

**CURSOS
TÉCNICOS**

**DESENVOLVIMENTO DE
SISTEMAS WEB II**

Eixo Informática para Internet

UNIDADE 2

SUMÁRIO

1.	Introdução.....	3
1.1	Funções.....	3
1.1.1	Declarando funções.....	3
1.1.2	Chamando uma função.....	3
1.1.3	Funções com parâmetros.....	4
1.1.4	Retorno de funções.....	5
1.2	Built-in Functions PHP.....	6
1.3	Array/listas no PHP.....	7
1.3.1	Array convencional.....	7
1.3.2	Array Associativo.....	8
1.3.3	Iterando sobre Arrays com foreach.....	8
1.3.4	Redefinindo, adicionando e excluindo valores de um array.....	9
1.3.5	Desestruturando/extraindo partes de um array.....	11
1.3.6	Isolando códigos.....	11
2.	Identificando problemas, criando funções e aplicando melhorias.....	13
2.1	Problema #1.....	14
2.2	Strings complexas.....	15
3.	Desafios práticos da unidade.....	16
4.	Referências.....	17

UNIDADE 2

1. Introdução

A partir desta unidade, vamos trabalhar com o foco de elaborar um projeto cujo tema será as contas bancárias, contemplando ações como: realizar transações, operações como saque, depósito e vínculo a uma pessoa. Com isso, será mais fácil assimilar os conceitos da linguagem PHP, a uma regra de negócio próxima da nossa realidade

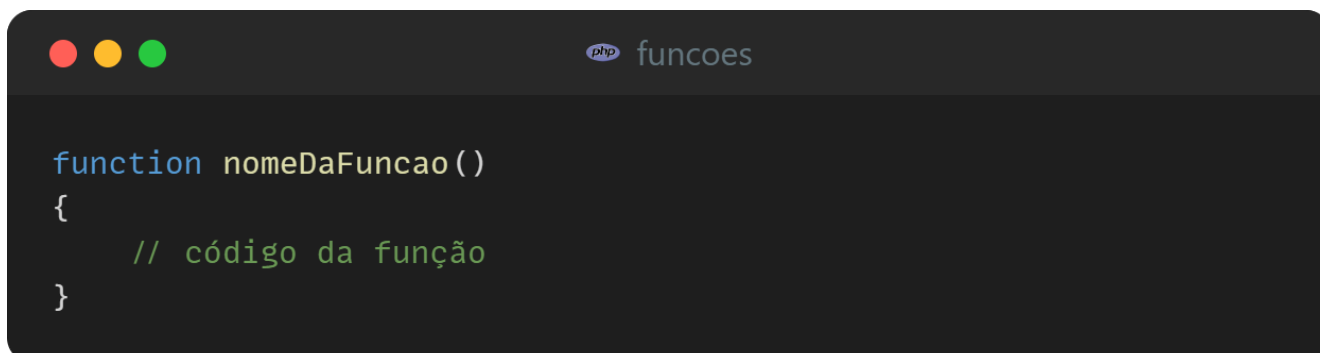
Portanto, vamos iniciar pelas funções, listas, *strings* complexas, separação de códigos no PHP e, evoluiremos para outros aspectos do PHP.

1.1 Funções

No PHP, as funções são blocos de código reutilizáveis que podem ser chamados em seu programa para realizar tarefas específicas. Elas são usadas para organizar e modularizar o código, tornando-o mais fácil de entender e manter.

1.1.1 Declarando funções

Você pode declarar uma função usando a palavra-chave **function**, seguida pelo **nome da função** e **um par de parênteses**, que pode conter parâmetros da função. A estrutura básica é a seguinte:



```

function nomeDaFuncao()
{
    // código da função
}
    
```

1.1.2 Chamando uma função

Para chamar uma função, você simplesmente usa o **nome da função**, seguido por **parênteses**. Se a função aceitar parâmetros, você deve fornecê-los dentro dos parênteses.

```

nomeDaFuncao();
outraFuncaoComParametro($parametro1, $parametro2);
  
```

1.1.3 Funções com parâmetros

Uma função pode aceitar zero ou mais parâmetros, que são valores que você passa para a função para que ela possa trabalhar com eles. Você declara parâmetros dentro dos parênteses da declaração da função. Estes parâmetros podem ser passados via **valor** ou **referência**.

```

function imprimeMensagem($mensagem)
{
    echo $mensagem . PHP_EOL;
}

imprimeMensagem("Hello World!"); //Hello World!
imprimeMensagem("Outra mensagem."); //Outra mensagem.
  
```

Ao chamar a função **imprimeMensagem** o comando **echo** é executado, e a **\$mensagem** passada como argumento para a função é exibida em tela. Esta passagem de argumentos se deu por **meio de valor**, ou seja, nós criamos uma **cópia desse valor** dentro da função, e **qualquer alteração feita na cópia não afeta a variável original** fora da função. Este valor pode ser de diversos tipos como **string, int, float, array e etc.**

Quanto à **passagem por referência**, por outro lado, quando um valor é passado por referência, a função trabalha diretamente com a variável original, e qualquer alteração feita na variável dentro da função também afeta a variável fora da função. Analise o comportamento do código abaixo.

Exemplo de passagem de parâmetro por referência.

```

codigos.php
1 <?php
2 $mensagem = ""; // inicialização de da variável mensagem
3
4 echo "echo 1: " . $mensagem . PHP_EOL; // primeiro echo
5
6 function imprimeMensagem(&$mensagem) // "&", passa a referência para a variável
7 { // mensagem
8     $mensagem = "A variável mensagem teve seu valor alterado aqui";
9 } // é atribuído um valor a var. mensagem
10
11 imprimeMensagem($mensagem); // função é executada. É passado como argumento a var.
12 // mensagem
13 echo "echo 2: " . $mensagem . PHP_EOL; // A variável mensagem teve seu valor alterado aqui.
14

```

PROBLEMAS SAÍDA CONSOLE DE DEPURACÃO TERMINAL PORTAS

PS D:\php\semana-02> php .\codigos.php
echo 1:
echo 2: A variável mensagem teve seu valor alterado aqui

Note que quando inicializada `$mensagem` tinha seu valor como uma string vazia, e isto, foi validado no primeiro `echo` da linha 4. Entretanto, quando executamos a função na linha 11 passando como argumento a variável `$mensagem` e recebendo esta referência na declaração da função na linha 6, podemos então alterar o valor desta variável na linha 8, e isto, irá refletir, inclusive, fora do escopo da função. No `echo` da linha 13, notamos que de fato, `$mensagem` teve seu valor alterado, e isto aconteceu dentro da função.

Note que, para definir um parâmetro de uma função como uma referência adicionamos o "&" (E comercial) antes da declaração do parâmetro.

1.1.4 Retorno de funções

Segundo a documentação oficial do PHP, os valores podem ser retornados utilizando a instrução opcional `return`. Qualquer tipo pode ser retornado, incluindo arrays e objetos. Isto faz com que as funções terminem sua execução imediatamente e passa o controle de volta para a linha de onde ela foi chamada. Ou seja, a instrução `return` encerra a função no momento em que é declarada. Se não houver uma instrução `return` em uma função, então o valor `null` é retornado.

Usando return para retornar um valor.

```

funcoes
<?php
function soma($num1, $num2)
{
    return $num1 + $num2;
}

echo soma(2, 2); // imprime '4'.

```

Ao chamar o comando `echo` e a execução da função temos o valor 4 sendo retornado, pois $2 + 2 = 4$, que é justamente o retorno da função `soma`.

Atribuindo o retorno de uma função a uma variável

```
<?php

function soma($num1, $num2)
{
    return $num1 + $num2;
}

$resultado = soma(2, 2);

echo $resultado; // imprime '4'.
```

Também é possível atribuir o resultado da função “soma” a variável `$resultado`, por exemplo. Neste caso, chamamos `$resultado` que justamente recebeu o resultado de $2 + 2$.

1.2 Built-in Functions PHP

O PHP oferece uma variedade de funções internas ou Built-in Functions populares que simplificam tarefas comuns de programação. Aqui estão algumas das funções internas mais usadas do PHP:

- ✓ **strlen()**: Retorna o comprimento de uma string.
- ✓ **count()**: Retorna o número de elementos em um array.
- ✓ **var_dump()**: Retorna informações mais específicas sobre uma variável como tipo, tamanho e valor.
- ✓ **isset()**: Determina se uma variável é declarada E diferente de **null**. Retorna **true**, se **var existe** E **diferente de nula**.
- ✓ **unset()**: Indetermina/zera uma variável.

Exemplos com as funções. Observe o comentário ao lado para o retorno.

```
<?php

//strlen()
$string = "abcdef";
echo strlen($string); // 6

$stringComEspacos = ' ab cd ';
echo strlen($stringComEspacos ); // 7

//count()
$array = [1, 2, 3, 4];
echo count($array); // 4

$array[] = 5; // Adiciona 5 na última pos.
echo count($array); // 5

//var_dump()
$stringDois = "abcdef";
var_dump($stringDois); // string(6) "abcdef"

//isset()
$stringInicializada = "Uma string";
$varNula = null;

var_dump(isset($stringInicializada)); //bool(true)
var_dump(isset($umaVarQueNaoExiste)); //bool(false)
var_dump(isset($varNula)); //bool(false)

//unset()
$numero = 2;
var_dump($numero); // int(2)

unset($numero); //Remove $numero
var_dump($numero); //NULL e Dispara: PHP Warning: Undefined variable $numero.
```

1.3 Array/listas no PHP

Um array no PHP é na verdade um mapa ordenado, e um mapa é um tipo que relaciona valores a chaves. Este tipo é otimizado para vários usos diferentes: ele pode ser tratado como um array, uma lista (vetor), *hashtable* (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente mais. Assim como existe a possibilidade dos valores do array serem outros arrays, árvores e arrays multidimensionais.

1.3.1 Array convencional

Em um array convencional do PHP, os **elementos são armazenados em índices numéricos sequenciais, começando por 0** e indo até $n-1$, onde "n" é o número de elementos no array. Os elementos são acessados usando os índices numéricos.

```

1 <?php
2
3 $arrayConvencional = [1, 2, 3, 4, 5];
4
5 var_dump($arrayConvencional);
  
```

Definição do array

print das infos da var. \$arrayConvencional

array com 5 elementos

chave numérica

tipo e valor armazenado naquela posição

```

PS D:\php\semana-02> php .\codigos.php
array(5) (
  [0]=>
    int(1)
  [1]=>
    int(2)
  [2]=>
    int(3)
  [3]=>
    int(4)
  [4]=>
    int(5)
)
  
```

Note que, quando declarado apenas os elementos de um array, o PHP automaticamente define as chaves das posições com números inteiros sequenciais começando em 0. Ou seja, para imprimir o número na primeira posição deste array teríamos que:

```

1 <?php
2
3 $arrayConvencional = [1, 2, 3, 4, 5];
4 echo $arrayConvencional[0];
  
```

array

chave/posição do array

1 número acessado na posição 0 do \$arrayConvencional

```

PS D:\php\semana-02> php .\codigos.php
1
  
```

1.3.2 Array Associativo

Em um array associativo, os **elementos são armazenados em pares chave-valor**, onde **cada elemento tem uma chave única associada a ele**. As **chaves podem ser strings ou números**, mas geralmente são strings descritivas para identificar os valores. Os **elementos são acessados usando as chaves associativas**. Confira o exemplo:

```

codigos.php
1  <?php
2
3  $arrayAssociativo = [
4  chave => "nome" => "Rafael", => valor
5  "idade" => 25,
6  "cidade" => "Gravataí"
7  ];
8
9  var_dump($arrayAssociativo);
10
PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS
PS D:\php\semana-02> php .\codigos.php
array(3) {
  ["nome"]=>
  string(6) "Rafael"
  ["idade"]=>
  int(25)
  ["cidade"]=>
  string(9) "Gravataí"
}

```

resultado produzido

Observe que agora, temos chaves que nós mesmos definimos (“nome”, “idade”, “cidade”). Isso faz com a manipulação da lista tenha mais sentido ao acessar as chaves para recuperar os valores.

```

3  $arrayAssociativo = [
4  "nome" => "Rafael",
5  "idade" => 25,
6  "cidade" => "Gravataí"
7  ];
8
9  echo $arrayAssociativo["nome"];
10
PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS
PS D:\php\semana-02> php .\codigos.php
Rafael

```

valor reproduzido

1.3.3 Iterando sobre Arrays com foreach

O foreach é frequentemente usado para iterar arrays associativos e numéricos. Ele percorre todos os elementos do array sem a necessidade de um contador.

Usando o foreach com array numérico:

```

1  <?php
2
3  $arrayConvencional = [1, 2, 3, 4, 5];
4      array para percorrer      nome dado ao item da iteração
5  foreach($arrayConvencional as $numero) {
6      echo $numero . "\t";
7  }      item da      adiciona um "tab", após o número.
          iteração    é outro escape de string
8
9
10

```

PS D:\php\semana-02> php .\codigos.php
 1 2 3 4 5 Resultado em tela
 PS D:\php\semana-02>

Observe que nomeamos o item da iteração como `$número`. Basicamente, o foreach recebe 2 parâmetros: um array para percorrer e um nome para o item da iteração atual. **Para cada (for each) item ele executa o código envolto das chaves.**

Usando o foreach com array associativo:

```

14
15 $arrayAssociativo = [
16     "nome" => "Rafael",
17     "idade" => 25,
18     "cidade" => "Gravataí"
19 ];
20
21 foreach($arrayAssociativo as $chave => $valor) {
22     echo "$chave: $valor" . PHP_EOL;
23 }

```

PS D:\php\semana-02> php .\codigos.php
 nome: Rafael
 idade: 25
 cidade: Gravataí
 PS D:\php\semana-02>

Em arrays associativos, além de extrair o `$valor` da iteração é possível recuperar OU não, a `$chave`. A lógica segue a mesma. Para cada propriedade do array o código envolto das chaves é executado.

1.3.4 Redefinindo, adicionando e excluindo valores de um array

Redefinindo elementos em arrays

```
<?php

$frutas = ["maçã", "banana", "laranja"];
$frutas[1] = "pera"; // Altera o valor na posição 1 para "pera"

$pessoa = [
    "nome" => "Rafael",
    "idade" => 25,
    "cidade" => "Gravataí"
];

$pessoa["nome"] = "Pedro"; // Altera o valor da chave "nome" para "Pedro"
```

Ao chamar o array podemos passar a chave da propriedade que queremos alterar e atribuir o novo valor. Ao printar em tela o array será possível observar a mudança.

Adicionando elementos em arrays

```
<?php

$frutas = ["maçã", "banana", "laranja"];
$frutas[] = "uva"; // Adiciona "uva" ao final do array

$pessoa = [
    "nome" => "Rafael",
    "idade" => 25,
    "cidade" => "Gravataí"
];

$pessoa["pais"] = "Brasil"; //Adiciona "pais" => "Brasil" ao final do array
```

Ao chamar o array podemos passar a chave da propriedade que queremos adicionar e atribuir um valor. Ao printar em tela o array será possível observar a inserção.

Removendo elementos em arrays

```
<?php

$frutas = ["maçã", "banana", "laranja"];
unset($frutas[1]); // Remove o elemento na posição 1 (no caso, "banana")

$ pessoa = [
    "nome" => "Rafael",
    "idade" => 25,
    "cidade" => "Gravataí"
];

unset($pessoa["cidade"]); // Remove a propriedade com chave "cidade" (no caso, "cidade" => "Gravataí")
```

Neste caso, utilizamos a função unset para remover a chave/posição especificada ao chamar o array.

1.3.5 Desestruturando/extraindo partes de um array

A desestruturação de arrays no PHP é uma técnica que permite atribuir valores de um array a variáveis individuais de forma mais conveniente.

```
9 $frutas = ["maçã", "banana", "laranja"];
10
11 list($fruta1, $fruta2, $fruta3) = $frutas;
12 [$fruta1, $fruta2, $fruta3] = $frutas; //Funciona da mesma forma que list.
13
14 echo $fruta1 . PHP_EOL; // maçã
15 echo $fruta2 . PHP_EOL; // banana
16 echo $fruta3 . PHP_EOL; // laranja
17
```

realizada a desestruturação e atribuição dos valores de acordo com as posições as variáveis e elementos do array.

PROBLEMAS SAÍDA CONSOLE DE DEPURACÃO TERMINAL PORTAS

powershell + v

PS D:\php\semana-02> php .\codigos.php

maçã
banana
laranja

Resultado obtido

PS D:\php\semana-02>

1.3.6 Isolando códigos

Na medida em que o projeto cresce, se faz necessário uma separação do código a fim de evitar arquivos com múltiplas linhas e códigos de diferentes contextos e responsabilidades. Com isso podemos contar com 4 comandos PHP diferentes para separar os nossos códigos de acordo com o cenário que mais se adeque.

- **require():** É usado para incluir um arquivo em um script PHP. Se o arquivo não for encontrado ou houver algum erro na inclusão, ele resultará em um erro fatal (E_COMPILE_ERROR) e interromperá a execução do script.
- **Use require quando o arquivo que você está incluindo é essencial para o funcionamento do seu código,** e você deseja que a falha na inclusão pare o script em caso de problemas.
- **require_once():** É semelhante ao require, mas garante que o arquivo seja incluído apenas uma vez no script, mesmo que seja chamado múltiplas vezes.
- **include():** Include é usado para incluir um arquivo em um script PHP, assim como o require. No entanto, se o arquivo não for encontrado ou houver algum erro na inclusão, ele resultará em um aviso (E_WARNING) e não interromperá a execução do script. Use include quando o arquivo que você está incluindo não é crítico para o funcionamento do código e pode ser tratado de forma mais flexível em caso de erro.
- **include_once():** include_once é semelhante ao include, mas garante que o arquivo seja incluído apenas uma vez, mesmo que seja chamado várias vezes no script.

EXEMPLO PRÁTICO

Vamos separar uma lista de contas em um arquivo “**contas-lista.php**” e renderizar as informações desta lista em outro arquivo “**renderizar-info.php**”.

Para que o “renderizar-info.php” tenha acesso a lista de contas iremos usar o comando **require_once(“<caminho-para-o-arquivo.php>”);**

Neste arquivo, colocamos informações de contas de pessoas fictícias, cuja a chave de cada array de conta é um cpf (“000.000.000-11”) e contém propriedades específicas como “nome”, “email” e “saldo” para cada uma. Esta variável que armazena a lista de contas está no arquivo “contas-lista.php”.

```

1 <?php
2
3 $contas = [
4     "000.000.000-11" => [
5         "nome" => "Rafael",
6         "email" => "rafael@email.com",
7         "saldo" => 300
8     ],
9     "000.000.000-22" => [
10        "nome" => "João",
11        "email" => "joao@email.com",
12        "saldo" => 140
13    ],
14    "000.000.000-33" => [
15        "nome" => "Maria",
16        "email" => "maria@email.com",
17        "saldo" => 700
18    ]
19 ];
  
```

Agora no arquivo “renderiza-info.php” temos:

```

1 <?php
2 require_once("../contas-lista.php"); Com este comando, agora temos acesso a TODO o código
3                                     contido em "contas-lista.php".
4 if(isset($contas)) {
5     echo "Dados dos clientes: " . PHP_EOL;
6
7     foreach($contas as $cpf => $propriedade) {
8         echo "CPF: $cpf" . PHP_EOL;
9         echo "Nome: " . $propriedade["nome"] . PHP_EOL;
10        echo "E-mail: " . $propriedade["email"] . PHP_EOL;
11        echo "Saldo: " . $propriedade["saldo"] . PHP_EOL;
12        echo "-----" . PHP_EOL;
13    }
14 } else {
15     echo "Nenhum cliente encontrado.";
16 }

```

linkamos um arquivo ao outro e, com isso, comparamos caso a variável `$contas` existisse e não fosse nula usando o `isset()`. Caso tudo desse certo, as informações das contas de todos os clientes eram mostradas no terminal. Confira:

```

PS D:\php\semana-02> php ./renderizar-info.php
Dados dos clientes:
CPF: 000.000.000-11
Nome: Rafael
E-mail: rafael@email.com
Saldo: 300
-----
CPF: 000.000.000-22
Nome: João
E-mail: joao@email.com
Saldo: 140
-----
CPF: 000.000.000-33
Nome: Maria
E-mail: maria@email.com
Saldo: 700
-----
PS D:\php\semana-02>

```

E caso, o `require_once`, por algum motivo estivesse apontando para o caminho errado do arquivo no nosso projeto **um erro seria emitido**. Mas, neste caso, se tivéssemos esquecidos de colocar o `require_once` OU se `$contas` não tivesse nenhuma conta cadastrada, **então cairia no else da linha 14** e o usuário iria enxergar a mensagem: “Nenhum cliente encontrado.”

2. Identificando problemas, criando funções e aplicando melhorias

Durante o código mostrado no último item, podemos perceber que existem muitos códigos que se repetem ou que estão pouco otimizados.

2.1 Problema #1

Estamos repetindo `echo` e `PHP_EOL` em todas as linhas. E se tivéssemos uma função para isso?

```

7   foreach($contas as $cpf => $propriedade) {
8       echo "CPF: $cpf" . PHP_EOL;
9       echo "Nome: " . $propriedade["nome"] . PHP_EOL;
10      echo "E-mail: " . $propriedade["email"] . PHP_EOL;
11      echo "Saldo: " . $propriedade["saldo"] . PHP_EOL;
12      echo "-----" . PHP_EOL;
13  }
14  } else {
15      echo "Nenhum cliente encontrado.";
16  }

```

Veja em destaque as partes que se repetem. **Como solução podemos: Criar um novo arquivo para funções auxiliares de nome “funcoes.php”** e chamar através do `require_once` novamente em “renderizar-info.php”

Solução #1: Criação da função `exibeMensagem($mensagem)` que recebe uma mensagem como parâmetro e executa o comando `echo` printando a mensagem. Entretanto, tivemos que fazer uma alteração na passagem da string no “renderizar-info.php”. Confira abaixo como ficaram os arquivos:

funcoes.php



```

1  <?php
2
3  function exibeMensagem($mensagem) {
4      echo $mensagem . PHP_EOL;
5  }

```

renderizar-info.php

```

1 <?php
2 require_once("../contas-lista.php");
3 require_once("../funcoes.php"); // inserção de comando para chamar "funcoes.php"
4
5 if(isset($contas)) {
6     echo "Dados dos clientes: " . PHP_EOL;
7
8     foreach($contas as $cpf => $propriedade) {
9         exibeMensagem("CPF: $cpf"); // chamada da função exibe mensagem passado a string desejada
10        exibeMensagem("Nome: {$propriedade["nome"]}");
11        exibeMensagem("E-mail: {$propriedade["email"]}"); // Note que, adicionamos "{}" envolta da chamada do array
12        exibeMensagem("Saldo: {$propriedade["saldo"]}"); // acessando a chave desejada, pois precisamos usar aspas
13        exibeMensagem("-----"); // duplas 2x
14    }
15 } else {
16     exibeMensagem("Nenhum cliente encontrado");
17 }
    
```

Nestes casos em que precisamos usar 2x aspas duplas como o exemplo das linhas 10 até 12 recorreremos a **sintaxe complexa de string**

2.2 Strings complexas

Strings complexas permitem que você inclua valores de variáveis e expressões **dentro de uma string delimitada por aspas duplas (")** de maneira mais complexa e expressiva.

- ✓ A parte entre chaves { } dentro da string é usada para delimitar a expressão complexa;
- ✓ \$propriedade["nome"], por exemplo, é uma expressão válida que está sendo avaliada e tendo seu resultado incorporado diretamente na string.

Essa sintaxe é útil quando você precisa incorporar valores de variáveis complexas, como **arrays associativos, objetos** ou **expressões mais complexas**, diretamente em uma string **sem a necessidade de concatenar manualmente os valores**. É uma maneira poderosa de criar strings com conteúdo dinâmico no PHP.

Note que reduzimos bastante o número de concatenações das string utilizando de strings complexas + a função exibeMensagem().

3. Desafios práticos da unidade

1. Crie uma lista de bancos

- A chave de cada array de um banco deverá ser o número da agência.
- Deve conter as informações: nome, cidade, estado para cada banco.
- Adicione os dados da tabela abaixo exatamente como estão

Número Agência	Nome	Cidade	Estado
4444	Banco 1	Porto Alegre	Rio Grande do Sul
5555	Banco 2	Rio de Janeiro	Rio de Janeiro
6666	Banco 3	São Paulo	São Paulo
7777	Banco 4	Gravataí	Santa Catarina

2. Realize operações com a lista de bancos

- Liste as informações de todos os bancos cadastrados na lista.

Exemplo: “4444 - Banco 1, Porto Alegre, Rio Grande do Sul. ”

- Adicione o seguinte banco na última posição da lista de bancos

Número Agência	Nome	Cidade	Estado
8888	Banco 5	Cachoeirinha	Rio Grande do Sul

- Remova da lista o banco cujo número de agência = 4444.
- Edite o estado do banco da **agência 7777** para **“Rio Grande do Sul”**.

3. Valide se existe algum banco cadastrado na lista de bancos. Se existir, será possível mostrar a informação de todos os bancos, mas caso não exista.

- Deve aparecer a mensagem: “Desculpe, não foi possível encontrar nenhum banco cadastrado. ”
- DICA:** Você pode usar o `unset($lista)` para limpar a lista e validar, se de fato a mensagem aparece.

4. Printe em tela a quantidade total de bancos cadastrados na sua lista.

5. Caso você não tenha separado os arquivos por responsabilidades faça isso a fim de aprimorar a legibilidade e organização do seu código. Seguem os critérios que podem ajudar na refatoração:

- a. É legal que sua lista de banco esteja em um arquivo separado, como se fosse um arquivo apenas para armazenar os dados.
- b. Caso você esteja usando muita concatenação de string ou repetindo muito um código em específico, talvez seja bacana criar um arquivo de funções úteis e criar uma função para solucionar o seu problema.

TÓPICO AVANÇADO:

Caso esteja com facilidade para desenvolver os desafios anteriores, então crie funções para **adicionar**, **editar**, **remover** e **listar** em um arquivo separado que recebam como referência a lista de bancos para realizar toda a manipulação da lista. Somente invoque estas funções no seu arquivo principal para realizar as operações.

4. Referências

Múltiplos autores: PHP: **Manual do PHP**, 2023. Disponível em: https://www.php.net/manual/pt_BR/. Acesso em: 22 de setembro de 2023.