**PHP 7**

**O que é?**

**Hypertext Preprocessor** ou PHP, é uma linguagem de programação focada na criação de aplicações do lado do servidor;

Open source- de uso livre, pode ser baixado gratuitamente em qualquer máquina;

Aplicações:

– Script de linha e comando (para automatização de tarefas), utilizado por exemplo no CMD do Windows;

- Aplicações desktop (PHP - GTK), aplicações que são instaladas no computador que pode ser interpretada com Client-Side;

- Aplicações Server-Side (Scripts), aplicações que rodam do lado do servidor, para criação de páginas Web dinâmicas.

É uma das linguagens de programação mais utilizadas para desenvolvimentos de sites e/ou sistemas Web dinâmicos;

Linguagem de programação interpretada (scripts não são submetidos a compilação prévia), os scripts serão interpretados em tempo de requisição das páginas;

Disponível em diversos sistemas operacionais (como Windows, Linus, OS X e Unix);

Inicialmente ele suportaria o paradigma estrutural, porém com o tempo, o paradigma de Orientação a Objetos foi incorporado. Hoje ele suporta não só a utilização, mas também, a combinação dos dois paradigmas;

Suporte a sistemas gerenciadores de banco de dados;

Suporte a protocolos de comunicações disponíveis na internet.

**Embutindo blocos PHP em páginas HTML**

**Nome Tag**

Tag padrão **<?php** código aqui**?>** Habilitada

Tag de impressão **<?=** código aqui**?>** Habilitada

Tag curta **<?** código aqui**?>** Desabilitada

\*Asp tag **<%** código aqui **%>**

Descontinuada na versão 7

Para utilizar códigos PHP, é necessário encapsular o código entre tags (que são específicas do PHP;

As tags podem ser utilizadas em qualquer local dentro do arquivo.

Na versão 8 a tag (<?...?>) não está mais disponível. Deve utilizar a tag padrão.

Config do PHP

XAMPP>Config>PHP (php.ini)

Localizar: XAMPP>Explorer>php> php.ini

Abre arquivo, CTRL+F para localizar ‘short\_open\_tag=Off’

Muda o ‘Off’ para ‘On’ e salva. Reinicia XAMPP (stop, start)

<?php

    echo 'Utilizando a tag padrão'

    ?>

    <br />

    <?= 'Utilizando a tag impressão' ?>

    <br />

    <? echo 'Utilizando a tag curta' ?>

Única tag que se diferencia é a de impressão que tem objetivo explícito

**Ponto e vírgula é obrigatório**, a não ser que tenha apenas uma instrução como no exemplo acima.

**Saída de dados ‘echo() e print()’**

Possuem objetivo claro – **impressão de conteúdo** do bloco de código **PHP dentro do** arquivo . Basicamente uma janela entre a programação e o arquivo HTML;

<hr/> e <br/> podem ser utilizados.

 <?php

    echo 'Comando echo<br/>'; //construtor da linguagem

    print "Comando print<hr/>"; //inicialmente era uma função (que pode ser utilizada normalmente)

?>

//quando função print é executada corretamente, retorna um, quando há erro, retorna 0

    echo print("comando print<hr/>")//result=1

**Funcionamento do PHP mais a fundo (páginas estáticas x dinâmicas)**

De um lado existe o cliente, onde é executado aplicações:

Browsers ou Navegadores – Chrome, FireFox, Edge, Safari e etc.

Essas aplicações fazem requisições através de um meio físico, a Internet.

A requisição tem que sair do computador e ir até uma outra ponta, que pode ser inclusive o computador se for utilizado o **localhost**.

Na outra ponta, deve existir um servidor HTTP (no nosso caso esse servidor é o Apache), capaz de receber e tratar dessas requisições.

A requisição é feita para um arquivo **.php** em que do lado do servidor haverá uma linguagem de programação que vai atuar sobre essa requisição que está sendo feita para o servidor.

Páginas estáticas: Arquivos HTML com codificação HardCoded, onde o servidor HTTP recupera essa requisição e devolve um arquivo;

Páginas dinâmicas: São produzidas no intervalo entre a requisição e a resposta que é recebida e devolvida pelo servidor HTTP (Apache, nesse caso).

**Comentários**

Usados principalmente para documentar o código.

Quando o código está sendo criado, tudo que está sendo feito fica fresco em nossa mente, logo, não há muita dificuldade para entender os blocos de código. Mas, quando depois de um tempo voltamos para fazer manutenção nesses mesmos códigos, é comum ter dificuldade na interpretação da codificação.

A uma confusão até que o entendimento do código seja estabelecido novamente. Principalmente quando há uma certa complexidade no código.

Com isso, comentar o código se torna extremamente útil para fins de documentação. Auxiliando na manutenção futura do código.

No PHP há três formas de fazer comentário nos códigos:

***Comentário de 1 linha***

    #Comentário de uma linha

***Comentário de 1 linha no estilo shell***

    #Comentário de uma linha

***Comentário de múltiplas linhas***

/\*

            Comentário de múltiplas linhas.

            Este tipo de comentário permite que várias linhas

            sejam comentadas ao mesmo tempo

        \*/

**Variáveis**

***Introdução***

**O que são?**

Utilizadas para armazenamento temporário de dados.

**Tipos de variáveis em PHP**

***string*** - armazenam textos, cadeias de caracteres;

***int*** – armazenam valores numéricos inteiros (positivos e negativos);

ex.: 2, 5, -10

***float*** – armazenam valores numéricos (positivos e negativos), mas, com fração;

ex.: 10.5, -7.4

***boolean*** – armazenam estados (true ou false), podem ser utilizados em testes lógicos;

***array*** – como um mapa ordenado de variáveis, relacionados através de chaves e valores.

**Regras de declaração de variáveis em PHP**

Deve obrigatoriamente iniciar com o caractere **‘$’**;

Não pode conter espaços ou caracteres especiais (com exceção do underline);

Não requer, nem suporta a definição explícita de tipos, tipagem dinâmica;

Case sensitive, faz distinção entre letras minúsculas e maiúsculas.

***$nome $Nome $NOME***

**Exemplos de variáveis válidas e inválidas**

**$nome $fone1 $endereco\_2 $\_nome1**

**$ idade $1fone $número $\* nome\_**

Caracteres numéricos não podem estar no início da palavra.

//string

    $nome = 'Ana Almeida'; //podem ser utilizadas aspas simples ou duplas

    //int

    $idade = 17;

    $estudos = 'Desenvolvimento de sistemas';

    //float

    $peso = 50.5;

    //boolean - simples, armazena estado de true ou false(muito utilizadas em testes lógicos)

    $fumante\_sn = true;//(true=1) ou false = vazio

**Concatenação**

//operador '.'

    echo 'Olá ' . $nome . ', vi que sua cor preferida é ' . $cor . ', que você possui ' . $idade . ' anos e que você gosta de ' . $hobby;

    //aspas duplas - variáveis ficam dentro das aspas

    echo '<br/>';

    echo "Olá $nome, vi que sua cor preferida é $cor, que você possui $idade anos e que você gosta de $hobby"

**Variáveis constantes**

Variáveis que valores não podem ser alterados.

//definindo variáveis constantes

    define('BD\_URL', 'endereco\_bd\_dev'); //essa função precisa ser passada com dois valores(nomedavariavel, valordavariavel)

    //usar caracteres maiúsculos é uma boa prática

echo BD\_URL . '<br/>';

**If/else - Introdução**

É uma estrutura de controle;

Nos possibilita com base em uma determinada condição, definir qual fluxo a aplicação vai tomar;

Através dele que podem ser feitas verificações para criação de fluxos dentro das nossas aplicações;

Verificação que atendam a todos os tipos de possibilidades, pelo menos, todas que puderem ser previstas.

**Operadores de comparação (condicionais)**

Servem para formar as expressões condicionais para o comando if.

***Operadores:***

Igual (**==**);

Idêntico (**===**) – verifica valor e tipo;

Diferente (**!=**);

Diferente (**<>**);

Não idêntico (**!==**) – verifica se valores são diferentes e de **tipos diferentes;**

Menor (**<**);

Maior (**>**);

Menor igual (**<=**);

Maior ou igual (**>=**).

if (1 >= 2) {

        echo 'Verdadeiro';

    } else {

        echo 'Falso';

    }

    //identico e nao identico verifica valores e tipos

***Operadores lógicos***

Servem para tornar comparação de condições mais complexas, maiores.

***Operadores:***

E (**AND** ou **&&**)- true se todas as operações são verdadeiras;

OU (**OR** ou **||**)- true se pelo menos uma operação é verdadeira;

Xor (**XOR**)– true se apenas uma das operações é verdadeira;

Negação (**!**)– inverter resultado da expressão.

//Operadores lógicos

    //E: && ou AND -> retorna verdadeiro se todas as expressões forem verdadeiros

    //f e f = f

    //f e v = f - v e f = f

    //v e v = v

    //OU: || ou OR -> retorna verdadeiro se pelo menos uma das expressões é verdadeiras

    //f ou f = f

    //v ou f = v - f ou v = v

    //v ou v = v

    //XOR: XOR -> retorna verdadeiro se apenas uma das expressões for verdadeira

    //f xor f = f - v xor v = f

    //v xor f = v - f xor v = v

    //! - inverte o resultado informado pela expressão

    //v xor v = !(f) = v

    //v xor f = !(v) = f

    //v e v e f = !(f) = v

    //estabelecer precedência - interpretador do php lê primeiro o que há entre parenteses

    //(v e v) = v ou f = v

***Praticando If/else e operadores***

 <?php

$usuariopossui\_cartaoloja = true;

    $valor\_compra = 225;

    $valor\_frete = 50;

    $recebeu\_desconto\_frete = false;

    if ($usuariopossui\_cartaoloja && $valor\_compra >= 100) {

        $valor\_frete = 0;

        $recebeu\_desconto\_frete = true;

    }

    ?>

<h1>Detalhes do pedido</h1>

    <p>Possui cartão da loja?

        <?php

        if ($usuariopossui\_cartaoloja) {

            echo 'SIM';

        } else {

            echo 'NÃO';

        }

        ?>

    </p>

***Condições encadeadas***

- Tem mais de duas condições if – else if

$recebeu\_desconto\_frete = true;

if ($usuariopossui\_cartaoloja && $valor\_compra >= 400) {

        $valor\_frete = 0;

    } else if ($usuariopossui\_cartaoloja && $valor\_compra >= 300) {

        $valor\_frete = 10;

    } else if ($usuariopossui\_cartaoloja && $valor\_compra >= 100) {

        $valor\_frete = 25;

    } else {

        $recebeu\_desconto\_frete = false;

    }

**Operador Ternário**

$valor\_frete\_aux = $usuariopossui\_cartaoloja && $valor\_compra >= 400 ? 0 : ($usuariopossui\_cartaoloja >= 300 ? 10 : ($usuariopossui\_cartaoloja && $valor\_compra >= 100 ? 25 : $valor\_frete));

    $recebeu\_desconto\_frete = $valor\_frete != $valor\_frete\_aux ? true : false;

    $valor\_frete = $valor\_frete\_aux;

**Switch**

Parece com o if mas é mais a sintaxe é mais simples.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*explicação do método:\*/

        var opcao = 2

        switch(opcao){ //opcao  -->variavel ou valor pré-definido

            case 1:/\* -- > função - comparar o valor encaminhado com o switch com o valor do case

                     -- > comparação feita utilizando o operador idêntico\*/

                //trecho de código que será executado

                break//precisa ser utilizado no final de cada

            case 2:

                //trecho de código que será executado

                break

            default://funciona como um else

                //trecho de código que será executado

                break

        }/\* -->nesse exemplo, o case que será disparado, será o 'case 2', por conta da comparação que será feita com a variável(nesse caso)'opcao'

        /

Default é opcional;

Precisa haver um break ao final de toda função;

Break é importante para interromper funções de um determinado case. Pois programa roda até achar um break;

Se nenhuma condição for satisfeita, loop entra no default;

Os cases podem ser de qualquer tipo.

**Casting de tipos com (int), (bool), (float) e (string)**

Float e double são as mesmas coisas;

Função ***gettype*** ajuda no debug;

Quando é convertida uma fração do tipo float para int, preserva apenas a parte inteira

//gettype() => retorna o tipo da variável -> importante nesse momento para debug

    //$valor = 10;

    //$valor2 = (float) $valor;//float, double

    //$valor2 = (string) $valor;

    $valor = true;

    //$valor2=(int)$valor;//preserva apenas parte inteira

    //$valor2=(string)$valor;//preserva apenas parte inteira

    $valor2 = (string) $valor; //boolean ou bool -  boolean para int retorna um ou zero

    //fato da variável representar um valor válido retorna true na conversão para bool- caso false não retorna nada

    $valor3=(int)$valor2;

**Operações aritméticas**

Operadores matemáticos que nos permitem efetuar cálculos.

Operadores

Adição (**+**)

Subtração (**-**)

Multiplicação (**\***)

Divisão (**/**)

Módulo (**%**)

Módulo só retorna número inteiro.

echo "A módulo entre $num1 e $num2, é: ". ($num1%$num2);

        echo'<br/>'

//=

    //+,-,\*,/,%

    $x = 10;

    $x = $x + 5;

    $x%=50;

    echo $x;

**Operadores de incremento e decremento**

É acrescentado(incremento) ou retirado(decremento) uma unidade do valor atual.

Pré-incremento (**++$a**)

$a = 7;

    echo "O valor contido em a é $a <br/>";

    echo 'O valor contido em a previamente do incremento, é '. ++$a. '<br/>'; //incremento é feito antes da interpretação da instrução//result:8

    echo "O valor atualizado é $a"; //result:8

Pós- incremento (**$a++**)

$a = 7;

    echo "O valor contido em a é $a <br/>";

    echo 'O valor contido em a após o incremento, é '. $a++. '<br/>'; //neste momento o valor ainda é o inicial, pois o incremento é feito após a interpretação da instrução

    echo "O valor atualizado é $a"; //valor=8

Pré- decremento (**--$a**)

$a = 7;

    echo "O valor contido em a é $a <br/>";

    echo 'O valor contido em a previamente do decremento, é '. --$a. '<br/>'; //decremento é feito antes da interpretação da instrução//result:6

    echo "O valor atualizado é $a"; //result:6

Pós-decremento (**$a--**)

$a = 7;

    echo "O valor contido em a é $a <br/>";

    echo 'O valor contido em a previamente do decremento, é '. --$a. '<br/>'; //decremento é feito antes da interpretação da instrução//result:6

    echo "O valor atualizado é $a"; //result:6

**Funções**

***Introdução***

Encapsulamento de um bloco de códigos com um objetivo definido;

Onde se escreve uma lógica que se torna reutilizável; gastando assim, menos linhas de código;

Caso tenha que ser feita alguma mudança na função, basta mudar o local onde a função foi criada e todos os pontos onde ela está sendo chamada, contam com essa atualização.

Declaração de uma função:

//void - pois não tem retorno

function exibirBoasVindas ()

    {

        echo = “Bem-vindo ao curso de PHP”;

    }

//return(com retorno)

function calcularAreaTerreno ($largura, $comprimento)

    {

        $area = $largura \* $comprimento;

        return $area;

    }

echo calcularAreaTerreno (2,3)//como está sendo feita uma multiplicação, valores precisam ser passados obrigatoriamente

Palavra function;

Nome da função. Padrão:

(1ºverbo, 2ºsubstantivo),

Começa com letra minúscula e se tiver mais palavras começa com letra maiúscula;

Parâmetros recebidos da função (entre parênteses) - se tiver;

Entre as chaves fica o bloco de código da função.

Parâmetros funcionam como entrada de dados da função;

Funções são como subprogramas;

Podem ser de dois tipos diferentes: void (quando há apenas o processamento de alguma lógica) ou do tipo com retorno (além de processar alguma lógica, retorna um valor para script que efetuou a chamada da função).

$resultado = calcularAreaTerreno (20, 10);

    echo $resultado

Maneira de exibir valor retornado.

**Funções nativas para manipular strings**

strtolower (**$texto**) -> transforma todos os caracteres da string em minúsculos;

strtoupper (**$texto**) -> transforma todos os caracteres da string em maiúsculos;

ucfirst (**$texto**) -> transforma o primeiro caractere da string em maiúsculo;

strlen (**$texto**) -> conta quantidade de caracteres de uma string;

str\_replace (**<procura por>**, **<substitui por>**, **$texto**) -> substitui uma cadeia de caracteres por outra dentro de uma string;

substr (**$texto**, **<posicao inicial>**, **<qtde caracteres>**) -> retorna parte de uma string.

//string to lower

echo strtolower($texto);

//string to upper

echo strtoupper($texto);

//upper case first

echo ucfirst($texto);

//string length

echo strlen($texto);

//string replace

echo str\_replace ('PHP', 'JavaScript', $texto); //case sensitive, a palavra tem que ser idêntica a da string da variável

//substr replace

echo substr ($texto, 6, 8); //variavel, posição inicial, qtd de caracteres a sere, resgatados

**Funções nativas para tarefas matemáticas**

ceil (**$numero**) -> arredonda o valor para cima;

$num = 7.3;

    echo ceil($num) . '<br/>'; //arredonda para cima

    //8

floor (**$numero**) -> arredonda o valor para baixo;

    $num = 7.3;

    echo floor($num) . '<br/>'; //arredonda para baixo

    //7

round (**$numero**) -> arredonda o valor com base nas casas decimais;

    $num = 7.5;

    echo round($num) . '<br/>'; //arredonda com base na fração / .0 e .4 => baixo | .5 => cima

    //8

rand () -> gera um inteiro aleatório;

 echo rand(10, 20) . '<br/>'; //gera valor aleatório de 0 até randmax(maior valor aleatório possível onde a aplicação está rodando)

    //quando coloca dois números, é gerado um valor aleatório entre eles

    echo getrandmax() . '<br/>';

sqrt (**$numero**) -> retorna a raiz quadrada.

echo sqrt (25)

**Funções nativas para manipular datas**

date (**formato**) -> recuperar data e hora atual do sistema;

date\_default\_timezone\_get (**timezone**) -> recuperar o timezone default da aplicação;

date\_default\_timezone\_set (**timezone**) -> atualizar o timezone default da aplicação;

strtotime (data) -> transformar dados textuais em segundos.

<?php

    /\*

    //recuperação da data atual / data corrente

    echo date('d/m/Y H:i');

    //existem tokens para cada coisa, que podem ser combinados, como no exemplo acima

    //o ideal seria sequência da data invertida(padrão dos EUA)

    //

    echo '<br/>';

    echo date\_default\_timezone\_get ();//onde a aplicação PHP está sendo executada, se tiver acesso aonde o php está instalado, o ideal é ir até o doc.ini, pesquisar por date.timezone=... e modificar isso

    date\_default\_timezone\_set ('America/Sao\_Paulo');//caso não tenha acesso a instalação do PHP

    echo '<br/>'

    \*/

    //quando se trabalha com datas em programação, se trabalha com essas datas em formato americano(ano-mês-dia) --- importante para padrões computacionais

    $data\_inicial = '2018-04-24';

    $data\_final = '2018-05-15';

    //timestamp - representação de tempo dessa data

    //01/01/1970

    $time\_inicial = strtotime($data\_inicial); //função espera data no formato americano, por isso é importante

    $time\_final = strtotime($data\_final);

    //retorno das datas em segundos, em relação a 01/01/1970

    echo $data\_inicial . '-' . $time\_inicial;

    echo '<br/>';

    echo $data\_final . '-' . $time\_final;

    $diferenca\_times = abs($time\_inicial - $time\_final); //abs - inverte resultado negativo para positivo(valor absolute)

    //ao invés de se usar abs, pode-se também efetuar o cálculo ao contrário; time\_final menos(-) time\_inicial

    echo '<br/>';

    echo "A diferença de segundos entre a data inicial e final é $diferenca\_times <br/> "; //diferenca de segundos entre as duas datas

    $segundos\_existem\_dia = 24 \* 60 \* 60;//horas(dia)\*minutos(hora)\*segundos(minuto)

    echo "Um dia possui $segundos\_existem\_dia segundos";//quantos segundos existem em um dia

    $diferencaentredatas = $diferenca\_times / $segundos\_existem\_dia;//calculo de diferença de dias entre as datas

    echo '<br/>';

    echo "A diferença em dias é: $diferencaentredatas";

    ?>

**Array básico**

Geralmente usados para fazer listas

***Array sequencial (numérico)***

Com variáveis

$variavel\_1 = 'Banana';

        $variavel\_2 = 'Maçã';

        $variavel\_3 = 'Morango';

        $variavel\_4 = 'Uva';

Com um único array

$lista\_frutas[1] = 'Banana';

        $lista\_frutas[2] = 'Maçã';

        $lista\_frutas[3] = 'Morango';

        $lista\_frutas[4] = 'Uva';

Sequência numérica do índice parte de 0;

Recupera-se valor de array com o número do índice.

//sequenciais (numéricos)

    //$lista\_frutas = array('Banana', 'Maçã', 'Morango', 'Uva', 'Abacate');

    $lista\_frutas = ['Banana', 'Maçã', 'Morango', 'Uva', 'Abacate'];

    $lista\_frutas[] = 'Abacaxi';

    echo '<pre>';

    var\_dump($lista\_frutas);

    echo '</pre>';

    echo '<hr/>';

    echo '<pre>';

    print\_r($lista\_frutas);

    echo '</pre>';

    echo $lista\_frutas[2];

***Array associativo***

Índice associativo;

‘<índice>’=>’<elemento-array>’

    //associativos

    $lista\_frutas = ['a' => 'Banana', 'b' => 'Maçã', 'x' => 'Morango', 'z' => 'Uva', '2' => 'Abacate'];

    $lista\_frutas['w'] = 'Abacaxi';

    echo '<pre>'; //melhor organização visual do array

    var\_dump($lista\_frutas);

    echo '</pre>';

    echo $lista\_frutas['w']

**Array multidimensional**

Array de array’s;

Podem ser encadeados quantas vezes for necessário.

$lista\_coisas = [];

    $lista\_coisas['frutas'] = array('Banana', 'Maçã', 'Morango', 'Uva');

    $lista\_coisas['pessoas'] = [1 => 'João', 2 => 'José', 3 => 'Maria']; - ajustando índices

    print\_r($lista\_coisas); - recuperando array’s

    echo $lista\_coisas['frutas'][3];

**Array – método de pesquisa**

//in\_array() - true ou false para a existência do que está sendo procurado

    //array\_search() - retorna o índice do valor pesquisado, caso ele exista

$existe = in\_array('Maçã', $lista\_frutas);

    //true: 1 -> false: vazio

    if ($existe) { //true / false

$existee = array\_search('Uva', $lista\_frutas);

    //se ñ existe -> null - vazio

    if ($existee != null) {

**método no array multidimensional**

  echo in\_array('Uva', $lista\_coisas['frutas']);

    echo in\_array('Maria', $lista\_coisas['pessoas']);

**False, null e empty**

 //false (true/false) - tipo variável boolean

    //null e empty - valores especiais

//valores null

    if (is\_null($funcionario1)) { //função nativa do php

//valores empty

    if (empty($funcionario1)) { //função nativa do php

 //uma variável está vazia quando de fato não tem nenhum valor ou quando está com valor null

    //uma variável é null, quando ela tiver aquele valor atribuído a ela

**Funções nativas para manipular arrays**

is\_array (**array**) -> verifica se parâmetro passado é um array (true or false);

***true:***

***$array = [];***

***$retorno = is\_array($array);***

array\_keys (**array**) – retorna chaves do array;

***$array = [1 => 'a', 7 => 'b', 18 => 'c'];***

*retorna elementos (****a, b, c****):*

***echo '<pre>'print\_r($array); echo '</pre>';***

*retorna índice/chaves (****1, 7, 18****):*

***$chaves\_array = array\_keys($array);***

***echo '<pre>'print\_r($chaves\_array); echo '</pre>';***

sort (**array**) -> ordena array (coloca em sequência alfabética) e reajusta índice;

***sort($array) //true ou false - retorna true ou false se o array foi ou não ordenado***

asort (**array**) -> ordena array preservando índice;

***asort($array);//true ou false - retorna true ou false se o array foi ou não ordenado, preservando índice***

count (**array**) -> conta quantidade de elementos de um array;

***echo count($array);***

array\_merge (**array**) -> funde um ou mais arrays;

***$array1 = ['osx', 'windows'];***

***$array2 = ['linux'];***

***$novo\_array = array\_merge ($array1, $array2);***

explode (**array**) -> divide uma string baseada em um delimitador;

***$string = '07/12/2017';***

***$array\_retorno = explode ('/', $string);***

*Formatando para data no estilo dos EUA:*

***echo $array\_retorno[2] . '-' . $array\_retorno[1] . '-' . $array\_retorno[0];***

implode (**array**) -> junta elementos de um array em uma string;

    $array = ['a', 'b', 'x', 'y']; //a,b,x,y

    $string\_retorno = implode(',', $array); - separador + array que contém os elementos

**Loops**

Permitem que façamos a repetição de um comando ou de um bloco de códigos até atender a uma determinada condição;

Pode ser feito um bloco de códigos para ser utilizado o mesmo loop para diversas partes do código;

Os laços esperam um critério de parada (instrução que indique que a repetição do comando ou bloco de códigos deve ser interrompida);

A ausência de um critério de parada pode resultar em um loop infinito, resultando no travamento da aplicação.

$num = 1;

    //operadores comparação / lógicos

    while ($num < 1000) { //inicia com a palavra while, seguida de uma condição entre parênteses, seguida de comandos a serem repetidos com base na condição

        $num++;

        if ($num == 2 || $num == 6) {

            continue; //tudo oque tiver até o fechamento do bloco de repetição é descartado e passa pro próxima interação

        }

        echo "$num <br/>";

        /\*if($num>100){

            break;//assim como no switch, interrompe execução do laço

        }

        \*/

        //enquanto break interrompe completamente a estrutura da repetição, o continue pula uma interação, indo para a seguinte, descartando o que tiver a seguir da interação continue

    } //repitir codificação enquanto num for <10

**Do while**

Condição vem após o bloco de código que será executado (contrário do while).

**do {**

**echo "X = $x <br/>";**

**$x++; //precisa ter um critério de parada para não ocasionar em um loop infinito**

**//continue**

**//break**

**} while ($x < 9);**

**For**

for ($x = 10; $x > 0; $x--) { //ordem é fundamental

        //break

        //continue

        echo "$x <br/>";

    }

Parâmetros: (variável; condição; incremento/decremento)

***for (variável; condição; incremento/decremento) {}***

**Percorrendo arrays (*while*, *do while* e *for*)**

**//WHILE**

**while ($idx < count($registros)) {**

**echo '<h3>' . $registros[$idx]['título'] . '</h3>'; //'tíulo' é um índice associativo que vem do array de registros**

**echo '<p>' . $registros[$idx]['conteúdo'] . '</p>'; //'conteúdo' é um índice associativo que vem do array de registros**

**echo '<hr/>';**

**$idx++;**

**}**

**//DO WHILE**

**do {**

**echo '<h3>' . $registros[$idx]['título'] . '</h3>'; //'tíulo' é um índice associativo que vem do array de registros**

**echo '<p>' . $registros[$idx]['conteúdo'] . '</p>'; //'conteúdo' é um índice associativo que vem do array de registros**

**echo '<hr/>';**

**$idx++;**

**} while ($idx < count($registros));**

**//FOR**

**for ($idx = 0; $idx < count($registros); $idx++) {**

**echo '<h3>' . $registros[$idx]['título'] . '</h3>'; //'tíulo' é um índice associativo que vem do array de registros**

**echo '<p>' . $registros[$idx]['conteúdo'] . '</p>'; //'conteúdo' é um índice associativo que vem do array de registros**

**echo '<hr/>';**

**}**

**Foreach**

foreach () {

}

Comando especializado para arrays e objetos;

Mais inteligente do que outros laços no quesito de lidar com array e objeto;

O controle de finalização das interações já existe no comando;

Array é passado e automaticamente ele finaliza as interações quando chega ao final desse array passado, para quando não há elementos dentro do array a serem percorridos.

***foreach ($itens as $item) {//chaves encapsulam o conteúdo que será executado para cada um dos elementos dentro do array***

***//array + as + definicao de variável que receberá valor contido em cada item do array***

***echo "$item ";***

***if ($item == 'mesa') { - verifica se variável criada pelo foreach con***

***echo ' (\*Compre uma mesa e ganhe 25% de desconto na compra de 4 cadeiras)';***

***}***

***echo '<br/>';***

***}***

***Exercício fixação***

Criar script com laço de repetição, que produza um array com 6 elementos (números aleatórios de 1 a 60);

Simulando sorteio da mega sena.

***<?php***

***$elementos = []; //array onde serão incluídos os números***

***while (count($elementos) <= 5) { //enquanto 'elementos' for menor ou igual a 5***

***$aleatorio = rand(1, 60); //variável 'aleatorio' gera um número aleatório entre 1 e 60***

***if (!in\_array($aleatorio, $elementos)) { //verificando se número aleatório está no array('!'se não estiver..)***

***$elementos[] = $aleatorio; //executa a adição do mesmo no array***

***}***

***}***

***print\_r($elementos); //exibindo elementos***

***?>***

**App Help Desk**

***Introdução***

Objetivo: colocar em prática conhecimentos adquiridos nessa parte do curso;

Será falado sobre segurança na parte de backend;

Poderemos compreender melhor o tráfego de informações (Request e Response);

Criando aplicações web mais dinâmicas, mais próximas da realidade e do que será visto no dia a dia;

Recursos que estão sendo utilizados no layout é apenas CSS e HTML combinados com bootstrap.

Em xampp > Config > Apache (**httpd.conf**) > Ctrl+f(**DirectoryIndex**), podemos ver os nomes dos arquivos que são prioridade na execução. Dois deles são:

**index.php; home.php.**

Nada impede que adicionemos algum nome lá (adiciona e reinicia (liga e desliga) Apache);

Se não tiver nenhum arquivo com algum desses nomes, é executada uma tela com as pastas e arquivos que há dentro do projeto.

**Formulários (métodos GET e POST)**

**Capturar dados** em um formulário e **encaminhá-los** (através da internet) **para o servidor HTTP**. Essa informação é recebida, ele processa algo e retorna alguma informação (**Request e Response**);

Entre a requisição e a resposta podemos anexar informações.

***Index.php***

<form action="valida\_login.php">

O action é o destino do submit de um form;

Quando submit é clicado, o próprio browser encapsula informações do formulário e dispara elas através de uma requisição HTTP para o servidor;

O destino que foi colocado é a validação de login em php.

<input name="email" type="email"

Adiciona atributo name à email e senha, levando os dados recebidos para a url; fazendo com que sendo possível resgatá-los e manipulá-los dentro do arquivo .php que guarda o código de validação.

***valida\_login.php***

***print\_r($\_GET); //get - resgata dados***

***echo '<br/>';***

***$\_GET['email'];***

***echo '<br/>';***

***$\_GET['senha'];***

Resultado visto no browser é uma resposta HTML;

Proveniente da interpretação do script, destinada para o Apache que devolve informação para o browser.

//index.php

<form action="valida\_login.php" method="post">

//valida\_login.php

print\_r($\_POST); //POST - posta dados (usuário consegue visualizar através deste método)

**Autenticação do usuário**

Recebe dados do front-end;

Envia dados para Back-end;

***Verifica se usuário existe ou não dentro da aplicação.***

***foreach ($usuarios\_app as $user) {***

***$user['email'];***

***$user['senha'];***

***if ($user['email'] == $\_POST['email'] && $user['senha'] == $\_POST['senha']) { //verifica se senha e email declaras na variável corresponde as que foram passadas no formulário***

***$usuarios\_autenticado = true; //fica true pois não há a necessidade da autenticação do usuário não é necessária***

***}***

***}***

***if ($usuarios\_autenticado) {***

***echo 'Usuário autenticado';***

***} else {***

***header('Location: index.php?login=erro'); //haverá um encaminhamento do usuário; destino: página inicial***

***} //login=erro é um parâmetro que será passado para a URL***

***Index.php***

***<div class="form-group">***

***<input name="senha" type="password" class="form-control" placeholder="Senha">***

***</div>***

***<?php***

***if (isset($\_GET['login']) && $\_GET['login'] == 'erro') { //caso a variável login com o valor erro estiver sendo setada, a seguinte mensagem é exibida?>***

***<div class="text-danger">Usuário ou senha inválido(s)</div>***

***<?php } ?>***

**Protegendo páginas restritas com SESSION**

**Sessão -**> faz com que uma determinada solicitação do browser a partir de um identificador único possa acessar uma sessão do lado do servidor.

Espaço da memória onde é possível que o servidor armazene informações que conecte a instância do lado do servidor com a instância do lado do cliente, criando uma ponte;

Essa identificação é transitada na requisição através de um cookie ou um URL. De modo que a instância do browser sempre que realizar uma requisição encaminhe essa informação para o servidor para que a linguagem de programação consiga recuperar essa informação e acessar aquele espaço de memória dedicado para aquela instância do browser em especial.

Criando o recurso de sessão podemos transitar informações em qualquer requisição feita pelo front-end para o back-end. Podendo ser utilizado dentro de qualquer script.

session\_start();//quando se trabalha com sessão é fundamental executar essa função, antes de qualquer instrução que emita uma saída de dados no navegador

//como comumente é feito pode-se colocar como primeira instrução no script, mas o importante é que ela fique antes de qualquer impressão de dados no navegador

//ao iniciar uma sessão, temos acesso a uma super global chamada 'session' - começa vazio, pois podemos em tempo de execução criar valores

$\_SESSION['x'] = 'Oi, sou um valor de sessão';

print\_r($\_SESSION); //print\_r pois é um array

echo '<hr/>';

essa variável poderá ser utilizada em outros lugares, e pode ser passado um novo índice e um novo valor para ela que poderá ser usado na index.php.

***index.php***

if (isset($\_GET['login']) && $\_GET['login'] == 'erro') {//caso a variável login com o valor erro estiver sendo setada, a seguinte mensagem é exibida?>

<div class="text-danger">Usuário ou senha inválido (s)</div>

<?php } ?>

<?php

if (isset($\_GET['login']) && $\_GET['login'] == 'erro2') {

?>

<div class="text-danger">Faça login antes de acessar as páginas protegidas</div>

<?php } ?>

verificando se usuário está logado para deixar restrito o acesso à páginas restritas

***home.php***

<?php

session\_start();

if (!isset($\_SESSION['autenticado']) || $\_SESSION['autenticado'] != 'SIM')

  ;

header('Location: index.php?login=erro2');

echo $\_SESSION['autenticado'];

?>

verificando questão de autenticação do usuário para acesso a home (página restrita).

**Incorporando scripts com include, include\_once, require e require\_once**

Evitar redundância dos códigos nas aplicações;

***menu.php***

INÍCIO | MINHA REDE | VAGAS

<hr />

***inicio.php***

<?php

include ("menu.php");

ou

include "menu.php";

?>

Passando conteúdo da página menu para as outras.

A diferença do **include** para o **require** é a maneira como cada um trata um potencial erro:

**include** – warning (aviso/alerta, não afeta a execução da aplicação);

**require** – fatal error (interrompe completamente o funcionamento do script).

**include\_once** – mesmo esquema de erros, porém, permite a inclusão de um script, uma única vez;

**require\_once** – mesmo comportamento na situação de erro, porém, permite a inclusão de um script uma única vez.

<?php

include("menu.php"); //warning

require "menu2.php"; //fatal error

?>

conteúdo da página (início)

<br />

<?php

include\_once("menu.php"); //warning

require\_once "menu2.php"; //fatal error

?>

Segunda tentativa de incluir script dará erro por conta do once, como é permitido apenas uma vez.

**require\_once** no projeto

***validador\_acesso.php***

<?php

session\_start();

if (!isset($\_SESSION['autenticado']) || $\_SESSION['autenticado'] != 'SIM') {

    header('Location: index.php?login=erro2');

}

?>

<?php require\_once "validador\_acesso.php" ?>

Se for feita a tentativa de acesso a página restrita, mensagem passada no validador é exibida e retorna para a página de login.

require\_once pois quando houver algum erro quero que exiba um fatal error, para aplicação ser parada de e ser feita a tentativa de resolução do erro e \_once porque é ideal que este o método de autenticação só precisa ser passado uma vez.

**Encerrando sessão (logoff)**

Variáveis de sessão (variáveis criadas no servidor) que ficam disponíveis para browser cliente

//remover índices do array de sessão

//unset(); - espera o array e o índice deve ser eliminado - não é exclusiva da super global session, serve para remover índices de qlqr array.

//destruir a variável de sessão completamente

//session\_destroy() - remove todos os índices contidos dentro da superglobal session

***logoff.php***

<?php

session\_start();

session\_destroy();

header('Location: index.php');

?>

Remove variáveis de sessão e redireciona para index. Evitando erros na hora de deslogar.

**Registrando chamados**

Através de uma req. HTTP **recuperar dados** do **formulário**;

**Encaminhar** para o servidor;

Esse envio pode ser feito através do método **GET** ou **POST**;

Do lado do servidor essas informações serão interrompidas (**GET** ou **POST** depende do método de envio dos dados);

Abrir, escrever e fechar um arquivo .txt;

De modo que a informação encaminhada de um formulário no front-end da aplicação rodando no browser de um cliente possa ser armazenada de forma digital do lado do servidor, ou seja;

Persistir essas informações para que seja possível utilizá-las depois.

***abrir\_chamado.php***

***<form method="post">***

Como o formulário é em campo descritivo (engloba diversos caracteres), sugestão – usar **post**;

                <form method="post" action="registra\_chamado.php">

Destino do form (aonde ele será manipulado).

                    <input name="titulo" type="text" class="form-control" placeholder="Título">

                    <select name="categoria" class="form-control">

                    <textarea name="descricao" class="form-control" rows="3"></textarea>

Adicionando name aos campos, para haver algum valor de associação para ser encaminhada no request (o que está sendo enviado para o back-end). Assim poderá ser feita a ação do método **POST**;

***registra\_chamado.php***

***$texto = $\_POST["titulo"] ."#". $\_POST["categoria"] ."#". $\_POST["descricao"];***

***echo $texto;***

***?>***

//abrindo arquivo

$arquivo = fopen('arquivo.hd', 'a'); //a - abre arquivo apenas para escrita

//escrevendo texto do arquivo

$texto = $titulo . "#" . $categoria . "#" . $descricao . PHP\_EOL; //php\_eol - fim de linha (para assim, quando forem adicionados outros textos haver a quebra de linha)

//exibindo texto do arquivo

fwrite($arquivo, $texto);

//fwrite - espera 2 parâmetros (referência do arquivo que foi aberto(arquivo.hd), oque será escrito dentro desse arquivo)

//fechando arquivo

fclose($arquivo);

header('Location: abrir\_chamado.php');//encaminhando para abrir\_chamado.php novamente

**Consultando chamados**

Diferentemente do que foi feito em registro de chamados, agora o arquivo.txt será aberto para ler ao invés de escrever;

Serão recuperados os registros e essa informação será processada retornando para o usuário uma página web dinâmica.

//feof(end of file) - percorre arquivo recuperando cada linha, até o fim do arquivo

Função nativa do PHP e é utilizada em situações que precisamos recuperar dados de um determinado arquivo;

O parâmetro passado é a referência do arquivo aberto(neste case $arquivo).

**Tela de consulta**

**<?php foreach ($chamados as $chamado) { ?>**

**<?php $chamado\_dados = explode('#', $chamado); //retorna array com os dados existentes em chamados que cada item foi nomeado como chamado**

**if (count($chamado\_dados) < 3) { //se houver menos que 3 elementos(titulo,categoria,descricao), que são os existentes dentro desse array. Estará faltando um deles.. logo:**

**continue; //não exibe, continua e segue para a impressão normalmente**

**}**

**?>**

**<div class="card mb-3 bg-light">**

**<div class="card-body">**

**<h5 class="card-title">**

**<?= $chamado\_dados[0] //recuperando dados do array de chamados, usando índice para trazer um por vez?>**

**</h5>**

**<h6 class="card-subtitle mb-2 text-muted">**

**<?= $chamado\_dados[1] ?>**

**</h6>**

**<p class="card-text">**

**<?= $chamado\_dados[2] ?>**

**</p>**

**</div>**

**</div>**

**<? } ?>**

**Aplicando controle de perfil do usuário**

Aplicação recebe mais usuários;

Terá que haver o controle de visibilidade dos chamados abertos na aplicação;

Perfis: **Administrativo** – quem tem perfil administrativo pode visualizar os chamados de qualquer usuário.

**Usuário** – quem é do perfil usuário, visualiza apenas os seus próprios chamados.

É importante adicionar ao chamado quem foi responsável por abrir aquele chamado;

De alguma forma chamado precisa ter alguma conexão com o usuário;

Precisa ser guardado um identificador no chamado, que indique quem foi o usuário que abriu;

Um identificador único será adicionado (ID), para não haver problemas caso seja modificado por exemplo o email do usuário.

**Atividade fixação**

Fazer a lógica de exibição de chamados diretamente no loop em que os registros são recuperados no arquivo.hd.

$registro = fgets($arquivo); //parâmetro encaminhado é a referência do arquivo que está aberto($arquivo)

  $registro\_dados = explode('#', $registro);

  if ($\_SESSION['perfil\_id'] == 2) {

    //só exibe chamado se perfil foi criado por usuário, que índice é 2

    if ($\_SESSION['id'] != $registro\_dados[0]) { //verificando se índice id(2) é diferente do usuário responsável por fazer cadastro

      continue; //desconsidera codificação a seguir, pois não é necessário

    } else {

      $chamados[] = $registro; //colocando cada registro no array de chamados no geral

    }

  } else {

    $chamados[] = $registro; //colocando cada registro no array de chamados no geral

  }

Código foi para a parte de cima (onde estava a codificação HTTP);

Foi adicionado um else. Se if não atender a essas condições, vai para o else que **exibe os chamados baseados no registro**.

**<div class="card-body">**

**<? foreach ($chamados as $chamado) { ?>**

**<?php**

**$chamado\_dados = explode('#', $chamado);**

**//não existe detalhes do chamado se ele não estiver completo**

**if (count($chamado\_dados) < 3) {**

**continue;}**

**?>**

**<div class="card mb-3 bg-light">**

**<div class="card-body">**

**<?= $chamado\_dados[1] //recuperando dados do array de chamados, usando índice para trazer um por vez?>**

**</h5>**

**<h6 class="card-subtitle mb-2 text-muted">**

**<?= $chamado\_dados[2] ?>**

**</h6>**

**<p class="card-text">**

**<?= $chamado\_dados[3] ?>**

**</p>**

**</p>**

**</div>**

**</div>**

**<?php } ?>**

No body do card, ainda existe o loop (foreach) que faz a contagem do chamado e se estiver tudo certo:

Onde estão sendo exibidos os chamados continua igual (exibe cada item baseado no dado do chamada\_dados);

**Segurança no back-end de aplicações web**

Evita que back-end da aplicação fique exposto (informações sigilosas possam ser acessadas de forma indevida);

**Cliente** -> *Request* > **internet** > *Response* <- **Servidor**

Quando um servidor HTTP está configurado para receber requisições de qualquer outro dispositivo conectado a internet e com base nas requisições recebidas fornecer respostas;

Ele também está configurado para acessar um determinado diretório público e manipular os scripts ou documentos contidos no diretório público de acordo com as requisições que foram feitas a partir de dispositivos na internet;

Tudo que está em um diretório público de um servidor HTTP está disponível para o mundo, não muito recomendado exatamente porque na maioria das aplicações ter scripts com regras de negócios sigilosas;

**Uma das vulnerabilidades do nosso app:**

***- Registros de chamados***

Esses registros estão localizados em arquivo.hd;

Esse arquivo está dentro de htdocs (diretório público do apache);

Mais especificamente dentro do diretório app\_help\_desk;

Pode ser acessado por qualquer computador conectado a internet;

Qualquer um pode fazer um acesso direto a por exemplo o arquivo.hd, basta passar no browser o servidor HTTP em questão e acessar o respectivo recurso (arquivo.hd).

Um outro exemplo seria o valida\_login.php;

Esse script guarda detalhes de acesso;

Detalhes esses que estão expostos no script de forma HardCode;

Por mais que há o redirecionamento para o script de login caso usuário seja inválido e caso haja a tentativa de acesso a uma página restrita;

Usuários mais avançados ou mal-intencionados poderiam utilizar alguma técnica para baixar o arquivo ao invés de interpretá-lo;

De modo que tivesse acesso ao script.

**Só fica no diretório público do servidor HTTP arquivos ou scripts que não são sigilosos.**

* Para maior entendimento, iremos resolver 2 problemas que é justamente o **arquivo.hd** e **valida\_login.php**;
* Arquivos que possuem informações sigilosas que não podem ser expostas para o mundo;
* Esses serão retirados do diretório público do servidor HTTP.

Em **C:/xampp/xampp/** foi criado um **novo diretório**: **app\_help\_desk**;

Este diretório está fora do diretório público que é htdocs;

Caminho para diretório público: **C:/xampp/xampp/htdocs/ app\_help\_desk**.

Caminho para diretório criado para arquivos e scripts sigilosos: **C:/xampp/xampp/app\_help\_desk**.