PULA_LINHA;
32 $tsAgora = strtotime("now");
33 echo "Agora: ".date("l, d/m/Y H:i:s", $tsAgora).PULA_LINHA;
34 $tsMaisUmDia = strtotime("+1 day");
35 echo "Mais um dia: ".date("l, d/m/Y H:i:s", $tsMaisUmDia).PULA_LINHA;
36 $tsMaisUmaSemana = strtotime("+1 week");
37 echo "Mais uma semana: ".date("l, d/m/Y H:i:s", $tsMaisUmaSemana).PULA_LINHA;
38
39 ?>
```

Outras opções autoexplicativas para o timestamp:

Agora: Thursday, 11/11/2021 20:40:15

Mais um dia: Friday, 12/11/2021 20:40:15

Mais uma semana: Thursday, 18/11/2021 20:40:15

Exercícios de Fixação:

- 1) Teste todos os códigos acima.

49.Funções recursivas

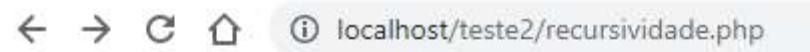
Definição: Funções recursivas são aquelas funções que chamam a si mesmas num looping infinito (o que não é recomendável) ou até que se satisfaça determinada condição.

Veja o exemplo de uma função simples que exibe um número:

Arquivo recursividade.php

```
recursividade.php > html
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <title>Funções recursivas</title>
6  </head>
7  <body>
8      <h2>Funções Recursivas</h2>
9      <hr/>
10     <span>Digite um número para exibir toda sua sequência até 1</span><br><br>
11     <form action="recursividade2.php" method="POST">
12         <label for="numero">Número</label>
13         <input type="number" name="numero" id="numero" min="1" max="10" default="1" /> <br/><br/>
14         <input type="submit" value="Enviar"/> <input type="reset" value="limpar"/>
15     </form>
16 </body>
17 </html>
```

Resultado no navegador



Funções Recursivas

Digite um número para exibir toda sua sequência até 1

Número

Arquivo recursividade2.php

```
recursividade2.php > html > body > exibeNumero
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <title>Funções recursivas</title>
6  </head>
7  <body>
8      <h2>Funções Recursivas</h2>
9      <hr/>
10     <?php
11         function exibeNumero(int $num){
12             echo "O valor passado para a função foi: $num<br>";
13         }
14         //Recuperando o campo na requisição
15         if( isset($_POST['numero'])){
16             exibeNumero($_POST['numero']);
17         } else {
18             echo "Valor inválido<br>";
19         }
20         <br>
21         <a href="recursividade.php">Voltar</a>
22
23     </body>
24 </html>
```

Resultado no navegador

← → C ⌂ ⓘ localhost/teste2/recursividade2.php

Funções Recursivas

O valor passado para a função foi: 7

[Voltar](#)

Obviamente, até o momento não há recursividade alguma. Vamos alterar a função exibeNúmero.

```
11         function exibeNumero(int $num){
12             echo "O valor passado para a função foi: $num<br>";
13             $numeroNovo = $num-1;
14             exibeNumero($numeroNovo);
15 }
```

Veja como ficaria no navegador



Funções Recursivas

```
O valor passado para a função foi: 7
O valor passado para a função foi: 6
O valor passado para a função foi: 5
O valor passado para a função foi: 4
O valor passado para a função foi: 3
O valor passado para a função foi: 2
O valor passado para a função foi: 1
O valor passado para a função foi: 0
O valor passado para a função foi: -1
O valor passado para a função foi: -2
O valor passado para a função foi: -3
O valor passado para a função foi: -4
O valor passado para a função foi: -5
O valor passado para a função foi: -6
O valor passado para a função foi: -7
O valor passado para a função foi: -8
O valor passado para a função foi: -9
O valor passado para a função foi: -10
O valor passado para a função foi: -11
O valor passado para a função foi: -12
O valor passado para a função foi: -13
O valor passado para a função foi: -14
```

Perceba que entraríamos em loop infinito quando na verdade queremos exibir apenas números positivos até 1 que sejam menores do que o número digitado.

O que faltou então?

Faltou o que é imprescindível em qualquer função recursiva. Uma condição de parada! Chame a si mesmo enquanto o argumento for maior que zero.

Veja a nova versão da função e em seguida o resultado no navegador.

```
11     function exibeNumero(int $num){
12         echo "O valor passado para a função foi: $num<br>";
13         $numeroNovo = $num-1;
14         if($numeroNovo>0)
15             exibeNumero($numeroNovo);
16     }
```

Resultado no navegador

← → C ⌂ ⓘ localhost/teste2/recursividade2.php

Funções Recursivas

O valor passado para a função foi: 7
O valor passado para a função foi: 6
O valor passado para a função foi: 5
O valor passado para a função foi: 4
O valor passado para a função foi: 3
O valor passado para a função foi: 2
O valor passado para a função foi: 1

[Voltar](#)

O código completo do arquivo recursividade2.php segue abaixo.

recursividade2.php > html > body

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <title>Funções recursivas</title>
6  </head>
7  <body>
8      <h2>Funções Recursivas</h2>
9      <hr/>
10     <?php
11         function exibeNumero(int $num){
12             echo "O valor passado para a função foi: $num<br>";
13             $numeroNovo = $num-1;
14             if($numeroNovo>0)
15                 exibeNumero($numeroNovo);
16         }
17         //Recuperando o campo na requisição
18         if( isset($_POST['numero']))
19             exibeNumero($_POST['numero']);
20         else
21             echo "Valor inválido<br>";
22     ?>
23     <br>
24     <a href="recursividade.php">Voltar</a>
25
26 </body>
27 </html>
```

Você certamente já estudou recursividade em outras linguagens com exemplos envolvendo a matemática, por exemplo. No entanto, quanto mais simples e menos complexo for o exemplo, maior é a possibilidade de compreensão do conceito a ser

ensinado.

Em PHP não utilizamos recursividade amplamente, mas há casos em que esse recurso pode ser útil.

Exercícios de fixação:

- 4) Teste todos os exemplos acima.

50.Hash e funções de hash

HASH

Hash é um resumo de tamanho fixo de um conteúdo de qualquer tamanho.

- Uma mudança em um único caractere no conteúdo faz o hash mudar totalmente;
- Hash's são usados para autenticação de conteúdo;
- Um conteúdo deve ter sempre o mesmo hash;

Funções de hash de string

- **crc32** – Hash numérico (retorna um inteiro)
- **md5** – Hash com 32 caracteres (retorna uma string)
- **sha1** – Hash com 40 caracteres (retorna uma string)
- **sha256 e sha512** – Hash com a quantidade de caracteres especificada.

Veja alguns exemplos:

```
$texto = 'Thiago';

echo 'CRC32: ', crc32( $texto ), '<br />';
echo 'MD5: ', md5( $texto ), '<br />';
echo 'SHA1: ', sha1( $texto ), '<br />';
```

Resultado:

```
CRC32: 1997885433
MD5: 6558db062ed0b23b8b01aee039bebff6
SHA1: 2d8eac8a00a187963367c2b3c7fe5d068d0e1615
```

Cuidados importantes

Nunca armazene o hash "puro" da senha no banco de dados.

Ao invés, concatene um texto à string original antes de realizar o hash.

Esse texto é usualmente chamado de "sal".

Na web há diversos sites com bancos de dados de hash pesquisáveis. Eles podem ser usados para descobrir senhas.

Exercícios de Fixação:

1) Teste todos os códigos acima

Funções de hash

Os hashes citados acima já foram amplamente utilizadas inclusive para armazenar senhas em bancos de dados. No entanto, atualmente a forma mais de criptografar e armazenar senhas em um banco de dados é por meio das funções de hash.

As funções de Hash servem para verificar a integridade de determinados dados e garantir que não tenham sido alterados inadvertidamente, como por exemplo, senhas de bancos de dados. Há vários algoritmos de hash que podem ser utilizados para essa finalidade. No PHP utilizaremos a função criptográfica bcrypt, que é baseada na cifra Blowfish, e implementada por meio da função “password_hash()“.

Função password_hash()

Sintaxe:

string password_hash (string \$senha , integer \$algoritmo, array \$opções)

Onde:

\$senha é a senha a ser hasheada.

\$algoritmo é o algoritmo de hash a ser utilizado no processo.

Os algoritmos suportados são:

PASSWORD_DEFAULT – algoritmo bcrypt. Se usado, recomenda-se armazenar o resultado do hash em um campo que permite até 255 caracteres, por questões de boas práticas.

PASSWORD_BCRYPT – Usa o algoritmo CRYPT_BLOWFISH para criar o hash, resultando em uma string de exatos 60 caracteres.

PASSWORD_ARGON2I – Utiliza o algoritmo Argon2 para a criação do hash.

\$opções é um array associativo que indica configurações especiais para a função. Estão disponíveis as duas funções a seguir:

- salt – permite fornecer manualmente um valor de sal à função – por padrão, o sal é gerado automaticamente, de forma aleatória. **É recomendado não gerar o sal manualmente. Note que esta opção foi deprecada no PHP 7.0.**
- cost – indica o custo algorítmico que deve ser utilizado. Por padrão, é usado o valor 10.

A função retorna uma string contendo o hash da senha, ou o valor FALSE se algo der errado. Tanto o cost quanto o salt são retornados como parte do próprio hash, de modo que não precisam ser armazenados separadamente.

Acrescentando sal ao dado criptografado.

Em criptografia, sal é um dado aleatório que é usado como uma entrada adicional para uma função que efetua o hash de uma senha. Trata-se de caracteres aleatórios acrescentados à senha antes de ela ser hasheada. A principal função do sal é defender a senha armazenada contra-ataques de dicionário ou contra-ataque de hashes pré-calculados, realizados por uso de rainbow tables (pesquise sobre isso).

Função password_verify()

Essa função verifica se uma senha fornecida confere com um hash armazenado.

Sintaxe:

```
boolean password_verify(string $senha, string $hash_da_senha)
```

Onde:

\$senha é a senha fornecida pelo usuário
\$hash_da_senha é o hash da senha do usuário, criado com a função `password_verify()`, que está armazenado em algum lugar, como um banco de dados.
A função retorna TRUE se a senha conferir com o hash, ou FALSE em caso contrário.

Veja um exemplo bem didático da utilização das duas funções:

```
hash.php > ...
1  <?php
2  echo "Criando hash de senha com sal e testando:<br>";
3  $hashSenha = password_hash('rafael_guimaraes_rodrigues', PASSWORD_DEFAULT);
4  // $hashSenha = password_hash('rafael_guimaraes_rodrigues', PASSWORD_BCRYPT);
5  // $hashSenha = password_hash('rafael_guimaraes_rodrigues', PASSWORD_BCRYPT, ['cost => 20']); //20
6  // $hashSenha = password_hash('rafael_guimaraes_rodrigues', PASSWORD_ARGON2I);
7  // $hashSenha = password_hash('rafael_guimaraes_rodrigues', PASSWORD_ARGON2I, ['cost => 12']); //12
8  if (password_verify('rafael_guimaraes_rodrigues', $hashSenha)) {
9      echo "Senha correta<br>";
10     echo "O Hash da senha é $hashSenha <br>";
11     echo "O Hash ocupa " . strlen($hashSenha) . " caracteres<br>";
12 }
13 else {
14     echo "Senha incorreta!";
15 }
?>
```

Fonte de consulta: <http://www.bosontreinamentos.com.br/php-programming/armazenando-senhas-de-forma-segura-em-php-funcoes-de-hash/>

Exercícios de Fixação:

- 1) Teste todos os códigos acima “descomentando” e comentando linhas e experimentando cada algoritmo.

51.Listas

No PHP as listas são utilizadas para realizar atribuições múltiplas. Com o uso de listas é possível atribuir os valores que estão em um array para um conjunto de variáveis. No entanto, para que a atribuição seja bem-sucedida é necessário que o array possua índices inteiros e não negativos. Veja exemplo:

```
list( $a, $b, $c ) = array( 10, 20, 30 );
//Atribuição bem sucedida
echo "RESULTADO DA 1ª ATRIBUIÇÃO: \$a = {$a}, \$b = {$b}, \$c = {$c}<br/>";

$arr = array( 1 => "um", 3 => "tres", "a" => "letraA", 2 => "dois" );
list( $a, $b, $c, $d ) = $arr;
//Atribuição parcialmente bem sucedida (com problemas de compilação)
echo "RESULTADO DA 2ª ATRIBUIÇÃO: \$a = {$a}, \$b = {$b}, \$c = {$c}, \$d = {$d}<br/>";
```

Veja a saída:

RESULTADO DA 1ª ATRIBUIÇÃO: \$a = 10, \$b = 20, \$c = 30

(!) Notice: Undefined offset: 0 in D:\wamp\www\aulas\teste17.php on line 7				
Call Stack				
#	Time	Memory	Function	Location
1	0.0000	241280	{main}()	..\teste17.php:0

RESULTADO DA 2ª ATRIBUIÇÃO: \$a = , \$b = um, \$c = dois, \$d = tres

Na 1ª atribuição não tivemos problemas já que ao criar um array sem especificar os índices, valores inteiros (a partir do zero) são atribuídos para os mesmos.

Na 2ª atribuição tivemos problemas porque não foi encontrado o índice 0 para atribuir valor à 1ª variável que é \$a. Os outros índices foram encontrados e as atribuições foram feitas.

IMPORTANTE: Os índices da lista servem apenas como referência para o interpretador PHP. Não podem ser acessados pelo programador.

Exercícios de Fixação:

- 1) Teste todos os códigos acima com arquivos reais.51.Trabalhando com data e hora

52.Configuração regional com strftime e setlocale()

A adicionar....

53.DateTime

A adicionar....

54.PDO

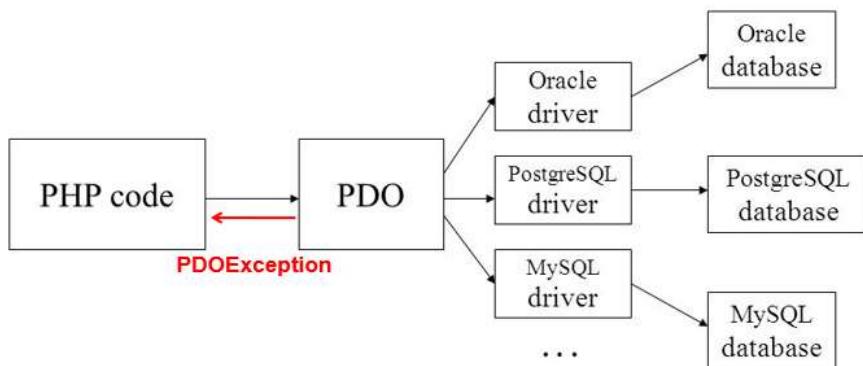
Atenção: Antes de ler este capítulo, assista à vídeo aula “01 - PDO – INTRODUÇÃO”.

No passado havia várias maneiras de se conectar com um banco de dados via PHP utilizando funções distintas para se conectar e manipular diversos SGBDs como MySQL, SQL Server, Postgres etc.

Pensando apenas no SGBD MySQL, tínhamos, só para conectar com o banco, algumas opções tais como `mysql_connect`, posteriormente `mysqli_connect` e desde algumas versões a biblioteca PDO. A partir da versão 7, se considerarmos o SGBD MySQL, temos apenas a biblioteca PDO e a classe `Mysqli` à disposição. Na verdade, o PDO também é uma classe. Vocês entenderão melhor esse conceito mais adiante, quando estivermos abordando Orientação a Objetos.

A vantagem do PDO (PHP Data Object) é que ele padroniza a forma de se conectar com diversos SGBDs diferentes. Independentemente de estar lidando com MySQL, Oracle, Postgres ou qualquer outro SGBD, as funções para conectar com a base de dados e as funções para executar comandos SQL são as mesmas.

A classe PDO fornece uma camada de abstração em relação a conexão com o banco de dados. A partir dessa classe você cria um objeto do tipo PDO que é utilizado para fazer a conexão com diversos bancos de dados da mesma maneira, modificando apenas a sua string de conexão. Não se preocupe em entender essas questões sobre classes e objetos por enquanto. Você pode entender melhor essa abstração na figura abaixo.



Na figura, através de um script PHP (PHP code), você solicita uma conexão com uma base de dados de um determinado SGBD via PDO e pode receber a conexão com o SGBD solicitado em caso de sucesso, ou um erro conhecido como PDOException. Esse erro, na verdade, é o que chamamos de Exceção. Abordaremos esse assunto mais adiante. Um “erro” do tipo PDOException pode ocorrer por diversos fatores. Dentre eles é possível destacar:

- Host inexistente;
- Base de dados inexistente;
- Usuário inexistente;
- Senha errada;
- Erro de sintaxe ao usar o método/ função para efetuar a conexão.

Não ocorrendo nenhum dos erros acima, há uma chance muito grande de você obter uma conexão com a base de dados desejada. Essa conexão é um objeto do tipo PDO configurado para trabalhar com aquela base de dados específica.

Observe atentamente o código abaixo e seus comentários:

```
conexao.php > ...
1  <?php
2      /*Para obter uma conexão via PDO, você precisa fornecer 3 informações:
3          **Data source Name --> representa uma string de conexão onde você define
4              o SGBD(mysql), um host(localhost) e uma base de dados(test);
5              A sintaxe é "sgbd:host=nomeOuIpDoHost;dbname=nomeDaBaseDeDados"
6
7          **Usuário do SGBD('root');
8          **Senha do SGBD('') No nosso caso estamos usando o usuário root sem senha.
9      */
10     $dsn = "mysql:host=localhost;dbname=test";
11     $user = "roott";
12     $pass = "";
13     try{
14         $conexao = new PDO($dsn,$user,$pass); //Linha de interrupção
15         echo "Conexao efetuada com sucesso!";
16         echo "<hr><pre>";
17         var_dump($conexao);
18         echo "</pre>";
19     }catch(PDOException $e){
20         echo "<hr>";
21         echo "Codigo do erro: {$e->getCode()} - Mensagem de erro: {$e->getMessage()}";
22         echo "<hr><pre>";
23         print_r($e);
24         echo "</pre>";
25     }
}
```

Na linha 14 eu tento obter uma conexão com a base de dados test do SGBD MySQL que fica em localhost. Para tal uso as credenciais “roott” e “” (usuário roott sem senha).

Para obter um objeto do tipo PDO eu preciso usar a palavra chave new e passar os 3 argumentos necessários. Em geral, a base de dados test já existe por padrão no MySQL. Se por acaso não existir, use seus conhecimentos adquiridos na disciplina banco de dados para criar essa base de dados.

Como estou sujeito a receber um erro conhecido como PDOException, eu preciso envolver o comando em uma estrutura try/catch que, em linhas gerais, significa:

- Tente obter uma conexão via PDO (try).
- Se tudo der certo, exiba uma mensagem de sucesso e o var_dump do objeto PDO recebido e armazenado na variável de referência \$conexao.
Todo objeto em PHP é uma referência, um endereço de memória. Logo, toda variável que armazena um objeto é, na verdade, uma variável de referência.
- Se não der certo (catch), tente capturar um objeto do tipo PDOException que estará representado pela variável de referência \$e. Imprima o código do erro (através do método getCode()) e a mensagem de erro (através do método getMessage()). Use o print_r para mostrar todas as informações contidas no objeto PDOException representado por \$e. Você vai perceber que esse objeto possui várias informações. Dentre elas temos code e message que são retornados, respectivamente, por getCode() e getMessage().

No código acima eu inseri um erro proposital. O usuário roott não existe. Salve o script conexão.php na pasta pdo dentro de htdocs, rode no navegador e veja o resultado.

```
PDOException Object
(
    [message:protected] => SQLSTATE[HY000] [1045] Access denied for user 'roott'@'localhost' (using password: NO)
    [string:Exception:private] =>
    [code:protected] => 1045
    [file:protected] => C:\xampp\htdocs\pdo\conexao.php
    [line:protected] => 14
    [trace:Exception:private] => Array
        (
            [0] => Array
                (
                    [file] => C:\xampp\htdocs\pdo\conexao.php
                    [line] => 14
                    [function] => __construct
                    [class] => PDO
                    [type] => -
                    [args] => Array
                        (
                            [0] => mysql:host=localhost;dbname=test
                            [1] => roott
                            [2] =>
                        )
                )
        )

    [previous:Exception:private] =>
    [errorInfo] => Array
        (
            [0] => HY000
            [1] => 1045
            [2] => Access denied for user 'roott'@'localhost' (using password: NO)
        )
)
```

O print_r e as tags “<pre>” ajudam a mostrar as informações do objeto \$e de forma mais organizada. Dentre as informações contidas no objeto, podemos identificar até mesmo em que linha ocorreu o erro. No atributo line podemos perceber que o erro ocorreu na linha 14. Se quisermos exibir esse erro, devemos usar o método getLine() do objeto \$e. Percebam que code, message e line podem ser obtidos pelos métodos getCode(), getMessage() e getLine() respectivamente.

IMPORTANTE: O que chamamos de atributos, vocês conhecem como variáveis e o que chamamos de métodos vocês conhecem como funções. Essas nomenclaturas são próprias do paradigma orientado a objetos que veremos mais adiante. Como PDO e PDOException são objetos, eles possuem atributos (características) e métodos (geralmente funções para retornar essas características). Não se preocupe em entender isso a fundo neste momento.

Em seguida, na linha 11 do script, troque o valor de \$user para “root” (um usuário válido) e rode novamente. Você vai ver, dessa vez, o que está programado dentro do try, ou seja, vai ver as mensagens de sucesso! Veja o resultado abaixo:

```
Conexao efetuada com sucesso!

object(PDO)#1 (0) {
```

O resultado do var_dump(\$conexao) contido na linha 17 é que ali temos uma informação que é um objeto do tipo PDO e que está no endereço de memória #1. É claro que ‘#1’ é apenas uma forma simplificada que o PHP utilizou para nos mostrar um endereço de memória que sabemos ser bem mais complexo do que isso. Agora que já temos uma conexão com a base de dados test, podemos executar, via PDO, diversos comandos.

Podemos:

- Criar uma base de dados (CREATE DATABASE);
- Criar uma tabela (CREATE TABLE);
- Alterar a estrutura de uma tabela (ALTER TABLE);
- Remover uma base de dados, uma tabela ou uma coluna (DROP DATABASE, DROP TABLE, DROP COLUMN);
- Inserir dados em uma tabela (INSERT);
- Alterar dados de uma ou mais linhas de uma tabela (UPDATE provavelmente junto com uma cláusula WHERE);
- Remover linha(s) de uma tabela (DELETE provavelmente junto com uma cláusula WHERE);
- Exibir determinados dados de uma ou mais linhas de uma tabela (SELECT).

Basicamente, podemos executar qualquer comando SQL a partir de agora!

Criando uma tabela na base de dados test.

Vamos usar o mesmo script para obter uma conexão com o banco de dados e criar uma tabela na base de dados test. As alterações começam a partir da linha 16. Veja como fica o código:

```
13 try{  
14     $conexao = new PDO($dsn,$user,$pass); //Linha de interrupção  
15     echo "Conexao efetuada com sucesso!<br>";  
16     /*echo "<br><pre>";  
17     var_dump($conexao);  
18     echo "</pre>";*/  
19 }catch(PDOException $e){  
20     echo "<br>";  
21     echo "Codigo do erro: {$e->getCode()} - Mensagem de erro: {$e->getMessage()}";  
22     echo "<br><pre>";  
23     print_r($e);  
24     echo "</pre>";  
25 }  
26  
27 $comandoSql = "CREATE TABLE user(id int auto_increment primary key,  
28 login varchar(10) not null, senha char(6) not null)engine=INNODB CHARSET=utf8 COLLATE=utf8_unicode_ci";  
29 $retorno = $conexao->exec($comandoSql); //vai retornar um inteiro se der certo ou false se houver erro  
30 var_dump($retorno);  
31 if($retorno != false){  
32     echo "<br>tabela criada com sucesso!";  
33 }else{  
34     echo "<br>Erro ao executar o comando sql.";  
35 }  
36 ?>
```

Observações sobre o código:

Linhas 27 e 28 → Armazenamos na variável \$comandoSql o comando necessário para a criação de uma tabela;

Linha 29 → Na variável \$retorno eu armazeno o retorno do método exe do meu objeto PDO \$conexao. Esse método recebe como argumento uma string que representa um comando em sql. Nesse caso, o conteúdo da variável \$comandoSql. Se tudo der certo, o método deverá retornar um número inteiro com a quantidade de linhas afetadas. Como se trata de um comando de criação de uma tabela, o resultado será 0. Se houver algum erro, o método retornará false.

Linhas 31 a 35 → Verificação do conteúdo da variável retorno para exibir mensagem de sucesso ou de erro.

Ao executar o script pela primeira vez, o resultado deverá ser o seguinte:

The browser window shows the URL `localhost/pdo/conexao.php`. The page content includes the message "Conexao efetuada com sucesso!" followed by "int(0)" and "tabela criada com sucesso!".

The second part of the image shows the PhpMyAdmin interface for the "test" database. The "Tabelas" (Tables) section lists the following tables:

Tabela	Ações	Registros	Tipo	Agrupamento (Collate)
alunos	Procurar Estrutura Pesquisar Insere Limpa Elimina	354	InnoDB	utf8_general_ci
table 1	Procurar Estrutura Pesquisar Insere Limpa Elimina	227	InnoDB	utf8_general_ci
table 3	Procurar Estrutura Pesquisar Insere Limpa Elimina	108	InnoDB	utf8_general_ci
table 4	Procurar Estrutura Pesquisar Insere Limpa Elimina	214	InnoDB	utf8_general_ci
table 7	Procurar Estrutura Pesquisar Insere Limpa Elimina	215	InnoDB	utf8_general_ci
table 8	Procurar Estrutura Pesquisar Insere Limpa Elimina	29	InnoDB	utf8_general_ci
table 12	Procurar Estrutura Pesquisar Insere Limpa Elimina	11	InnoDB	utf8_general_ci
user	Procurar Estrutura Pesquisar Insere Limpa Elimina	0	InnoDB	utf8_unicode_ci
vagrupado	Procurar Estrutura Pesquisar Insere Limpa Elimina	0	MyISAM	latin1_swedish_ci
valunos	Procurar Estrutura Pesquisar Insere Limpa Elimina	0	MyISAM	latin1_swedish_ci
vassinaturas	Procurar Estrutura Pesquisar Insere Limpa Elimina	0	MyISAM	latin1_swedish_ci
vw_alunos	Procurar Estrutura Pesquisar Insere Limpa Elimina	0	MyISAM	latin1_swedish_ci
vw_alunoss	Procurar Estrutura Pesquisar Insere Limpa Elimina	0	InnoDB	latin1_swedish_ci

Total: 13 tabelas Soma: 1,158 InnoDB latin1_swedish_ci

Na imagem acima é possível perceber que, de fato, a tabela user foi criada. Se você clicar na tabela, verá que a estrutura é a mesma projetada em nosso comando SQL.

O que acontece se eu executar o mesmo script uma segunda vez?

O resultado vai ser um erro do tipo PDOException visto que a tabela a ser criada, dessa vez já existe! Perceba que dessa vez optamos por não tratar o erro. Por isso recebemos a mensagem de erro padrão também conhecida como rastro da Exceção. Não se preocupe com isso por enquanto.

The browser window shows the URL `localhost/pdo/conexao.php`. The page content includes the message "Conexao efetuada com sucesso!" followed by an error message: "Fatal error: Uncaught PDOException: SQLSTATE[42S01]: Base table or view already exists: 1050 Table 'user' already exists in C:\xampp\htdocs\pdo\conexao.php:29 Stack trace: #0 C:\xampp\htdocs\pdo\conexao.php(29): PDO->exec('CREATE TABLE us...') #1 {main} thrown in C:\xampp\htdocs\pdo\conexao.php on line 29".

No MySQL, podemos adicionar o IF NOT EXISTS logo após o CREATE TABLE.

Assim o SGBD só tentará criar uma determinada tabela se ela ainda não existir. \

Veja o código com alterações nas linhas 27 e 32.

```
27 $comandoSql = "CREATE TABLE IF NOT EXISTS user(id int auto_increment primary key,
28 login varchar(10) not null, senha char(6) not null)engine=INNODB CHARSET=utf8 COLLATE=utf8_unicode_ci";
29 $retorno = $conexao->exec($comandoSql); //vai retornar um inteiro se der certo ou false se houver erro
30 var_dump($retorno);
31 if($retorno != false){
32     echo "<br>A tabela foi criada com sucesso ou já existia!";
33 }else{
34     echo "<br>Erro ao executar o comando sql.";
35 }
36 ?>
```

Ao rodar o script novamente, o SGBD nem tenta criar a tabela pois percebe que ela já existe. Veja o resultado exibido no navegador. Dessa vez, sem receber um PDOException.

Conexao efetuada com sucesso!
int(0)
A tabela foi criada com sucesso ou já existia!

Exercícios de fixação.

- 1) Teste todos os códigos deste capítulo na ordem em que são apresentados.

55. Realizando CRUD com PDO

Atenção: Antes de ler este capítulo, assista à vídeo aula “02 - PDO – CRUD”.

O que chamamos de CRUD na verdade é a representação das 4 operações básicas em banco de dados. CRUD é o acrônimo do inglês Create, Read, Update and Delete que representa respectivamente as operações necessárias para inserir, ler, alterar e remover linhas e dados de determinada tabela do seu banco de dados. Portanto, mostraremos como realizar essas 4 operações via PDO utilizando nossa tabela user da base test criada anteriormente.

No entanto, é hora de começar a organizar melhor nosso código. O arquivo conexão.php deve ser utilizado toda vez que eu precisar de uma conexão com o banco de dados. Portanto, comentaremos todo o código desnecessário e incluiremos esse arquivo através de require_once sempre que necessário. Veja como deve ficar o arquivo conexão.php com comandos comentados a partir da linha 22.

```
9  */
10 $dsn = "mysql:host=localhost;dbname=test";
11 $user = "root";
12 $pass = "";
13 try{
14     $conexao = new PDO($dsn,$user,$pass); //Linha de interrupção
15     echo "Conexao efetuada com sucesso!<br>";
16     /*echo "<hr><pre>";
17     var_dump($conexao);
18     echo "</pre>";*/
19 }catch(PDOException $e){
20     echo "<hr>";
21     echo "Codigo do erro: {$e->getCode()} - Mensagem de erro: {$e->getMessage()}";
22     /*echo "<hr><pre>";
23     print_r($e);
24     echo "</pre>";*/
25 }
26
27 /*$comandoSql = "CREATE TABLE IF NOT EXISTS user(id int auto_increment primary key,
28 login varchar(10) not null, senha char(6) not null)engine=INNODB CHARSET=utf8 COLLATE=utf8_unicode_ci";
29 $retorno = $conexao->exec($comandoSql); //vai retornar um inteiro se der certo ou false se houver erro
30 var_dump($retorno);
31 if($retorno !== false){
32     echo "<br>A tabela foi criada com sucesso ou já existia!";
33 }else{
34     echo "<br>Erro ao executar o comando sql.";
35 }*/
36 ?>
```

Rafael, por que você não removeu o comando create contido nas linhas 27 a 35?

Eu não removi para que vocês guardem um exemplo de como executar um comando CREATE TABLE. 😊

No arquivo comandoExec.php, veja como executar os comandos INSERT, UPDATE e DELETE.

INSERT

```
 comandoExec.php > ...
1  <?php
2      require_once('conexao.php');//Incluindo o arquivo responsável por obter a conexão
3      //INSERT responsável por inserir 5 linhas na tabela
4      $comandoSql = "INSERT INTO user(login,senha) VALUES('rafael','123456'),
5      ('renata','123457'),('fulano','123458'),('bia','123459'),('rebeca','123410')";
6      $retorno = $conexao->exec($comandoSql); // $retorno contém a quantidade de linhas afetadas
7      if($retorno != false){
8          echo "<br>Foram inseridas {$retorno} linhas!";
9      }else{
10         echo "<br>Erro ao executar o comando sql.";
11     }
```

Ao rodar o script no navegador, o resultado deverá ser o seguinte:

← → C ⌂ localhost/pdo/comandoExec.php

Conexao efetuada com sucesso!

Foram inseridas 5 linhas!

Para ter certeza, faça uma consulta no phpMyAdmin.
O resultado deve ser parecido com a imagem abaixo.

Mostrar Caixa do query

MySQL não retornou nenhum registo. (A consulta demorou 0,0006 segundos.)

use test

A mostrar registo de 0 - 4 (5 total, A consulta demorou 0,0041 segundos.)

select * from user

Mostrar tudo | Número de registo: 25 ▾ Filtrar registo: Pesquisar esta tabela

+ Opções

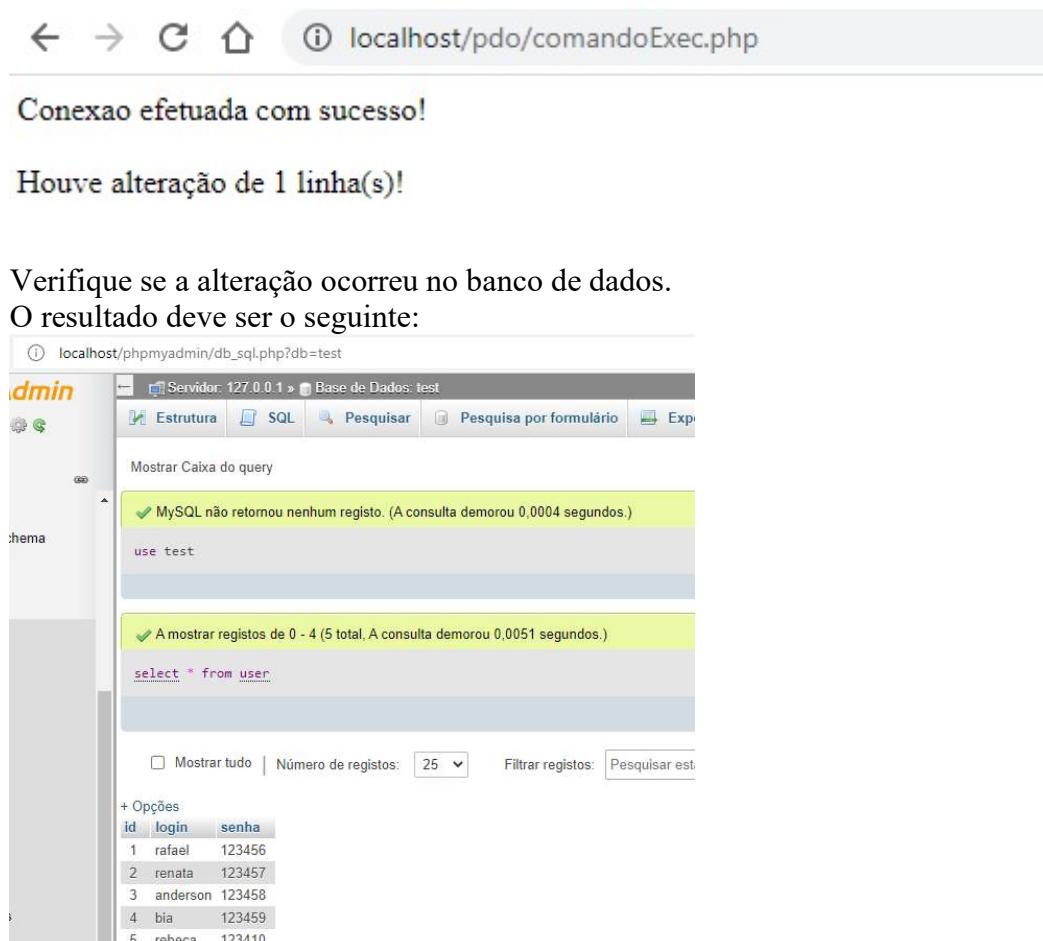
	id	login	senha
1	rafael	123456	
2	renata	123457	
3	fulano	123458	
4	bia	123459	
5	rebeca	123410	

UPDATE

Obs: Não se esqueça de comentar todo o comando INSERT(linhas 4 a 11)!

```
 comandoExec.php > ...
1 <?php
2     require_once('conexao.php');//Incluindo o arquivo responsável por obter a conexão
3 //INSERT responsável por inserir 5 linhas na tabela
4 /*$comandoSql = "INSERT INTO user(login,senha) VALUES('rafael','123456'),
5 ('renata','123457'),('fulano','123458'),('bia','123459'),('rebeca','123410')";
6 $retorno = $conexao->exec($comandoSql); // $retorno contém a quantidade de linhas afetadas
7 if($retorno !== false){
8     echo "<br>Foram inseridas {$retorno} linhas!";
9 }else{
10    echo "<br>Erro ao executar o comando sql.";
11 }
12 */
13 //UPDATE
14 $comandoSql = "UPDATE user SET login = 'anderson' WHERE login='fulano'";
15 $retorno = $conexao->exec($comandoSql); // $retorno contém a quantidade de linhas afetadas
16 if($retorno !== false){
17     echo "<br>Houve alteração de {$retorno} linha(s)!";
18 }else{
19     echo "<br>Erro ao executar o comando sql.";
20 }
```

Execute novamente o script no navegador. O resultado deve ser o seguinte:

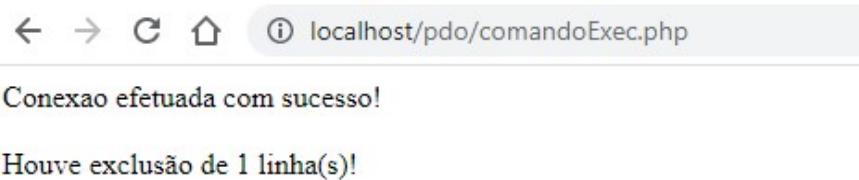


DELETE

Obs: Não se esqueça de comentar todo o comando UPDATE(linhas 14 a 20)! Mesmo se você esquecesse de comentar, nada aconteceria. Afinal, não existe mais nenhuma linha cujo login seja ‘fulano’.

```
13 //UPDATE
14 /*$comandoSql = "UPDATE user SET login = 'anderson' WHERE login='fulano'";
15 $retorno = $conexao->exec($comandoSql); // $retorno contém a quantidade de linhas afetadas
16 if($retorno !== false){
17     echo "<br>Houve alteração de {$retorno} linha(s)!";
18 }else{
19     echo "<br>Erro ao executar o comando sql.";
20 }*/
21
22 //DELETE
23 $comandoSql = "DELETE FROM user WHERE (id=3 OR login='anderson')";
24 $retorno = $conexao->exec($comandoSql); // $retorno contém a quantidade de linhas afetadas
25 if($retorno !== false){
26     echo "<br>Houve exclusão de {$retorno} linha(s)!";
27 }else{
28     echo "<br>Erro ao executar o comando sql.";
29 }
30 ?|
```

Execute novamente o script no navegador. O resultado deve ser o seguinte:



Verifique se a alteração ocorreu no banco de dados.
O resultado deve ser o seguinte:

The screenshot shows the phpMyAdmin interface for the 'test' database. In the SQL tab, the command `use test` is run, followed by `select * from user`. The results show that the row where `login = 'anderson'` has been deleted, leaving 4 rows in the table.

id	login	senha
1	rafael	123456
2	renata	123457
4	bia	123459
5	rebeca	123410

SELECT com fetchAll()

Obs: Não se esqueça de comentar o comando DELETE!

Leia atentamente os comentários! Se tiver dúvidas, acione o professor (Eu)!

```
22 //DELETE
23 /*$comandoSql = "DELETE FROM user WHERE (id=3 OR login='anderson')";
24 $retorno = $conexao->exec($comandoSql); //$retorno contém a quantidade de linhas afetadas
25 if($retorno !== false){
26     echo "<br>Houve exclusão de {$retorno} linha(s)!";
27 }else{
28     echo "<br>Erro ao executar o comando sql.";
29 }*/
30
31 //SELECT com fetch
32 $comandoSql = "SELECT * FROM user order by login";
33 $stmt = $conexao->query($comandoSql); //Aqui eu não uso mais exec pois não estou modificando nada.
34 //Estou apenas buscando informações na tabela. Portanto, trata-se de uma consulta. Uma query
35 //stmt recebe um PDOStatement (Uma instrução preparada)
36 $usuarios = $stmt->fetchAll(); //usuarios recebe as informações fatiadas(fetchAll()) na forma de uma matriz associativa
37 echo "Como eu ordenei pelo login....<br/>";
38 //Pegando uma linha pelo índice e suas colunas pelo nome
39 echo "O primeiro usuário tem id {$usuarios[0]['id']} e login {$usuarios[0]['login']}<br/>";
40 //Pegando uma linha pelo índice e suas colunas pelo índice
41 echo "O segundo usuário tem id {$usuarios[1][0]} e login {$usuarios[1][1]}<br/>";
42 echo "<pre>";
43 print_r($usuarios);
44 echo "</pre>";
45 echo "<br>";
46
47 ?>
```

Execute novamente o script no navegador. O resultado deve ser o seguinte:

Conexao efetuada com sucesso!
Como eu ordenei pelo login....
O primeiro usuário tem id 4 e login bia
O segundo usuário tem id 1 e login rafael

```
Array
(
    [0] => Array
        (
            [id] => 4
            [0] => 4
            [login] => bia
            [1] => bia
            [senha] => 123459
            [2] => 123459
        )

    [1] => Array
        (
            [id] => 1
            [0] => 1
            [login] => rafael
            [1] => rafael
            [senha] => 123456
            [2] => 123456
        )

    [2] => Array
        (
            [id] => 5
            [0] => 5
            [login] => rebeca
            [1] => rebeca
            [senha] => 123410
            [2] => 123410
        )
)
```

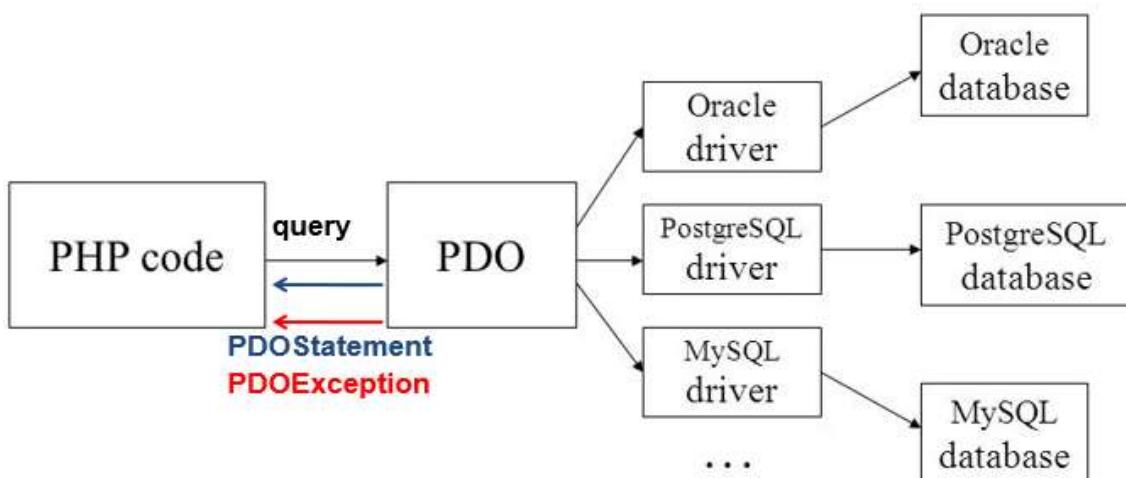
Exercícios de fixação.

- 1) Teste todos os códigos deste capítulo na ordem em que são apresentados.
- 2) Como já é sabido, o comando `$conexao->exec` pode retornar um `PDOException`. Trate esse erro com `try/catch` em todas as 4 operações. Você mesmo vai perceber que fazendo isso, os “if-elses” se tornarão desnecessários.

56.PDO – query e os tipos de fetch

Atenção: Antes de ler este capítulo, assista à vídeo aula “03 - PDO - QUERY e os tipos de FETCH”.

Neste capítulo trataremos do comando `query` e dos tipos de `fetch`, ou seja, dos formatos de retorno fatiado de uma consulta. Como pode ser visto na figura abaixo, ao executar o comando `query` você pode receber um `PDOStatement` (um conjunto de linhas e colunas que ainda precisa ser fatiado) em caso de sucesso ou um “erro” do tipo `PDOException` caso algo tenha saído errado.



O método `fetchAll()`

Abaixo podemos ver a imagem do arquivo `comandoQuery.php`. Observe attentamente suas linhas e posteriormente o resultado de sua execução. Ao executar o comando `query`, temos o resultado ainda em seu estado “bruto” guardado na variável `$stmt`.

Ao executar o método `fetchAll()` a partir de `$stmt`, obtemos o resultado fatiado em um array bidimensional onde cada linha da tabela possui um outro array com as colunas. Observe que o resultado do `print_r` mostra isso de forma bem clara.

Na linha 12, por exemplo, é possível verificar como podemos extrair dados da primeira linha desse array bidimensional (matriz).

```
1 <?php
2     require_once('conexao.php');
3     //SELECT com fetchAll
4     $comandoSql = "SELECT * FROM user ORDER BY login";
5     try{
6         $stmt = $conexao->query($comandoSql);
7         $usuarios = $stmt->fetchAll(); //Recebe um PDOStatement
8         echo "<pre>";
9         print_r($usuarios);
10        echo "</pre>";
11        echo "<hr>";
12        echo "O usuário de id {$usuarios[0]['id']} é {$usuarios[0]['login']}";
13    }catch(PDOException $e){
14        echo "Erro ao executar o comando.<br/>
15        <span style='color:red'>{$e->getMessage()}</span>.";
```

Ao executar o comando no navegador, temos o resultado abaixo.

← → C ⌂ ① localhost/pdo/comandoQuery.php

Conexão efetuada com sucesso!

```
Array
(
    [0] => Array
        (
            [id] => 4
            [0] => 4
            [login] => bia
            [1] => bia
            [senha] => 123459
            [2] => 123459
        )
    [1] => Array
        (
            [id] => 1
            [0] => 1
            [login] => rafael
            [1] => rafael
            [senha] => 123456
            [2] => 123456
        )
    [2] => Array
        (
            [id] => 5
            [0] => 5
            [login] => rebeca
            [1] => rebeca
            [senha] => 123410
            [2] => 123410
        )
    [3] => Array
        (
            [id] => 2
            [0] => 2
            [login] => renata
            [1] => renata
            [senha] => 123457
            [2] => 123457
        )
)
```

O usuário de id 4 é bia

Logo abaixo fizemos uma alteração na linha 4 do código para forçar um erro (usuário inexistente) e ver o que acontece ao cair no catch(PDOException \$e).

```
1 <?php
2     require_once('conexao.php');
3     //SELECT com fetchAll
4     $comandoSql = "SELECT * FROM userXXXX ORDER BY login";
5     try{
6         $stmt = $conexao->query($comandoSql);
7         $usuarios = $stmt->fetchAll(); //Recebe um PDOStatement
8         echo "<pre>";
9         print_r($usuarios);
10        echo "</pre>";
11        echo "<hr>";
12        echo "O usuário de id {$usuarios[0]['id']} é {$usuarios[0]['login']}";
13    }catch(PDOException $e){
14        echo "Erro ao executar o comando.<br/>
15            <span style='color:red'>{$e->getMessage()}</span>.";
```

← → C ⌂ ⓘ localhost/pdo/comandoQuery.php

Conexão efetuada com sucesso!

Erro ao executar o comando.

SQLSTATE[42S02]: Base table or view not found: 1146 Table 'test.userxxxx' doesn't exist.

O método fetch()

A diferença básica entre os métodos fetch() e fetchAll() é que o primeiro tem por objetivo pegar apenas o primeiro registro/linha retornado. Por essa razão, o comando deve ser executado a partir de um SELECT com cláusula WHERE que limite o resultado a uma linha apenas.

Observe o resultado do print_r e verá que nesse caso não temos mais um array bidimensional, mas sim um array simples.

```
17 //SELECT com fetch
18 try{
19     $comandoSql = "SELECT * FROM user WHERE id=2";
20     $stmt = $conexao->query($comandoSql);
21     $usuario = $stmt->fetch(); //Recebe um PDOStatement
22     echo "<pre>";
23     print_r($usuario);
24     echo "</pre>";
25     echo "<hr>";
26     echo "O usuário de id {$usuario['id']} é {$usuario['login']}";
27 }catch(PDOException $e){
28     echo "Erro ao executar o comando.<br/>
29         <span style='color:red'>{$e->getMessage()}</span>.";
```

← → C ⌂ ⓘ localhost/pdo/comandoQuery.php

Conexão efetuada com sucesso!

Array

```
( [id] => 2
  [0] => 2
  [login] => renata
  [1] => renata
  [senha] => 123457
  [2] => 123457
)
```

O usuário de id 2 é renata

Definindo o formato do fetchAll() com as constantes PDO::FETCH_ASSOC, PDO::FETCH_OBJ e PDO::FETCH_NUM

PDO::FETCH_ASSOC (Matriz associativa)

```
32 //SELECT com fetchAll e PDO::FETCH_ASSOC
33 try{
34     $comandoSql = "SELECT * FROM user order by login";
35     $stmt = $conexao->query($comandoSql);
36     $usuarios = $stmt->fetchAll(PDO::FETCH_ASSOC); //Recebe um PDOStatement
37     $linha = "O usuário de id {$usuarios[0]['id']} é {$usuarios[0]['login']}";
38     echo "<pre>";
39     print_r($usuarios);
40     echo "</pre>";
41     echo "<hr>";
42     echo $linha;
43 }catch(PDOException $e){
44     echo "Erro ao executar o comando.<br/>
45     <span style='color:red'>{$e->getMessage()}</span>.";
```

Perceba que diferentemente do que acontece com o fetchAll() sem argumentos, ao utilizar a constante PDO::FETCH_ASSOC, continuaremos a ter um array bidimensional, mas apenas com os nomes das colunas em sua 2^a dimensão. Não teremos mais os índices, como acontece com o fetchAll() sem argumentos. Observe!

Conexao efetuada com sucesso!

```
Array
(
    [0] => Array
        (
            [id] => 4
            [login] => bia
            [senha] => 123459
        )
    [1] => Array
        (
            [id] => 1
            [login] => rafael
            [senha] => 123456
        )
    [2] => Array
        (
            [id] => 5
            [login] => rebeca
            [senha] => 123410
        )
    [3] => Array
        (
            [id] => 2
            [login] => renata
            [senha] => 123457
        )
)
```

O usuário de id 4 é bia

PDO::FETCH_NUM (Array bidimensional)

A diferença básica é que na 2^a dimensão não temos mais os nomes das colunas. Apenas os seus índices (suas posições). A desvantagem é que você precisa saber em que posição cada informação está organizada. Veja abaixo o código, a forma de acessar (linha 37) e o resultado.

```
32 //SELECT com fetchAll e PDO::FETCH_ASSOC, PDO::FETCH_NUM e PDO::FETCH_OBJ
33 try{
34     $comandoSql = "SELECT * FROM user order by login";
35     $stmt = $conexao->query($comandoSql);
36     $usuarios = $stmt->fetchAll(PDO::FETCH_NUM); //Recebe um PDOStatement
37     $linha = "O usuário de id {$usuarios[0][0]} é {$usuarios[0][1]}";
38     echo "<pre>";
39     print_r($usuarios);
40     echo "</pre>";
41     echo "<hr>";
42     echo $linha;
43 }catch(PDOException $e){
44     echo "Erro ao executar o comando.<br/>
45     <span style='color:red'>{$e->getMessage()}</span>.";
```

Conexão efetuada com sucesso!

```
Array
(
    [0] => Array
        (
            [0] => 4
            [1] => bia
            [2] => 123459
        )

    [1] => Array
        (
            [0] => 1
            [1] => rafael
            [2] => 123456
        )

    [2] => Array
        (
            [0] => 5
            [1] => rebeca
            [2] => 123410
        )

    [3] => Array
        (
            [0] => 2
            [1] => renata
            [2] => 123457
        )
)
```

O usuário de id 4 é bia

PDO::FETCH_OBJ (Array com objetos na 2ª dimensão)

Basicamente estamos falando de obter o resultado fatiado em um array de objetos, ou seja, cada linha é representada por um objeto com propriedades em vez de uma linha com colunas. Observe atentamente a linha 37 e perceba que a sintaxe é bem diferente. Nesse caso, \$usuarios[0] é o primeiro objeto (do tipo StdClass) do array e ao digitar \$usuarios[0]→id ou \$usuarios[0]→login, estamos acessando os valores contidos nas propriedades id e login deste objeto.

```
32 //SELECT com fetchAll e PDO::FETCH_ASSOC, PDO::FETCH_NUM e PDO::FETCH_OBJ
33 try{
34     $comandoSql = "SELECT * FROM user order by login";
35     $stmt = $conexao->query($comandoSql);
36     $usuarios = $stmt->fetchAll(PDO::FETCH_OBJ); //Recebe um PDOStatement
37     $linha = "O usuário de id {$usuarios[0]->id} é {$usuarios[0]->login}";
38     echo "<pre>";
39     print_r($usuarios);
40     echo "</pre>";
41     echo "<br>";
42     echo $linha;
43 }catch(PDOException $e){
44     echo "Erro ao executar o comando.<br/>
45         <span style='color:red'>{$e->getMessage()}</span>.";
```

Conexão efetuada com sucesso!

```
Array
(
    [0] => stdClass Object
    (
        [id] => 4
        [login] => bia
        [senha] => 123459
    )

    [1] => stdClass Object
    (
        [id] => 1
        [login] => rafael
        [senha] => 123456
    )

    [2] => stdClass Object
    (
        [id] => 5
        [login] => rebeca
        [senha] => 123410
    )

    [3] => stdClass Object
    (
        [id] => 2
        [login] => renata
        [senha] => 123457
    )
)
```

O usuário de id 4 é bia

Exercícios de fixação.

- 1) Teste todos os códigos deste capítulo na ordem em que são apresentados.

57.PDO – query e foreach

Atenção: Antes de ler este capítulo, assista à vídeo aula “04 - PDO - QUERY e foreach”.

Neste capítulo veremos as melhores formas de percorrer o resultado de um `fetchAll()` de acordo com o formato escolhido. No entanto, para entender a melhor forma de escrever os `foreachs` (laço `for` com sintaxe “açucarada”), é importante observar o formato das informações a serem percorridas, ou seja, a estrutura dos arrays mostrada através do comando `print_r`.

Nesse primeiro exemplo temos o `print_r` da estrutura no formato `PDO::__FETCH_ASSOC`. Veja o código (comandoQuery2.php) e o resultado de sua execução logo em seguida.

```
2 require_once('conexao.php');
3 //Percorrendo os registros(linhas) retornados (__FETCH_ASSOC)
4 try{
5     $comandoSql = "SELECT * FROM user order by id";
6     $stmt = $conexao->query($comandoSql);
7     $usuarios = $stmt->fetchAll(PDO::__FETCH_ASSOC); //Recebe um PDOStatement
8     echo "=====print_r======<br/>";
9     echo "<pre>";
10    print_r($usuarios);
11    echo "</pre>";
12    echo "<hr><br>";
13 }catch(PDOException $e){
14     echo "Erro ao executar o comando.<br/>
15     <span style='color:red'>{$e->getMessage()}</span>.";
```

Conexão efetuada com sucesso!

=====print_r=====

```
Array
(
    [0] => Array
        (
            [id] => 1
            [login] => rafael
            [senha] => 123456
        )

    [1] => Array
        (
            [id] => 2
            [login] => renata
            [senha] => 123457
        )

    [2] => Array
        (
            [id] => 4
            [login] => bia
            [senha] => 123459
        )

    [3] => Array
        (
            [id] => 5
            [login] => rebeca
            [senha] => 123410
        )
)
```

Foreach simples

Na modificação a seguir, veja o exemplo de um foreach simples (linhas 14 a 18) com PDO::FETCH_ASSOC. O foreach simples percorre as linhas de \$usuarios de forma que a cada iteração estejamos tratando da próxima linha que estará sendo sempre referenciada pela variável \$usuario. Nesse caso, a cada iteração \$usuario aponta para um array com os seus valores desde que utilizemos as chaves id, login e senha, ou seja, \$usuario['id'], \$usuario['login'] e \$usuario['senha']. Veja o exemplo e o resultado da execução.

```
2 require_once('conexao.php');
3 //Percorrendo os registros(linhas) retornados (FETCH_ASSOC)
4 try{
5     $comandoSql = "SELECT * FROM user order by id";
6     $stmt = $conexao->query($comandoSql);
7     $usuarios = $stmt->fetchALL(PDO::FETCH_ASSOC); //Recebe um PDOStatement
8     /*echo "=====print_r=====";
9      echo "<pre>";
10     print_r($usuarios);
11     echo "</pre>";
12     echo "<hr><br>*/";
13     echo "=====FOREACH SIMPLES=====";
14     foreach($usuarios as $usuario){
15         echo "Id: {$usuario['id']}<br>";
16         echo "Login: {$usuario['login']}<br>";
17         echo "<hr>";
18     }
19 }catch(PDOException $e){
20     echo "Erro ao executar o comando.<br/>
21     <span style='color:red'>{$e->getMessage()}</span>.";
```

← → ⌂ ⌂ localhost/pdo/comandoQuery2.php

Conexao efetuada com sucesso!

=====FOREACH SIMPLES=====

Id: 1

Login: rafael

Id: 2

Login: renata

Id: 4

Login: bia

Id: 5

Login: rebeca

Exercícios de fixação.

- 1) Teste todos os códigos acima na ordem em que são apresentados com PDO::FETCH_ASSOC.

Foreach chave/valor dentro de um foreach simples

Na modificação a seguir, veja o exemplo de um foreach simples (linhas 20 a 25) com PDO::FETCH_ASSOC. O foreach simples percorre as linhas de \$usuarios de forma que a cada iteração estejamos tratando da próxima linha onde as chaves(id, login

e senha) estarão sendo referenciadas pela variável \$usuário. Cada \$usuário aponta para um array com 3 posições para id, login e senha.

Como a cada iteração \$usuário estará apontando para um outro array com as chaves id, login e senha e seus respectivos valores, no for mais interno (linhas 21 a 23), podemos separar, a cada iteração, as informações de \$usuário em \$chave e \$valor.

Nesse caso, como \$usuário possui 3 chaves e 3 valores, teremos 3 iterações para extrair a chave id e o seu valor, depois a chave login e o seu valor e por fim a chave senha e o seu valor. Veja o exemplo e o resultado da execução.

```
2 require_once('conexao.php');
3 //Percorrendo os registros(linhas) retornados (FETCH_ASSOC)
4 try{
5     $comandoSql = "SELECT * FROM user order by id";
6     $stmt = $conexao->query($comandoSql);
7     $usuarios = $stmt->fetchAll(PDO::FETCH_ASSOC); //Recebe um PDOStatement
8     /*echo "=====print_r=====";
9      echo "<pre>";
10     print_r($usuarios);
11     echo "</pre>";
12     echo "<br><br>*/;
13     /*echo "=====FOREACH SIMPLES=====";
14     foreach($usuarios as $usuario){
15         echo "Id: {$usuario['id']}<br>";
16         echo "Login: {$usuario['login']}<br>";
17         echo "<br>";
18     }*/
19     echo "=====FOREACH CHAVE VALOR=====";
20     foreach ($usuarios as $usuario) {
21         foreach ($usuario as $chave => $valor) {
22             echo "{$chave}: {$valor}<br>";
23         }
24         echo "<br>";
25     }
26 }catch(PDOException $e){
27     echo "Erro ao executar o comando.<br/>
28     <span style='color:red'>{$e->getMessage()}</span>.";
```

← → ⌂ ⌂ localhost/pdo/comandoQuery2.php

Conexão efetuada com sucesso!

=====FOREACH CHAVE VALOR=====

id: 1
login: rafael
senha: 123456

id: 2
login: renata
senha: 123457

id: 4
login: bia
senha: 123459

id: 5
login: rebeca
senha: 123410

Exercícios de fixação.

- 2) Teste todos os códigos acima na ordem em que são apresentados com PDO::FETCH_ASSOC.

Foreach simples e foreach chave/valor dentro de um foreach simples com PDO::FETCH_OBJ

Na modificação a seguir, veja o exemplo de um foreach simples (linhas 42 a 46) com PDO::FETCH_OBJ. O foreach simples percorre as linhas de \$usuarios de forma que a cada iteração estejamos tratando da próxima linha onde temos um objeto \$usuario com os atributos id, login e senha.

Nas linhas 48 a 54 percorremos as duas dimensões de \$usuarios, dessa vez tratando com as diferenças de sintaxe entre o retorno de PDO::FETCH_ASSOC (abordado no exercício anterior) e PDO::FETCH_OBJ. O raciocínio é o mesmo. Tente entender e em caso de dúvidas, entre em contato com o professor (Eu! 😊).

```
31 //Percorrendo os registros retornados (FETCH_OBJ)
32 try{
33     $comandoSql = "SELECT * FROM user order by id";
34     $stmt = $conexao->query($comandoSql);
35     $usuarios = $stmt->fetchAll(PDO::FETCH_OBJ); //Recebe um PDOStatement
36     echo "=====print_r=====";
37     echo "<pre>";
38     print_r($usuarios);
39     echo "</pre>";
40     echo "<hr><br>";
41     echo "=====FOREACH SIMPLES=====";
42     foreach($usuarios as $usuario){
43         echo "Id: {$usuario->id}<br>";
44         echo "Login: {$usuario->login}<br>";
45         echo "<hr>";
46     }
47     echo "=====FOREACH CHAVE VALOR=====";
48     foreach ($usuarios as $usuario) {
49         foreach ($usuario as $chave => $valor) {
50             $valor = "{$usuario->$valor}";
51             echo "{$chave}: {$usuario->$chave}<br>";
52         }
53     }
54 }
55 }catch(PDOException $e){
56     echo "Erro ao executar o comando.<br/>
57     <span style='color:red'>{$e->getMessage()}</span>.";
```

Conexão efetuada com sucesso!

=====print_r=====

```
Array
(
    [0] => stdClass Object
        (
            [id] => 1
            [login] => rafael
            [senha] => 123456
        )

    [1] => stdClass Object
        (
            [id] => 2
            [login] => renata
            [senha] => 123457
        )

    [2] => stdClass Object
        (
            [id] => 4
            [login] => bia
            [senha] => 123459
        )

    [3] => stdClass Object
        (
            [id] => 5
            [login] => rebeca
            [senha] => 123410
        )
)
```

=====FOREACH SIMPLES=====

Id: 1
Login: rafael

Id: 2
Login: renata

Id: 4
Login: bia

Id: 5
Login: rebeca

=====FOREACH CHAVE VALOR=====

id: 1
login: rafael
senha: 123456

id: 2
login: renata
senha: 123457

id: 4
login: bia
senha: 123459

id: 5
login: rebeca
senha: 123410

Exercícios de fixação.

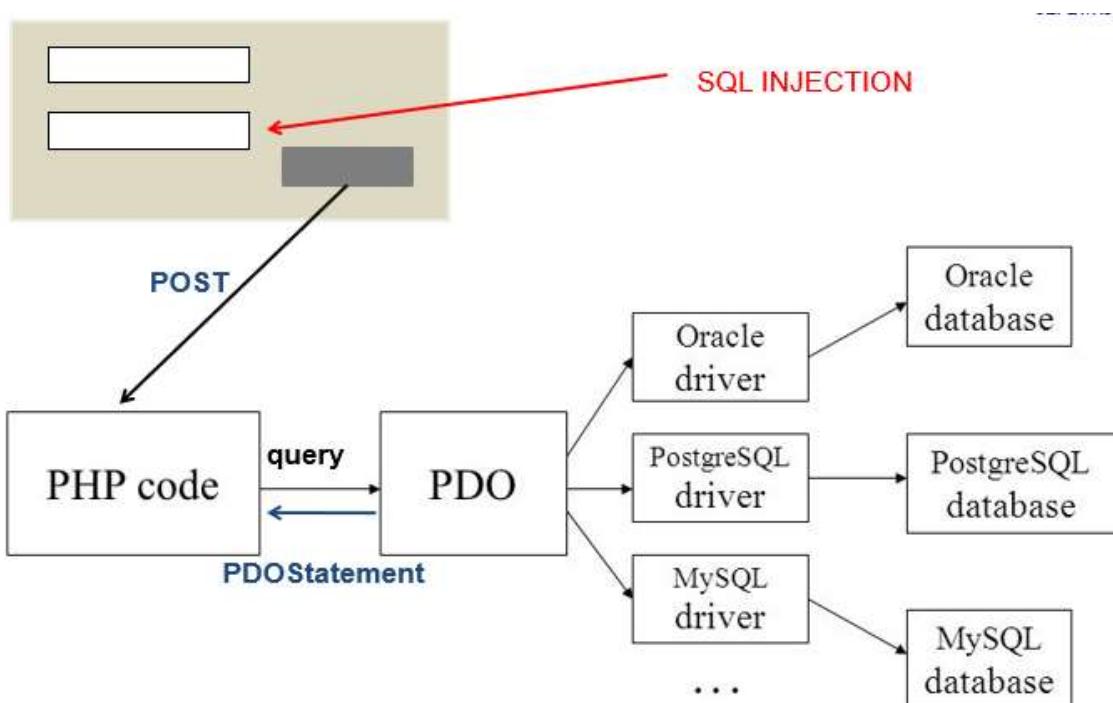
- 1) Teste todos os códigos acima na ordem em que são apresentados com PDO::FETCH_OBJ.
- 2) Faça os mesmos testes substituindo PDO::FETCH_OBJ por PDO::FETCH_NUM.

58.PDO – SQL INJECTION - PROBLEMA

Atenção: Antes de ler este capítulo, assista à vídeo aula “05 - PDO - SQL INJECTION - PROBLEMA”.

Quando falamos de operações que envolvem acesso a banco de dados, uma das maiores preocupações é com a questão da segurança. Caso os códigos não sejam muito bem escritos, corre-se o risco de expor seus dados a uma prática maldosa conhecida como SQL INJECTION.

Nesse capítulo mostraremos a falha que da abertura para que esse tipo de prática aconteça e no capítulo seguinte, corrigimos o problema. Como pode ser observado na imagem abaixo, sempre que recebemos informações de um formulário para integrar o nosso comando SQL a ser executado no banco de dados, corremos o risco de expor nosso código a essa injeção de código maldoso conhecido como SQL INJECTION.



Como acontece o SQL INJECTION?

Geralmente permitimos que o usuário digite um código nos inputs de nosso formulário de modo a executar um comando qualquer diferente daquele que esperamos receber.

Que tipo de comando pode ser esse?

Basicamente qualquer tipo de comando. Vou exemplificar alguns que podem fazer um estrago muito grande. São eles:

- Deletar um ou mais registros de uma tabela;
- Alterar as informações de um ou mais registros de uma tabela;
- Alterar a estrutura de uma tabela ou até mesmo excluir uma tabela com dados e tudo o mais.

Esses são apenas alguns exemplos de códigos maldosos que podem ser injetados em nossos formulários.

No exemplo abaixo criamos um formulário para validar um usuário. Para isso utilizaremos a mesma tabela user do capítulo anterior. Na verdade, escrevemos 2 arquivos a saber:

- sql-injection.html que contém um formulário para o usuário digitar login e senha e enviar para um script php no lado servidor chamado validaUser.php. Repare que tivemos o cuidado de definir o campo senha como sendo do tipo password. Assim os caracteres que estão sendo digitados não aparecem.
- validaUser.php recebe os dados e realiza, teoricamente, uma operação no banco de dados para saber se o login e senha enviados existem em uma das linhas da tabela user.

Veja os dois códigos abaixo:

sql-injection.html

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="utf-8">
5      <title>SQL INJECTION</title>
6  </head>
7  <body>
8      <h2>Exemplo de SQL INJECTION</h2>
9      <hr>
10     <p>Entre com as credenciais p/ verificar se o usuário existe.</p>
11    <main>
12        <form action="validaUser.php" method="POST">
13            <label for="login">Login: </label>
14            <input type="text" name="login" id="login"><br>
15            <label for="senha">Senha: </label>
16            <input type="password" name="senha" id="senha"><br><br>
17            <input type="submit" value="Validar">
18            <input type="reset" value="Limpar">
19        </form>
20    </main>
21  </body>
22  </html>
```

validaUser.php

```
2
3  require_once('conexao.php');
4  if(!empty($_POST['login']) && !empty($_POST['senha'])){
5      //Faço algo
6      $comandoSql = "SELECT * FROM user WHERE login = '{$_POST['login']}'"
7      AND senha = '{$_POST['senha']}' ";
8      echo "<hr>{$comandoSql}<hr>";
9      $stmt = $conexao->query($comandoSql);
10     $usuario = $stmt->fetch(PDO::FETCH_ASSOC); //Recebe um PDOStatement
11     echo "<pre>";
12     print_r($usuario);
13     echo "</pre>";
14     echo "<hr>";
15 }else{
16     echo "<span style='color:red'>As informações não chegaram ao servidor.
17     Impossível validar!</span>";
18 }
```

Veja um exemplo de como o usuário pode alterar o propósito do formulário via navegador. Observe atentamente a imagem abaixo. Ao lado esquerdo vemos o navegador e ao lado direito vemos a ferramenta DevTools do mesmo. Nessa ferramenta é possível alterar o tipo de input, a quantidade de caracteres que ele aceira e daí por diante. Perceba que nesse caso o usuário foi na aba elements e alterou type do segundo input para “text”.

Onde esperava-se apenas uma senha, o usuário digitou o seguinte:

123456';UPDATE user SET login='ERRO' WHERE id=2;

Percebam que logo após o caractere 6, o usuário encerrou um comando com ‘ e ; e logo em seguida iniciou um outro comando que altera o login do usuário de id = 2.

Obviamente, o código poderia fazer alterações muito mais significativas. No entanto, o objetivo aqui é mostrar que um código extra pode ser executado. Após clicar em validar, o resultado é o seguinte:

Perceba que na aba network, selecionamos o arquivo validaUser.php para verificar qual informação foi enviada. Na aba payload podemos ver que enviamos login: rafael e senha: 123456';UPDATE user SET login='ERRO' WHERE id=2;

O SELECT foi executado normalmente, tanto que imprimiu a linha encontrada por meio de um print_r. O problema é que o UPDATE também foi executado e o usuário de id 2 que tinha o login ‘renata’ passou a ter o login ‘ERRO’. Veja o resultado no banco de dados.

The screenshot shows the PhpMyAdmin interface. At the top, a green bar displays the message: "A mostrar registos de 0 - 3 (4 total, A consulta demorou 0,0010 segundos.)". Below this, a query is shown: "SELECT * FROM `user`". Under the results section, there are filtering options: "Mostrar tudo" (Show all), "Número de registos: 25", and a search field "Filtrar registos: Pesc". A table titled "user" is displayed with columns: id, login, senha. The data is as follows:

	id	login	senha
<input type="checkbox"/>	1	rafael	123456
<input type="checkbox"/>	2	ERRO	123457
<input type="checkbox"/>	4	bia	123459
<input type="checkbox"/>	5	rebeca	123410

Por que isso aconteceu?

Veja o código contido em validaUser.php novamente. Nas linhas 5 e 6 você vai perceber que nosso estamos montando nosso código SQL com informações vindas do formulário (`$_POST['login']` e `$_POST['senha']`). Isso acontece porque o comando `query` não está recebendo parâmetros como é o correto. Ele monta o código sql com informações externas e o executa.

Da mesma forma que podemos executar vários comandos separados por ; no console do PhpMyAdmin, podemos fazer utilizando PDO ou qualquer outra biblioteca/ classe que promova acesso a banco de dados.

Como resolver o problema?

No capítulo seguinte mostraremos como utilizar uma Instrução preparada contendo a promessa de esperar por parâmetros. Assim, o comando só será executado se fornecermos os parâmetros fornecidos. O mais importante e que nos traz mais segurança é que dessa nova forma, o comando SQL não pode ser alterado por valores externos. Veremos no próximo capítulo!

Exercícios de fixação.

- 1) Teste todos os códigos acima na ordem em que são apresentados e utilizando a senha correta.

Antes de ir para o próximo capítulo, vocês podem estar se perguntando:
E se eu tivesse digitado login ou senha incorretos?

O resultado seria o mesmo. A única diferença é que o primeiro comando não retornaria nenhuma linha. O segundo comando (o comando maldoso) seria executado da mesma forma. Veja e teste a seguir!

SENHA ERRADA!!

123321';UPDATE user SET login='renata' WHERE id=2;

The screenshot illustrates a SQL injection attack on a web application. At the top, a browser window shows a form with fields for 'Login' (rafael) and 'Senha' (123321'; UPDATE user SET). Below the form, the page title is 'Exemplo de SQL INJECTION'. A message says 'Entre com as credenciais p/ verificar se o usuário existe.' (Enter credentials to verify if the user exists). Buttons for 'Validar' and 'Limpar' are present. To the right, the browser's DevTools Network tab shows the raw POST request sent to 'validaUser.php': 'login: rafael&senha: 123321'; UPDATE user SET login='renata' WHERE id=2;'. The response from 'validaUser.php' indicates success: 'Conexão efetuada com sucesso!' and displays the injected SQL query. The bottom part of the screenshot shows a table listing user records with columns: id, login, senha. The table includes rows for users with IDs 1 through 5, each with edit, copy, and delete options.

	id	login	senha
<input type="checkbox"/>	1	rafael	123456
<input type="checkbox"/>	2	renata	123457
<input type="checkbox"/>	4	bia	123459
<input type="checkbox"/>	5	rebeca	123410

Exercícios de fixação.

- 1) Teste todos os códigos acima na ordem em que são apresentados e utilizando a senha INCORRETA ou até mesmo o login INCORRETO.

59.PDO – Evitando SQL INJECTION – PREPARE e EXECUTE

Atenção: Antes de ler este capítulo, assista à vídeo aula “06 - PDO - EVITANDO SQL INJECTION - PREPARE e EXECUTE”.

Como já foi adiantado no capítulo anterior, a resolução do problema passa por um código mais rígido no lado servidor. Portanto, a única alteração no arquivo sql-injection.html está no action do form que agora envia para validaUser2.php.

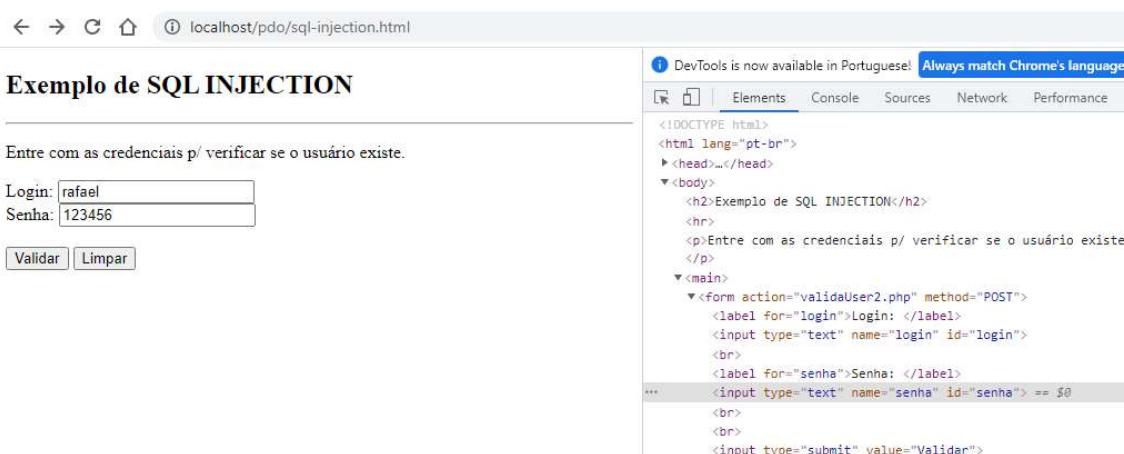


```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <meta charset="utf-8">
5   <title>SQL INJECTION</title>
6 </head>
7 <body>
8   <h2>Exemplo de SQL INJECTION</h2>
9   <hr>
10  <p>Entre com as credenciais p/ verificar se o usuário existe.</p>
11 <main>
12  <form action="validaUser2.php" method="POST"> ← Red arrow
13    <label for="login">Login: </label>
14    <input type="text" name="login" id="login"><br>
15    <label for="senha">Senha: </label>
16    <input type="password" name="senha" id="senha"><br><br>
17    <input type="submit" value="Validar">
18    <input type="reset" value="Limpar">
19  </form>
20 </main>
21 </body>
22 </html>
```

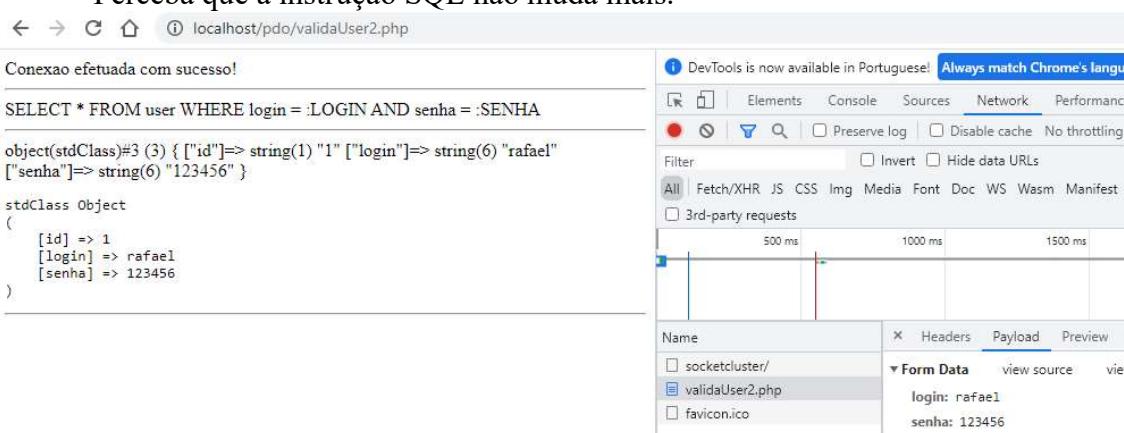
Em validaUser2.php, as mudanças estão no \$comandoSql que agora possui parâmetros (linhas 6 e 7), na preparação da instrução através do método prepare (linha 9), na passagem dos parâmetros (linhas 10 e 11) e no execute() (linha 13). Essa é a sintaxe para definir uma instrução preparada (PreparedStatement), passar os parâmetros e só depois executar.

O execute() só funciona se antes forem passados os parâmetros prometidos. Observe atentamente o código abaixo com a execução normal e posteriormente com a tentativa mal sucedida de SQL INJECTION. Não deixe de fazer os exercícios propostos testando todos os códigos do capítulo!

```
2 //Usando PREPARE e EXECUTE
3 require_once('conexao.php');
4 if(!empty($_POST['login']) && !empty($_POST['senha'])){
5     //Faço algo
6     $comandoSql = "SELECT * FROM user WHERE login = :LOGIN
7     AND senha = :SENHA ";
8     echo "<hr>{$comandoSql}<hr>";
9     $stmt = $conexao->prepare($comandoSql);
10    $stmt->bindValue(':LOGIN', $_POST['login']);
11    $stmt->bindValue(':SENHA', $_POST['senha']);
12    try{
13        $stmt->execute(); ←
14        $usuario = $stmt->fetch(PDO::FETCH_OBJ); //Recebe um PDOStatement
15        var_dump($usuario);
16        echo "<pre>";
17        print_r($usuario);
18        echo "</pre>";
19        echo "<hr>";
20    }catch(PDOException $e){
21        echo "<span style='color:red'>Erro ao realizar a consulta.
22        {$e->getCode()} - {$e->getMessage()}</span>";
23        exit();
24    }
25 }else{
26     echo "<span style='color:red'>As informações não chegaram ao servidor.
27     Impossível validar!</span>";
28     exit();
29 }
```



Perceba que a instrução SOL não muda mais.



Exercícios de fixação.

- 1) Teste todos os códigos acima, QUE NÃO TEM TENTATIVA DE SQL INJECTION, na ordem em que são apresentados e verifique se o resultado exibido é o esperado.

Tentativa de SQL INJECTION. Veja o código abaixo e depois teste para ver se realmente estamos protegidos.

The screenshot shows a web page titled "Exemplo de SQL INJECTION". The page contains a form with fields for "Login" (containing "rafael") and "Senha" (containing "123456';UPDATE user SET"). Below the form are two buttons: "Validar" and "Limpar". The page content includes a message: "Entre com as credenciais p/ verificar se o usuário existe." To the right, the Chrome DevTools Network tab is open, showing a request to "localhost/validaUser2.php". The "Payload" section of the Network tab shows the raw POST data: "login: rafael&senha: 123456';UPDATE user SET login='ERRO' WHERE id=2;".

Exercícios de fixação.

- 1) Teste todos os códigos acima na ordem em que são apresentados e verifique se o resultado exibido é o esperado.

Usando o PDO::PARAM

```

2 //Usando PREPARE e EXECUTE
3 require_once('conexao.php');
4 if(!empty($_POST['login']) && !empty($_POST['senha'])){
5     //Faço algo
6     $comandoSql = "SELECT * FROM user WHERE login = :LOGIN
7     AND senha = :SENHA ";
8     echo "<hr>{$comandoSql}<hr>";
9     $stmt = $conexao->prepare($comandoSql);
10    $stmt->bindValue(':LOGIN', $_POST['login'], PDO::PARAM_STR);
11    $stmt->bindValue(':SENHA', $_POST['senha'], PDO::PARAM_INT);
12    try{
13        $stmt->execute();
14        $usuario = $stmt->fetch(PDO::FETCH_OBJ); //Recebe um PDOStatement
15        var_dump($usuario);
16        echo "<pre>";
17        print_r($usuario);
18        echo "</pre>";
19        echo "<hr>";
20    }catch(PDOException $e){
21        echo "<span style='color:red'>Erro ao realizar a consulta.
22        {$e->getCode()} - {$e->getMessage()}</span>";
23        exit();
24    }
25 }else{
26     echo "<span style='color:red'>As informações não chegaram ao servidor.
27     Impossível validar!</span>";
28     exit();
29 }

```

The screenshot shows a web browser window for 'localhost/pdo/sql-injection.html' with the title 'Exemplo de SQL INJECTION'. The page contains a form with fields for 'Login' (rafael) and 'Senha' (123456';UPDATE user SET). Below the form are 'Validar' and 'Limpar' buttons. To the right is the Chrome DevTools Network tab, which displays the raw HTML code of the page and the network traffic for the request to 'validaUser2.php'. The Network tab shows a POST request with the payload 'login: rafael&senha: 123456';UPDATE user SET login='ERRO' WHERE id=2;.

A parte inteira do início do comando foi transformada em int e o restante foi descartado, ou seja, '123456' foi transformado em 123456 e o comando foi aceito.

Leia mais sobre as constantes PDO::PARAM em
https://www.php.net/manual/pt_BR/pdo.constants.php

Exercícios de fixação.

- 1) Teste todos os códigos acima utilizando o PDO::PARAM, na ordem em que são apresentados, e verifique se o resultado exibido é o esperado.

60.CRUD via PDO (HTML, CSS e PHP)

Servidor: 127.0.0.1

Base de Dados SQL Estado Contas de utilizador Exportar

Executar consulta(s) SQL no servidor “127.0.0.1”:

```

1 DROP DATABASE IF EXISTS cinedesweb;
2 CREATE DATABASE cinedesweb CHARSET=utf8 COLLATE=utf8_unicode_ci;
3 USE cinedesweb;
4 CREATE TABLE filmes_assistidos(id INT AUTO_INCREMENT PRIMARY KEY,
5                                 titulo VARCHAR(80) NOT NULL,
6                                 avaliacao DECIMAL(9,1) NOT NULL DEFAULT 0.0,
7                                 UNIQUE INDEX idx_filme_titulo(titulo)
8 )ENGINE=INNODB;
9 INSERT INTO filmes_assistidos VALUES(1,'INTERESTELAR',9.8),
10 (2,'AMOR A TODA PROVA',9.7), (3,'SIMPLEMENTE AMOR', 9.5),
11 (4,'GLADIADOR',8.9),(5,'DJANGO LIVRE',9.3),
12 (6,'ENTRE FACAS E SEGREDOS',9.5), (7,'MATRIX', 5.5),
13 (8,'GRAVIDADE',6.9),(9,'PULP FICTION',9.4),
14 (10,'AS BRANQUELAS',5.4), (11,'EFEITO BORBOLETA',9.5);

```

A mostrar registos de 0 - 10 (11 total, A consulta demorou 0.0125 segundos.)

```
SELECT * FROM `filmes_assistidos`
```

Mostrar tudo Número de registos: 25 Filtrar registos: Pesquisar est

+ Opções

	id	titulo	avaliacao
<input type="checkbox"/>	1	INTERESTELAR	9.8
<input type="checkbox"/>	2	AMOR A TODA PROVA	9.7
<input type="checkbox"/>	3	SIMPLEMENTE AMOR	9.5
<input type="checkbox"/>	4	GLADIADOR	8.9
<input type="checkbox"/>	5	DJANGO LIVRE	9.3
<input type="checkbox"/>	6	ENTRE FACAS E SEGREDOS	9.5
<input type="checkbox"/>	7	MATRIX	5.5
<input type="checkbox"/>	8	GRAVIDADE	6.9
<input type="checkbox"/>	9	PULP FICTION	9.4
<input type="checkbox"/>	10	AS BRANQUELAS	5.4
<input type="checkbox"/>	11	EFEITO BORBOLETA	9.5

x +

localhost/crud-php/ Home Inserir novo filme

Lista de Filmes assistidos

ID	TÍTULO	NOTA	AÇÕES
1	INTERESTELAR	9.8	[Alterar] [Excluir]
2	AMOR A TODA PROVA	9.7	[Alterar] [Excluir]
3	SIMPLEMENTE AMOR	9.5	[Alterar] [Excluir]
4	GLADIADOR	8.9	[Alterar] [Excluir]
5	DJANGO LIVRE	9.3	[Alterar] [Excluir]
6	ENTRE FACAS E SEGREDOS	9.5	[Alterar] [Excluir]
7	MATRIX	5.5	[Alterar] [Excluir]
8	GRAVIDADE	6.9	[Alterar] [Excluir]
9	PULP FICTION	9.4	[Alterar] [Excluir]
10	AS BRANQUELAS	5.4	[Alterar] [Excluir]
11	EFEITO BORBOLETA	9.5	[Alterar] [Excluir]

Todos os direitos reservados

Título: Avaliação: 8.7 [Enviar] [Limpar]

Todos os direitos reservados

Id: 7 Título: MATRIX REVOLUTIONS Avaliação: 5.5 [Enviar] [Limpar]

Todos os direitos reservados

Filme de id 7 atualizado com sucesso!

ID	TÍTULO	NOTA	ACÕES
1	INTERESTELAR	9.8	[Alterar] [Excluir]
2	AMOR A TODA PROVA	9.7	[Alterar] [Excluir]
3	SIMPLESMENTE AMOR	9.5	[Alterar] [Excluir]
4	GLADIADOR	8.9	[Alterar] [Excluir]
5	DJANGO LIVRE	9.3	[Alterar] [Excluir]
6	ENTRE FACAS E SEGREDOS	9.5	[Alterar] [Excluir]
7	MATRIX REVOLUTIONS	5.5	[Alterar] [Excluir]
8	GRAVIDADE	6.9	[Alterar] [Excluir]
9	PULP FICTION	9.4	[Alterar] [Excluir]
10	AS BRANQUELAS	5.4	[Alterar] [Excluir]
11	EFEITO BORBOLETA	9.5	[Alterar] [Excluir]

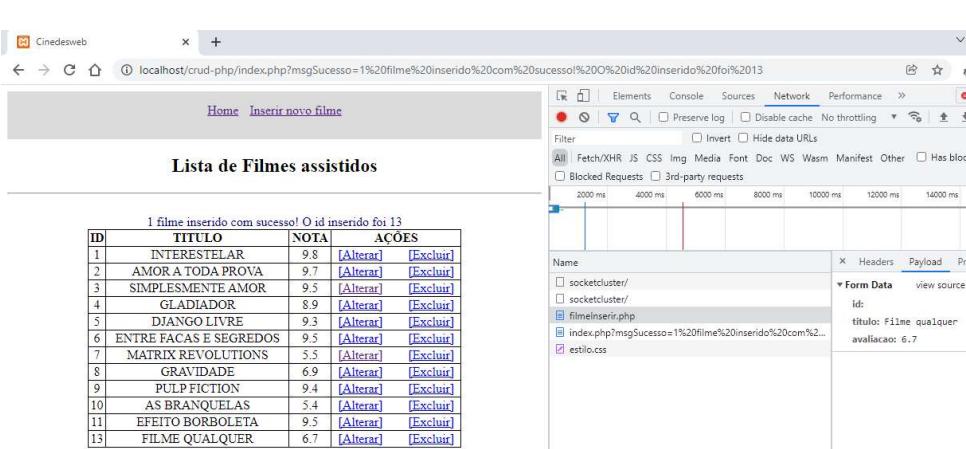
Todos os direitos reservados

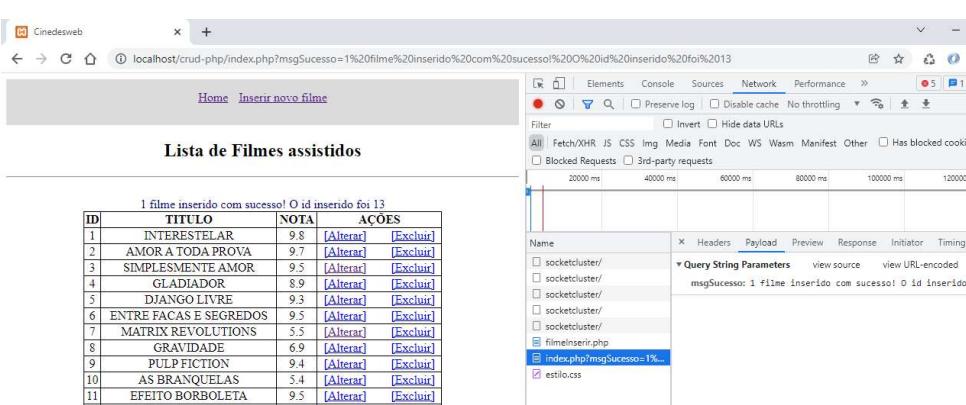
Título: Interestelar Avaliação: 9.8 [Enviar] [Limpar]

Todos os direitos reservados









ID	TÍTULO	NOTA	AÇÕES
1	INTERESTELAR	9.8	[Alterar] [Excluir]
2	AMOR A TODA PROVA	9.7	[Alterar] [Excluir]
3	SIMPLESMENTE AMOR	9.5	[Alterar] [Excluir]
4	GLADIADOR	8.9	[Alterar] [Excluir]
5	DJANGO LIVRE	9.3	[Alterar] [Excluir]
6	ENTRE FACAS E SEGREDOS	9.5	[Alterar] [Excluir]
7	MATRIX REVOLUTIONS	5.5	[Alterar] [Excluir]
8	GRAVIDADE	6.9	[Alterar] [Excluir]
9	PULP FICTION	9.4	[Alterar] [Excluir]
10	AS BRANQUELAS	5.4	[Alterar] [Excluir]
11	EFEITO BORBOLETA	9.5	[Alterar] [Excluir]

ID	TÍTULO	NOTA	AÇÕES
1	INTERESTELAR	9.8	[Alterar] [Excluir]
2	AMOR A TODA PROVA	9.7	[Alterar] [Excluir]
3	SIMPLESMENTE AMOR	9.5	[Alterar] [Excluir]
4	GLADIADOR	8.9	[Alterar] [Excluir]
5	DJANGO LIVRE	9.3	[Alterar] [Excluir]
6	ENTRE FACAS E SEGREDOS	9.5	[Alterar] [Excluir]
7	MATRIX REVOLUTIONS	5.5	[Alterar] [Excluir]
8	GRAVIDADE	6.9	[Alterar] [Excluir]
9	PULP FICTION	9.4	[Alterar] [Excluir]
10	AS BRANQUELAS	5.4	[Alterar] [Excluir]
11	EFEITO BORBOLETA	9.5	[Alterar] [Excluir]

Como

```

File Edit Selection View ...
EXPLORER ...
CRUD-PHP
conexao.php
# estilico.css
filmeAtualizar.php
filmeBuscar.php
filmeExcluir.php
filmeFormInserir.php
filmeInserir.php
index.php
resto.php
script.sql

conexao.php > ...
1  ?php
2   *Para obter uma conexão via PDO, você precisa fornecer 3 informações:
3   **Data source Name --> representa uma string de conexão onde você define
4   o SGBD(mysql), um host(localhost) e uma base de dados(cinedesweb);
5   A sintaxe é "sgbd:host=nomeOuIpDoHost;dbname=nomeDaBaseDeDados"
6
7   **Usuário do SGBD('root');
8   **Senha do SGBD('') No nosso caso estamos usando o usuário root sem senha.
9 */
10 $dsn = "mysql:host=localhost;dbname=cinedesweb";
11 $user = "root";
12 $pass = "";
13 try{
14   $conexao = new PDO($dsn,$user,$pass); //Linha de interrupção
15   //echo "Conexao efetuada com sucesso!<br>";
16
17 }catch(PDOException $e){
18   //Relançando a exceção com a própria mensagem de erro
19   throw new PDOException("Codigo do erro: {$e->getCode()} - Mensagem de erro: {$e->getMessage()}");
20 }
21 ?|

```

```
 1 index.php > html > head > title
 2   <!DOCTYPE html>
 3   <html lang="pt-br">
 4     <head>
 5       <meta charset="UTF-8">
 6       <meta http-equiv="X-UA-Compatible" content="IE=edge">
 7       <meta name="viewport" content="width=device-width, initial-scale=1.0">
 8       <title>Cinedesweb</title>
 9       <link rel="stylesheet" href="estilo.css" />
10       <style>
11     </style>
12   </head>
13   <body>
14     <nav>
15       <ul class="menu">
16         <li><a href="index.php">Home</a></li>
17         <li><a href="filmeFormInserir.php">Inserir novo filme</a></li>
18       </ul>
19     </nav>
20     <main>
21       <h2 class="titulo">Lista de Filmes assistidos</h2>
22       <hr/>
23       <?php
24         $msgErro = (isset($_GET['msgErro']) && $_GET['msgErro'] != null) ? $_GET['msgErro'] : null;
25         $msgSucesso = (isset($_GET['msgSucesso']) && $_GET['msgSucesso'] != null) ? $_GET['msgSucesso'] : null;
26         $inicioTabela = null; $meioTabela = null; $fimTabela = null;
27         try {
28           require_once('conexao.php');
29           $sql = "SELECT id,titulo,avaliacao FROM filmes_assistidos";
30           $stmt = $conexao->prepare($sql);
31           if ($stmt->execute()) {
32             $rs = $stmt->fetch(PDO::FETCH_ASSOC);
33             $inicioTabela =
34               "<thead><tr><th>ID</th><th>TITULO</th><th>NOTA</th><th>AÇÕES</th></tr>".
35               "</thead><tbody>";
36             $meioTabela .=
37               "<tr><td>{$rs['id']}</td><td>{$rs['titulo']}</td><td>{$rs['avaliacao']}</td>".
38               "<td><center>".
39                 "<a href='filmeBuscar.php?id={$rs['id']}'>[Alterar]</a>".
40                 "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;".
41                 "<a href='filmeExcluir.php?id={$rs['id']}'>[Excluir]</a>".
42               "</center></td></tr>";
43             $fimTabela = "</tbody>";
44           }
45         } else {
46           $msgErro = "Erro: Não foi possível recuperar os dados do banco de dados";
47         }
48       } catch (PDOException $e) {
49         $msgErro = "Erro de sql: ". $e->getMessage();
50       }
51     ?>
52     <div id="erro">
53       <span id="msgErro">
54         <?php echo (isset($msgErro) && $msgErro != null) ? $msgErro : "" ; ?>
55       </span>
56       <span id="msgSucesso">
57         <?php echo (isset($msgSucesso) && $msgSucesso != null) ? $msgSucesso : "" ; ?>
58       </span>
59     </div>
60     <table id="tblFilmes">
61       <?php
62         if( $inicioTabela != null && $meioTabela != null && $fimTabela != null){
63           echo $inicioTabela;
64           echo $meioTabela;
65           echo $fimTabela;
66         }
67       ?>
68     </table>
69     <br/>
70   </main>
71   <footer>
72     <hr/>
73     Todos os direitos reservados
74   </footer>
75 </body>
76 </html>
```

```
# estilo.css > table
1  nav{
2      background-color: #DBDBDB;
3      height: 60px;
4  }
5  ul.menu{
6      margin-top: 0;
7      padding: 15px 15px;
8      text-align: center; /* centraliza na horizontal */
9  }
10 ul.menu li {
11     display: inline-block; /* centraliza na horizontal */
12     margin: 0 5px;
13 }
14 .titulo{
15     text-align: center; /* centraliza na horizontal */
16 }
17 table, th, td {
18     border: 1px solid black;
19     width: 70%;
20     margin-top: 10px;
21     border-collapse: collapse;
22     text-align: center;
23 }
24 #tblFilmes {
25     width: 70%;
26     margin: 0px auto; /* centraliza na horizontal */
27 }
28 th, td{
29     width: auto;
30 }
31 #erro{
32     text-align: center;
33 }
34 #msgErro{
35     color: red;
36 }
37 #msgSucesso{
38     color: navy;
39 }
40 footer{
41     text-align: center;
42     font-size: smaller;
43 }
```

```
filmeFormInserir.php > html > body > main > section > form > input
1  <!DOCTYPE html><html lang="pt-br">
2  <head>
3      <meta charset="UTF-8">
4      <meta http-equiv="X-UA-Compatible" content="IE=edge">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Cinedesweb</title>
7      <link rel="stylesheet" href="estilo.css" />
8  </head>
9  <body>
10     <nav>
11         <ul class="menu">
12             <li><a href="index.php">Home</a></li>
13         </ul>
14     </nav>
15     <main>
16         <h2 class="titulo">Inserir novo filme assistido</h2><hr/>
17         <section><form action="filmeInserir.php" method="POST">
18             <input type="hidden" name="id" />
19             Título: <input type="text" name="titulo" />
20             Avaliação: <input type="number" min="0" max="10" step="0.1" name="avaliacao" />
21             <input type="submit" value="Enviar" />
22             <input type="reset" value="Limpar" />
23         </form></section><br/>
24     </main>
25     <footer>
26         <hr/>
27         Todos os direitos reservados
28     </footer>
29 </body></html>
```

```
file inserir.php > ...
1  <?php
2  require_once('conexao.php');
3  $titulo = (isset($_POST["titulo"]) && $_POST["titulo"] != null) ? strtoupper($_POST["titulo"]) : "";
4  $avaliacao = (isset($_POST["avaliacao"]) && $_POST["avaliacao"] != null) ? $_POST["avaliacao"] : "";
5  $msgErro = null;
6  $msgSucesso = null;
7  if($titulo != "" && $avaliacao != ""){
8      try {
9          $sql = "INSERT INTO filmes_assistidos(titulo,avaliacao) VALUES(?,?)";
10         $stmt = $conexao->prepare($sql);
11         $stmt->bindParam(1, $titulo );
12         $stmt->bindParam(2, $avaliacao);
13
14         if ($stmt->execute()) {
15             $msgSucesso = "{$stmt->rowCount()} filme inserido com sucesso! O id inserido foi {$conexao->lastInsertId()}";
16         } else {
17             $msgErro = "Erro: Não foi possível efetuar a inserção no BD";
18         }
19     }catch(PDOException $e) {
20         $msgErro = "Erro: ".$e->getMessage();
21     }finally{
22         if($msgErro != null)
23             header("Location: http://localhost/crud-php/index.php?msgErro=$msgErro");
24         else
25             header("Location: http://localhost/crud-php/index.php?msgSucesso=$msgSucesso");
26         die();
27     }
28 }
29 ?>
```

```
filmeBuscar.php > html > body
1  <!DOCTYPE html><html lang="pt-br">
2  <head>
3      <meta charset="UTF-8">
4      <meta http-equiv="X-UA-Compatible" content="IE=edge">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Cinedesweb</title>
7      <link rel="stylesheet" href="estilo.css" />
8  </head>
9  <body>
10     <nav>
11         <ul class="menu">
12             <li><a href="index.php">Home</a></li>
13             <li><a href="filmeFormInserir.php">Inserir novo filme</a></li>
14         </ul>
15     </nav>
16 
17     <?php
18     $msgErro = (isset($_GET['msgErro']) && $_GET['msgErro'] != null) ? $_GET['msgErro'] : null;
19     $msgSucesso = (isset($_GET['msgSucesso']) && $_GET['msgSucesso'] != null) ? $_GET['msgSucesso'] : null;
20     $id = (isset($_GET['id']) && $_GET['id'] != null) ? $_GET['id'] : null;
21     $titulo = null;
22     $avaliacao = null;
23     if ($id != null){
24         try {
25             require_once('conexao.php');
26             $sql = "SELECT id,titulo,avaliacao FROM filmes_assistidos WHERE id=?";
27             $stmt = $conexao->prepare($sql);
28             $stmt->bindParam(1, $id);
29             if ($stmt->execute()) {
30                 while ($rs = $stmt->fetch(PDO::FETCH_ASSOC)) {
31                     $titulo = strtoupper($rs['titulo']);
32                     $avaliacao = $rs['avaliacao'];
33                 }
34             } else {
35                 $msgErro = "Erro: Não foi possível recuperar os dados do banco de dados";
36             }
37         }catch(PDOException $e) {
38             $msgErro = "Erro de sql: ".$e->getMessage();
39         }
40     }else{
41         $msgErro = "ERRO: id inválido";
42     }
43     ?>
44 
45     <main>
46         <h2 class="titulo">Alterar filme assistido</h2><hr/>
47         <div id="erro">
48             <span id="msgErro">
49                 <?php echo (isset($msgErro) && $msgErro != null) ? $msgErro : "" ; ?>
50             </span>
51             <span id="msgSucesso">
52                 <?php echo (isset($msgSucesso) && $msgSucesso != null) ? $msgSucesso : "" ; ?>
53             </span>
54         </div>
55         <section><form action="filmeAtualizar.php" method="POST">
56             Id: <input type="text" name="id" readonly value="<?php echo $id; ?>" />
57             Título: <input type="text" name="titulo" value="<?php echo $titulo; ?>" />
58             Avaliação:
59             <input type="number" min="0" max="10" step="0.1" name="avaliacao" value="<?php echo $avaliacao; ?>" />
60             <input type="submit" value="Enviar" />
61             <input type="reset" value="Limpar" />
62         </form></section><br/>
63     </main>
64     <footer>
65         <hr/>
66         Todos os direitos reservados
67     </footer>
68 </body>
69 </html>
```

```
file: filmeAtualizar.php > $msgSucesso
1  <?php
2  require_once('conexao.php');
3  $id = (isset($_POST['id']) && $_POST['id'] != null) ? $_POST['id'] : "";
4  $titulo = (isset($_POST['titulo']) && $_POST['titulo'] != null) ? strtoupper($_POST['titulo']) : "";
5  $avaliacao = (isset($_POST['avaliacao']) && $_POST['avaliacao'] != null) ? $_POST['avaliacao'] : "";
6  $msgErro = null; $msgSucesso = null;
7  if( $id != "" && $titulo != "" && $avaliacao != ""){
8      try {
9          $sql = "UPDATE filmes_assistidos SET titulo=?,avaliacao=? WHERE id=?";
10         $stmt = $conexao->prepare($sql);
11         $stmt->bindParam(1,$titulo);
12         $stmt->bindParam(2,$avaliacao);
13         $stmt->bindParam(3,$id);
14         if ($stmt->execute()) {
15             $msgSucesso = "Filme de id $id atualizado com sucesso!";
16         } else {
17             $msgErro = "Erro: Não foi possível efetuar a atualização no BD";
18         }
19     }catch(PDOException $e) {
20         $msgErro = "Erro: ".$e->getMessage();
21     }finally{
22         if($msgErro != null)
23             header("Location: http://localhost/crud-php/index.php?msgErro=$msgErro");
24         else
25             header("Location: http://localhost/crud-php/index.php?msgSucesso=$msgSucesso");
26         die();
27     }
28 }
29 ?>
```

```
file: filmeExcluir.php > ...
1  <?php
2  require_once('conexao.php');
3  $id = (isset($_GET['id']) && $_GET['id'] != null) ? $_GET['id'] : null;
4  $msgErro = null;
5  $msgSucesso = null;
6  if( $id != null ){
7      try {
8          $sql = "DELETE FROM filmes_assistidos WHERE id=?";
9          $stmt = $conexao->prepare($sql);
10         $stmt->bindParam(1,$id);
11
12         if ($stmt->execute()) {
13             $msgSucesso = "Filme de id $id excluído com sucesso!";
14         } else {
15             $msgErro = "Erro: Não foi possível efetuar a exclusão no BD";
16         }
17     }catch(PDOException $e) {
18         $msgErro = "Erro: ".$e->getMessage();
19     }finally{
20         if($msgErro != null)
21             header("Location: http://localhost/crud-php/index.php?msgErro=$msgErro");
22         else
23             header("Location: http://localhost/crud-php/index.php?msgSucesso=$msgSucesso");
24         die();
25     }
26 }
27 ?>
```

61.CRUD com requisições AJAX/AJAJ

O protocolo HTTP

Nas aplicações web a comunicação entre cliente (browser) e servidor acontece via protocolo HTTP.

O protocolo HTTP (Protocolo de Transferência de Hiper Texto) é executado na camada de Aplicação (Modelo OSI) e usa, por padrão, a porta 80.

Esta comunicação segue o modelo cliente-servidor, que implementa o paradigma "requisição e resposta". Veja a figura:



O protocolo HTTP dispõe de diversos **comandos** (métodos). Já pensando em arquitetura MVC e modelo REST, assuntos abordados mais adiante, utilizaremos 4 métodos de acordo com as operações de banco de dados que pretendemos realizar. Lembrando que independentemente do método/operação, estaremos sempre enviando e recebendo informações no formato JSON.

- **GET**: Solicita informações do banco de dados e pode retornar uma ou muitas linhas. Trata-se, portanto, de uma consulta (SELECT).
- **POST**: Envia dados para serem INSERIDOS (INSERT) no banco de dados.
- **DELETE**: Envia dados para serem REMOVIDOS (DELETE) do banco de dados.
- **PUT**: Envia dados para serem ALTERADOS (UPDATE) no banco de dados.

Por padrão, definimos que sempre que o servidor devolver uma resposta para o cliente, essa resposta será um objeto chamado resposta com os atributos erro, msg e dados. Veja ilustração abaixo:

resposta	
erro	Sempre vai conter true ou false indicando se houve um erro ou não em relação ao que foi solicitado ao servidor
msg	Em caso de erro = true, vai conter a mensagem de erro a ser exibida para o cliente. Em caso de erro = false, vai conter uma mensagem de sucesso que pode ou não ser exibida para o cliente.
dados	Geralmente vai conter o(s) dado(s) solicitado(s) ao servidor via método GET (SELECT) ou null se esses dados não existirem. No caso dos métodos POST, PUT e DELETE é muito provável que contenha sempre null.

- LISTAR
- ➔ Respondendo o cliente (Listar – método GET)

Baseado no formato estabelecido para a devolução de uma resposta para o cliente, podemos criar uma função responder em um arquivo funcoesUtil.php. Veja como pode ficar nossa função.

```
14 function responder(bool $erro, string $msg, array $dados=null){  
15     //Definindo o formato e o tipo de acentuação da resposta que será enviada ao cliente  
16     header("Content-Type:application/json;charset=utf-8");  
17     //Definindo um array com os dados da resposta a ser enviada ao cliente  
18     $resposta = ["erro"=>$erro, "msg"=>$msg, "dados"=>$dados];  
19     //Utilizando a função json_encode para transformar a resposta em um JSON serializado  
20     $respostaJSON = json_encode($resposta);  
21     //Já que o envio da resposta no formato JSON é a última etapa da comunicação com o cliente,  
22     //utilizamos die no lugar de echo.  
23     die($respostaJSON);  
24 }
```

Nesse mesmo arquivo funcoesUtil.php podemos definir uma função que devolve uma conexão com o banco de dados via PDO. Observe.

```
2     function getConexao(){
3         //Declara uma conexão (um objeto do tipo PDO)
4         $pdo=null;
5         try{
6             $dsn="mysql:host=localhost;dbname=supermercado;charset=utf8";
7             $pdo = new PDO($dsn,"root","");
8             //Coloca o PDO para trabalhar em modo de exceção
9             $pdo->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
10            //Retorna a conexão via pdo
11            return $pdo;
12        }catch(PDOException $e){
13            /*Ao encontrar algum erro/ inconsistência com o BD, respondemos o cliente com uma
14            mensagem em português concatenada com a mensagem nativa de PDOException*/
15            responder(true,"Erro ao conectar com o BD. {$e->getMessage()}");
16        }
17    }
```

Observe que na linha 15 já estamos utilizando a função responder passando apenas os 2 primeiros parâmetros que são os obrigatórios.

Esse arquivo funcoesUtil.php geralmente fica dentro de uma pasta chamada model.

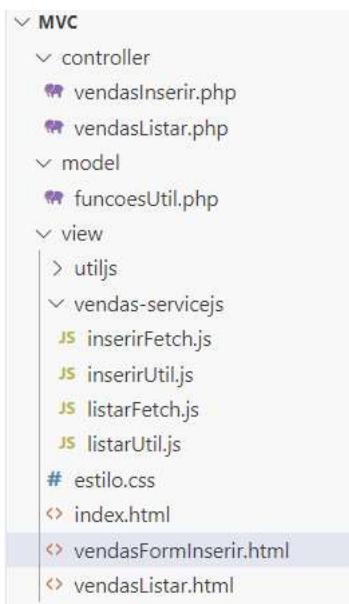
No caso de uma consulta, esses dados geralmente são transformados em uma matriz associativa via método fetchAll recebendo como argumento a constante PDO::FETCH_ASSOC.

```
2     //Obtenha a conexão com o banco de dados
3     require_once("../model/funcoesUtil.php");
4     //Solicitando a conexão com o banco de dados
5     $conexao = getConexao();
6     //Tente executar o comando de consulta na referida tabela
7     try{
8         $sql = <<<SQL
9             SELECT * from vendas_q1
10            SQL;
11         $stmt = $conexao->prepare($sql);
12         $stmt->execute();
13         //Em caso de sucesso, envie as linhas da tabela pelo atributo dados de $resposta
14         //Esses dados deverão estar no formato de matriz associativa (PDO::FETCH_ASSOC)
15         $vendas = $stmt->fetchAll(PDO::FETCH_ASSOC);
16         //Erro,msg,dados (ou não)
17         responder(false,"Vendas do 1º quadrimestre listadas com sucesso!",$vendas);
18     }catch(PDOException $e){
19         responder(true,"Erro ao listar vendas do 1º quadrimestre.");
20     }
```

Esse poderia ser o código de um arquivo vendasListar.php dentro de uma pasta controller.

Perceba que escrevemos todo o código do lado servidor para uma requisição via método GET. Quando definimos um formato padrão de comunicação (JSON) e um objeto padrão de resposta (resposta com erro, msg e dados) utilizando esse formato, tanto faz escrever o lado cliente ou o lado servidor antes.

Considerando que nosso projeto se chama mvc, teríamos a seguinte estrutura de pastas.



Para listar as vendas, já mostramos e explicamos os arquivos model/funcoesUtil.php e controller/vendasListar.php.

➔ O lado cliente (Listar – método GET)

No lado cliente temos que escrever os seguintes arquivos, por enquanto:

- view/index.html
 - Vai conter apenas um menu e referência ao arquivo fornecido view/estilo.css.
- view/vendasListar.html
 - Além do menu vai conter uma tabela onde renderizaremos as linhas em seu tbody. Esse arquivo referencia view/vendas-servicejs/listarFetch.js como um módulo.
- view/vendas-servicejs/listarUtil.js
 - Vai exportar algumas funções úteis para o arquivo citado abaixo.
- view/vendas-servicejs/listarFetch.js
 - Vai disparar, utilizando o método GET, uma requisição ajax (via fetch para controller/vendasListar.php) solicitando as vendas para exibir na tabela.

Seguem os arquivos e comentários quando for pertinente.

view/index.html

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Listagem de Vendas</title>
8      <link rel="stylesheet" href="estilo.css" />
9  </head>
10 <body>
11     <h2 class="titulo">Menu principal</h2>
12     <nav>
13         <ul class="menu">
14             <li><a href="index.html">Home</a></li>
15             <li><a href="vendasFormInserir.html">Inserir nova venda</a></li>
16             <li><a href="vendasListar.html">Listar vendas</a></li>
17         </ul>
18     </nav>
19 </body>
20 </html>
```

view/vendasListar.html

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Listagem de Vendas</title>
8      <link rel="stylesheet" href="estilo.css" />
9  </head>
10 <body>
11     <nav>
12         <ul class="menu">
13             <li><a href="index.html">Home</a></li>
14             <li><a href="vendasFormInserir.html">Inserir nova venda</a></li>
15         </ul>
16     </nav>
17     <main>
18         <h2 class="titulo">Lista de Vendas</h2>
19         <hr/>
20         <div id="erro">
21             <span id="msgErro"></span> <!--Exibição em vermelho--&gt;
22             &lt;span id="msgSucesso"&gt;&lt;/span&gt; <!--Exibição em azul--&gt;
23         &lt;/div&gt;
24         &lt;table id="tblVendas"&gt;
25             &lt;thead&gt;
26                 &lt;tr&gt;&lt;th&gt;ID&lt;/th&gt;&lt;th&gt;CLIENTE&lt;/th&gt;&lt;th&gt;VALOR&lt;/th&gt;&lt;th&gt;QUADRIMESTRE&lt;/th&gt;&lt;/tr&gt;
27             &lt;/thead&gt;
28             &lt;tbody&gt;
29                 &lt;!-- Aqui a tabela vai ser renderizada --&gt;
30             &lt;/tbody&gt;
31         &lt;/table&gt;
32         &lt;br/&gt;
33     &lt;/main&gt;
34     &lt;footer&gt;
35         &lt;hr/&gt;
36         Todos os direitos reservados
37     &lt;/footer&gt;
38     &lt;script src="vendas-servicejs/listarFetch.js" type="module"&gt;&lt;/script&gt;
39 &lt;/body&gt;
40 &lt;/html&gt;</pre>
```

view/vendas-servicejs/listarUtil.js

```
1  function montarTabela(vendas){
2      //Recupera o corpo da tabela
3      const corpoTabela = document.querySelector('tbody');
4      //percorre o array de vendas criando uma linha para cada venda e pendurando no corpo da tabela
5      vendas.forEach(venda=>{
6          const linha = criarLinha(venda);
7          corpoTabela.appendChild(linha);
8      });
9  }//Fim da função
10 function criarLinha(venda){
11     //Para cada venda, use a função map para criar 4 colunas
12     const tr = document.createElement('tr');
13     const [ cId, cCliente, cValor, cQuadrimestre ] =
14     [ 'td', 'td', 'td', 'td' ].map( coluna => document.createElement(coluna) );
15     //Pendure de uma única vez as 4 colunas
16     tr.append(cId, cCliente, cValor, cQuadrimestre);
17     //Extraia os atributos de venda para 4 variáveis
18     let {id,cliente,valor_venda,quadrimestre} = venda;
19     //Preencha cada coluna com seu valor (conteúdo de sua variável)
20     cId.textContent = id;
21     cCliente.textContent = cliente;
22     cValor.textContent = valor_venda;
23     cQuadrimestre.textContent = quadrimestre;
24     //Retorne a linha criada
25     return tr;
26 }
27 //Funções que tratam a resposta
28 function fcSucessoListarVendas(resposta){
29     let dados = resposta.dados;
30     montarTabela(dados);
31 }
32 function fcErroListarVendas(msg){
33     exibirMensagemDeErro(msg);
34 }
35 function exibirMensagemDeErro(msg){
36     document.querySelector('#msgErro').textContent = msg;
37     return;
38 }
39 }
40
41 export {fcSucessoListarVendas,fcErroListarVendas}
```

view/vendas-servicejs/listarFetch.js

```
1  //Fazendo os imports
2  import { fcErroListarVendas, fcSucessoListarVendas } from "./util.js";
3  //Utilizando IIFE (Função autoinvocável)
4  (()=>{
5      fetch("../controller/vendasListar.php")
6          .then(resposta => { //Cada then representa uma promise
7              if(! resposta.ok) //Lance um erro a ser capturado pelo catch
8                  throw new Error(resposta.status + " - " + resposta.statusText);
9              return resposta; //Retorne a resposta dando prosseguinte à promise
10         })
11         .then(resposta => resposta.json()) //Equivalente à função JSON.parse()
12         .then(resposta=>{
13             if(resposta.erro === false){
14                 fcSucessoListarVendas(resposta);
15             }else{
16                 fcErroListarVendas(resposta.msg);
17             }
18         })
19         .catch(error=>fcErroListarVendas(error));
20     /*O catch capture o erro lançado ou qualquer outra falha de comunicação
21     com o servidor (status a partir de 400)*/
22 })();
```

- INSERIR

➔ O lado cliente (Inserir – método POST)

view/vendasFormInserir.html

```
1  <!DOCTYPE html><html lang="pt-br">
2  <head>
3      <meta charset="UTF-8">
4      <meta http-equiv="X-UA-Compatible" content="IE=edge">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Supermercado</title>
7      <link rel="stylesheet" href="estilo.css" />
8  </head>
9  <body>
10     <nav>
11         <ul class="menu">
12             <li><a href="index.html">Home</a></li>
13             <li><a href="vendasListar.html">Listar vendas</a></li>
14         </ul>
15     </nav>
16     <main>
17         <h2 class="titulo">Inserir nova venda</h2><hr/>
18         <div id="erro">
19             <span id="msgErro"></span>
20             <span id="msgSucesso"></span>
21         </div>
22         <section>
23             <form>
24                 Cliente: <input type="text" name="cliente" id="cliente"/>
25                 Quadrimestre: <select name="cmbQuadrimestres" id="cmbQuadrimestres">
26                     <option value="1"> 1 </option>
27                     <option value="2"> 2 </option>
28                     <option value="3"> 3 </option>
29                 </select>
30                 Valor: <input type="number" min="10" step="0.1" name="valor" id="valor" />
31                 <input type="submit" value="Enviar" id="enviar"/>
32                 <input type="reset" value="Limpar" />
33             </form>
34             </section><br/>
35         </main>
36         <footer>
37             <hr/>
38             Todos os direitos reservados
39         </footer>
40         <script src="vendas-servicejs/inserirFetch.js" type="module"></script>
41     </body></html>
```

view/vendas-servicejs/inserirUtil.js

```
1  function fcSucessoInserirVenda(resposta){
2      //Exibe a mensagem por 2 segundos e redireciona p/ vendasListar.html
3      document.querySelector('#msgSucesso').textContent = resposta.msg;
4      setTimeout(function(){
5          window.location.href = "../view/vendasListar.html";
6      },2000);
7  }
8
9  function fcErroInserirVenda(msg){
10     exibirMensagemDeErro(msg);
11 }
12
13 function exibirMensagemDeErro(msg){
14     document.querySelector('#msgErro').textContent = msg;
15     return;
16 }
17 export{fcSucessoInserirVenda, fcErroInserirVenda}
```

view/vendas-servicejs/inserirFetch.js

```
1 import { fcSucessoInserirVenda, fcErroInserirVenda } from "./inserirUtil.js";
2 //Recupera o elemento (Também poderíamos usar o form e o evento submit)
3 const btnEnviar = document.querySelector('#enviar');
4 btnEnviar.addEventListener('click', function(event){
5     //Evita o comportamento default de submeter o formulário
6     event.preventDefault();
7     //Monta um objeto venda recuperando os elementos do DOM
8     let cliente = document.querySelector('#cliente').value;
9     let quadrimestre = parseInt(document.querySelector('#cmbQuadrimestres').value);
10    let valor = parseFloat(document.querySelector('#valor').value);
11    let venda = {
12        "cliente": cliente,
13        "quadrimestre" : quadrimestre,
14        "valor" : valor
15    };
16
17    fetch("../controller/vendasInserirr.php",{
18        method : "POST",
19        body : JSON.stringify(venda), //texto JSON
20        headers : {
21            "Content-Type" : "application/json; charset=UTF-8"
22        }
23    })
24    .then(resposta => {//Cada then representa uma promise
25        if(!resposta.ok) //Lance um erro a ser capturado pelo catch
26            throw new Error(resposta.status + " - " + resposta.statusText);
27        return resposta; //Retorne a resposta dando prosseguimento à promise
28    })
29    .then(resposta => resposta.json()) //Equivalente à função JSON.parse()
30    .then(resposta=>{
31        if(resposta.erro==false)
32            fcSucessoInserirVenda(resposta);
33        else
34            fcErroInserirVenda
35    })
36    .catch(erro=>fcErroInserirVenda(erro));
37 })
```

Linhas 17 a 22 → Observe que quando se trata de um método diferente de GET, precisamos montar um cabeçalho informando o método (nesse caso POST), o body (a informação a ser enviada transformada em JSON serializado utilizando a função JSON.stringify(venda)) e os cabeçalhos informando o formato e o tipo de acentuação dos dados que estamos enviando para o servidor.

Dica: Se você não quiser poluir o escopo, pode utilizar uma IIFE.

➔ Respondendo o cliente (Inserir – método POST)

controller/vendasInserir.php

IMPORTANTE: Leia atentamente os comentários.

```
1 <?php
2 //Obtenha a conexão com o banco de dados
3 require_once("../model/funcoesUtil.php");
4 $conexao = getConexao();
5 //Recupera o texto JSON via post (Quando se envia via requisição AJAX, não temos $_POST)
6 $vendaPost = file_get_contents('php://input');
7 //Decodifica o texto JSON p/ uma matriz associativa (argumento true de json_decode)
8 $vendaMatriz = json_decode($vendaPost, true);
9
10 $cliente = (isset($vendaMatriz["cliente"])) && $vendaMatriz["cliente"] != null) ?
11 strToUpper($vendaMatriz["cliente"]) : "";
12 $quadrimestre = (isset($vendaMatriz["quadrimestre"])) && $vendaMatriz["quadrimestre"] != null) ?
13 $vendaMatriz["quadrimestre"] : "";
14 $valor = (isset($vendaMatriz["valor"])) && $vendaMatriz["valor"] > 0) ? $vendaMatriz["valor"] : "";
15
16 if( $cliente != "" && $quadrimestre != "" && $valor!= ""){
17     try {
18         $tabela = "vendas_q".$quadrimestre;
19         $sql = "INSERT INTO $tabela(cliente, quadrimestre, valor_venda) VALUES(?, ?, ?)";
20         //Prepara a instrução e em seguida passa os argumentos. Evita SQL Injection
21         $stmt = $conexao->prepare($sql);
22         $stmt->bindParam(1, $cliente );
23         $stmt->bindParam(2, $quadrimestre);
24         $stmt->bindParam(3, $valor);
25         $stmt->execute();
26         responder(false, "{$stmt->rowCount()} venda inserida com sucesso! O id inserido
27 foi {$conexao->lastInsertId()}");
28     }catch(PDOException $e) {
29         responder(true, "Erro: Não foi possível efetuar a inserção no BD". $e->getMessage());
30     }
31 }
32 ?>
```

Exercícios:

- 1) Crie uma base de dados e uma tabela da sua escolha (não pode ser vendas, nem filmes ou qualquer outro exemplo utilizado anteriormente)
- 2) Utilizando a mesma estrutura de pastas e o mesmo padrão de projeto, crie um sistema capaz de listar as informações contidas no banco de dados e inserir outras novas informações. O professor deverá fornecer os arquivos html e css.
- 3) Se possível, refatore ainda mais o código.

Exercícios para excluir uma venda.

- EXCLUIR

➔ O lado cliente (Inserir – método DELETE)

Ajustes para trabalhar com a exclusão de linhas.

- 1) Utilize o projeto fornecido pelo professor.
- 2) Copie o código de script.sql e execute.
- 3) Vamos alterar nossa lógica de listar para que agora possamos listar as vendas de todos os 3 quadrimestres. Em ..controller\vendasListar.php, faça a seguinte alteração:

```
8     $sql = <<<SQL
9         SELECT * from vendas_q1
10        UNION
11        SELECT * from vendas_q2
12        UNION
13        SELECT * from vendas_q3
14        ORDER BY quadrimestre,id
15
16 SQL;
```

- 4) Altere o arquivo html responsável por listar as vendas adicionando uma coluna referente às ações na tabela de vendas.

view/vendasListar.html

```
25 <thead>
26   <tr><th>ID</th><th>CLIENTE</th><th>VALOR</th><th>QUADRIMESTRE</th><th>AÇÕES</th></tr>
27 </thead>
```

- 5) Observe atentamente todas as funções contidas em utilJs/sistemaUtil.js
- 6) Observe os arquivos listarUtil.js e inserirUtil.js.
- 7) Experimente fazer uma inclusão para ver tudo funcionando com uma janela modal.
- 8) Em listarUtil.js, na função montarTabela, limpe o corpo da tabela antes de recarregar com novas informações. Já na função criarLinha, cuide para que seja criada uma 5ª coluna e que nela seja colocado um link ([Excluir]) com os atributos idVenda e quadrimestre. Esses atributos nos serão muito úteis mais adiante.

```
3 function montarTabela(vendas){
4   //Recupera o corpo da tabela
5   const corpoTabela = document.querySelector('tbody');
6   corpoTabela.innerHTML = "";
```

Atenção para as linhas 16, 17, 19 e 27

```
13 function criarLinha(venda){
14   //Para cada venda, use a função map para criar 4 colunas
15   const tr = document.createElement('tr');
16   const [ cId, cCliente, cValor, cQuadrimestre, cAcoes ] =
17     [ 'td', 'td', 'td', 'td' ].map( coluna => document.createElement(coluna) );
18   //Pendure de uma única vez as 4 colunas
19   tr.append(cId, cCliente, cValor, cQuadrimestre, cAcoes);
20   //Extraia os atributos de venda para 4 variáveis
21   let {id,cliente,valor_venda,quadrimestre} = venda;
22   //Preencha cada coluna com seu valor (conteúdo de sua variável)
23   cId.textContent = id;
24   cCliente.textContent = cliente;
25   cValor.textContent = valor_venda;
26   cQuadrimestre.textContent = quadrimestre;
27   cAcoes.innerHTML = `<a href="#" idVenda=${id} quadrimestre=${quadrimestre}>[Excluir]</a>`;
28   //Retorne a linha criada
29   return tr;
30 }
```

- 9) Perceba que todo o conteúdo do fetch de listarFetch foi colocado dentro de uma função listar() em listarUtil.js e em seguida foi exportada. Parte final de listarUtil.js

```
32 function listar(){
33     fetch("../controller/vendasListar.php")
34     .then(resposta => { //Cada then representa uma promise
35         if(! resposta.ok) //Lance um erro a ser capturado pelo catch
36         |   throw new Error(resposta.status + " - " + resposta.statusText);
37         return resposta; //Retorne a resposta dando prosseguimento à promise
38     })
39     .then(resposta => resposta.json()) //Equivalente à função JSON.parse()
40     .then(resposta=>{
41         if(resposta.erro === false){
42             fcSucessoListarVendas(resposta);
43         }else{
44             fcErroListarVendas(resposta.msg);
45         }
46     })
47     .catch(erro=>fcErroListarVendas(erro));
48 }
49
50 //Funções que tratam a resposta
51 function fcSucessoListarVendas(resposta){
52     let dados = resposta.dados;
53     montarTabela(dados);
54 }
55 function fcErroListarVendas(msg){
56     exibirMensagemDeErro(document.querySelector(".msgErro"),msg)
57 }
58
59 export {fcSucessoListarVendas,fcErroListarVendas, listar}
```

- 10) Agora, em listarFetch.js, bastou importar e invocar a função. Veja como ficou listarUtil. Observe também os botões para carregar o formulário e abrir o modal (linhas 8 a 18)

```
1 //Fazendo os imports
2 import { listar } from "./listarUtil.js";
3 import { preencherHtml } from "../utiljs/sistemaUtil.js";
4 //Utilizando IIFE (Função autoinvocável)
5 (()=>{
6     //carregar a lista
7     listar();
8     //carrega o formInserir e abre o modal
9     const btnInserir = document.querySelector("#btnInserirVenda");
10    btnInserir.addEventListener('click',()=>{
11        preencherHtml("formInserir.html", document.querySelector("#modalFormInserir"));
12        document.querySelector(".meuModal").showModal();
13    })
14    //Fecha o modal
15    const btnFecharModal = document.querySelector("#btnFecharModal");
16    btnFecharModal.onclick=()=>{
17        document.querySelector(".meuModal").close();
18    }
19 })();
```

- 11) Ainda é possível refatorar nosso código. Perceba que todo o código de uma fetch está bem parecido em todas as lógicas. Podemos criar uma função meuFetch em utilJs/obtemRespostaFetch.js. Essa função receberia a url do recurso solicitado, o método (GET, POST, PUT, DELETE), uma função de call-back para o caso de sucesso e outra função de call-back para o caso de erro e, por último receberia uma informação a ser enviada (dado) ou null. Não se esqueça de exportar essa função!

```
1  function meuFetch(metodo, url, dados = null){
2      let configMetodo = null;
3      if(metodo !=="GET"){
4          configMetodo = {
5              method : metodo,
6              body : JSON.stringify(dados), //texto JSON
7              headers : {
8                  "Content-Type" : "application/json;charset=UTF-8"
9              }
10         }
11     }
12
13    const meuFetch = fetch(url,configMetodo)
14    .then(resposta => verificarSeTemErro(resposta)) //devolve o retorno da função
15    .then(resposta => resposta.json()) //devolve a promessa de que virá um Response JSON
16    return meuFetch;
17
18    function verificarSeTemErro(resposta){
19        console.log(resposta);
20        if(! resposta.ok)
21            | throw new Error(resposta.status + " - " + resposta.statusText);
22        return resposta;
23    }
24
25
26 export {meuFetch}
```

- 12) Agora faça com que inserirFetch utilize essa nossa função sem esquecer de importar. Veja como ficou bem mais simples.

```
1  import { fcSucessoInserirVenda, fcErroInserirVenda } from "./inserirUtil.js";
2  import { meuFetch } from "../utiljs/obtemRespostaFetch.js";
3  //Recupera o elemento (Também poderíamos usar o form e o evento submit)
4  const formEnviar = document.querySelector('#modalFormInserir');
5  formEnviar.addEventListener('submit', function(event){
6      //Evita o comportamento default de submeter o formulário
7      event.preventDefault();
8      //Monta um objeto venda recuperando os elementos do DOM
9      let cliente = document.querySelector('#cliente').value;
10     let quadrimestre = parseInt(document.querySelector('#cmbQuadrimestres').value);
11     let valor = parseFloat(document.querySelector('#valor').value);
12     let venda = {
13         "cliente": cliente,
14         "quadrimestre" : quadrimestre,
15         "valor" : valor
16     };
17     //Faz o fetch
18     meuFetch("POST","../controller/vendasInserir.php",venda)
19     .then(resposta=>{
20         if(resposta.erro==false)
21             | fcSucessoInserirVenda(resposta);
22         else
23             | fcErroInserirVenda(resposta.msg);
24     })
25     .catch(erro=>fcErroInserirVenda(erro));
26 })
```

- 13) Agora faça com que listar de listarUtil utilize essa nossa função (Não esqueça de importar meuFetch). Veja como ficou a parte final de listarUtil.js. Observe também as funções de sucesso e erro.

```
33  function listar(){
34      meuFetch("GET", "../controller/vendasListar.php")
35      .then(resposta=>{
36          if(resposta.erro === false){
37              fcSucessoListarVendas(resposta);
38          }else{
39              fcErroListarVendas(resposta.msg);
40          }
41      })
42      .catch(erro=>fcErroListarVendas(erro));
43  }
44
45 //Funções que tratam a resposta
46 function fcSucessoListarVendas(resposta){
47     let dados = resposta.dados;
48     montarTabela(dados);
49 }
50 function fcErroListarVendas(msg){
51     exibirMensagemDeErro(document.querySelector(".msgErro"),msg)
52 }
53
54 export {fcSucessoListarVendas,fcErroListarVendas, listar}
```

14) Crie o arquivo excluirUtil.js com especial atenção às linhas 1 e 9.

```
1 import { listar } from "./listarUtil.js";
2 import { exibirMensagemDeErro, exibirMensagemDeSucesso } from "../utiljs/sistemaUtil.js";
3
4 function fcSucessoExcluirVenda(resposta){
5     //Exibe a mensagem por 2 segundos e redireciona p/ vendasListar.html
6     exibirMensagemDeSucesso(document.querySelector(".msgSucesso"),resposta.msg,2000,listar)
7     document.querySelector('.meuModal').close();
8 }
9
10 function fcErroExcluirVenda(msg){
11     exibirMensagemDeErro(document.querySelector(".msgErro"),msg)
12 }
13
14 export [fcSucessoExcluirVenda, fcErroExcluirVenda]
```

15) Crie o arquivo excluirFetch.js com a todas as linhas.

```
1 import { fcSucessoExcluirVenda, fcErroExcluirVenda } from "./excluirUtil.js";
2 import { meuFetch } from "../utiljs/obtemRespostaFetch.js";
3 (()=>{
4     const corpoTabela = document.querySelector('tbody');
5     corpoTabela.onclick = function(evento){
6         if(evento.target.tagName === 'A'){
7             const link = evento.target;
8             let idVenda = parseInt(link.getAttribute("idVenda"));
9             let quadrimestreVenda = parseInt(link.getAttribute("quadrimestre"));
10            if(confirm(`Deseja realmente excluir a venda de id ${idVenda} e quadrimestre ${quadrimestreVenda}?`)){
11                let venda = {id:idVenda, quadrimestre:quadrimestreVenda};
12                meuFetch("DELETE", "../controller/vendasExcluir.php",venda)
13                .then(resposta=>{
14                    if(resposta.erro==false)
15                        fcSucessoExcluirVenda(resposta);
16                    else
17                        fcErroExcluirVenda(resposta.msg);
18                })
19                .catch(erro=>fcErroExcluirVenda(erro))
20            }
21        }
22    }
23 })();
```

16) Não se esqueça de referenciar excluirFetch.js como módulo em vendas.html

```
8 <link rel="stylesheet" href="estilo.css" />
9 <script src="vendas-servicejs/listarFetch.js" type="module" defer></script>
10 <script src="vendas-servicejs/inserirFetch.js" type="module" defer></script>
11 <script src="vendas-servicejs/excluirFetch.js" type="module" defer></script>
12 </head>
```

17) Crie, no lado servidor, controller/vendasExcluir.php

```
1 <?php
2 //Obtenha a conexão com o banco de dados
3 require_once("../model/funcoesUtil.php");
4 $conexao = getConexao();
5 //Recupera o texto JSON via post (Quando se envia via requisição AJAX, não temos $_POST)
6 $vendaDelete = file_get_contents('php://input');
7 //Decodifica o texto JSON p/ uma matriz associativa (argumento true de json_decode)
8 $vendaMatriz = json_decode($vendaDelete, true);
9
10 $id = (isset($vendaMatriz["id"]) && $vendaMatriz["id"] != null) ?
11 intval($vendaMatriz["id"]) : "";
12 $quadrimestre = (isset($vendaMatriz["quadrimestre"]) && $vendaMatriz["quadrimestre"] != null) ?
13 intval($vendaMatriz["quadrimestre"]) : "";
14
15 if( $id != "" && $quadrimestre != ""){
16     try {
17         $tabela = "vendas_q".$quadrimestre;
18         $sql = "DELETE FROM $tabela WHERE id=? AND quadrimestre=?";
19         //Prepara a instrução e em seguida passa os argumentos. Evita SQL Injection
20         $stmt = $conexao->prepare($sql);
21         $stmt->bindParam(1, $id );
22         $stmt->bindParam(2, $quadrimestre);
23         $stmt->execute();
24         responder(false,"{$stmt->rowCount()} venda de id {$id} e quadrimestre {$quadrimestre} excluída com sucesso!
25 Linhas afetadas: {$conexao->lastInsertId()}");
26     }catch(PDOException $e) {
27         responder(true,"Erro: Não foi possível efetuar a exclusão no BD".$e->getMessage());
28     }
29 }else
30     responder(true,"Erro: Não foi possível efetuar a exclusão. Informações incompletas enviadas.");
31 ?>
```

- 18) Teste todas as operações e perceba que ao excluir uma venda, não precisamos mais redirecionar. Afinal, já estamos na página correta. Basta recarregar os dados da tabela.
- 19) Ainda é possível refatorar nosso código para trabalhar melhor com async/await. Perceba que o código de início de uma fetch até o momento em que se devolve uma resposta como objeto literal Javascript com resposta.json() está bem parecido em todas as lógicas. Podemos criar uma função respostaFetch em útilJs/obtemRespostaFetch.js. Essa função receberia todas as informações necessárias para criar e retornar um fetch com a promessa de uma resposta padrão fetch ou até mesmo no formato Javascript literal. Faremos no padrão fetch. Não se esqueça de exportar essa função! Veja como deve ficar o fim do arquivo:

```
26 async function respostaFetch(metodo, url, dado = null){
27     let configMetodo = null;
28     if(metodo != "GET"){
29         configMetodo = {
30             method: metodo,
31             body: JSON.stringify(dado),
32             headers: {
33                 "Content-Type": "application/json; charset=UTF-8"
34             }
35         }
36     }
37     const resposta = await fetch(url,configMetodo)
38     if(! resposta.ok) //Lance um erro a ser capturado pelo catch
39         throw new Error(resposta.status + " - " + resposta.statusText);
40     return resposta; //Retorne a resposta dando sequencia à promise
41 }
42
43 export {meuFetch, respostaFetch}
```

- 20) Veja como ficaria um exemplo da lógica de excluir referenciando excluirFetch2.js em vendasListar.html

```
1 import { fcSucessoExcluirVenda, fcErroExcluirVenda } from "./excluirUtil.js";
2 import { respostaFetch } from "../utiljs/obtemRespostaFetch.js";
3 ()=>{
4     const corpoTabela = document.querySelector('tbody');
5     corpoTabela.onclick = async function(evento){
6         if(evento.target.tagName === 'A'){
7             const link = evento.target;
8             let idVendaExcluir = parseInt(link.getAttribute("idVenda"));
9             let quadrimestreExcluir = parseInt(link.getAttribute("quadrimestre"));
10            if(confirm(`Deseja realmente excluir a venda de id ${idVendaExcluir} e quadrimestre ${quadrimestreExcluir}?`)){
11                let venda = {id:idVendaExcluir, quadrimestre:quadrimestreExcluir};
12                try{
13                    console.log('inicio')
14                    let resposta = await respostaFetch("DELETE", "../controller/vendasExcluir.php", venda);
15                    console.log(resposta)
16                    resposta = await resposta.json();
17                    console.log(resposta)
18                    if(resposta.erro==false)
19                        fcSucessoExcluirVenda(resposta);
20                    else
21                        fcErroExcluirVenda(resposta.msg);
22                    console.log('fim')
23                }catch(error){
24                    fcErroExcluirVenda(error);
25                }
26            }
27        }
28    })();
29 }
```

- 21) Volte a referenciar excluirFetch em vendas.html
- 22) DESAFIO: No arquivo obtemFetch, escreva uma função meuFetch2 que antes dos dados recebe dois call-backs para sucesso e erro respectivamente.
- 23) Teste essa nova função em suas lógicas.