Centro Paula Souza
GOVERNO DO ESTADO SÃO PAULO

ADRIANO CILHOS DOIMO

PHP BÁSICO

INFORMAÇÃO E COMUNICAÇÃO

A expansão do Ensino Técnico no Brasil, fator importante para melhoria de nossos recursos humanos, é um dos pilares do desenvolvimento do País. Esse objetivo, dos governos estaduais e federal, visa à melhoria da competitividade de nossos produtos e serviços, vis-à-vis com os dos países com os quais mantemos relações comerciais.

Em São Paulo, nos últimos anos, o governo estadual tem investido de forma contínua na ampliação e melhoria da sua rede de escolas técnicas - Etecs e Classes Descentralizadas (fruto de parcerias com a Secretaria Estadual de Educação e com Prefeituras). Esse esforço fez com que, de agosto de 2008 a 2011, as matrículas do Ensino Técnico (concomitante, subsequente e integrado, presencial e a distância) evoluíssem de 92.578 para 162.105. Em 2016, no primeiro semestre, somam 186.619.

A garantia da boa qualidade da educação profissional desses milhares de jovens e de trabalhadores requer investimentos em reformas, instalações, laboratórios, material didático e, principalmente, atualização técnica e pedagógica de professores e gestores escolares.

A parceria do Governo Federal com o Estado de São Paulo, firmada por intermédio do Programa Brasil Profissionalizado, é um apoio significativo para que a oferta pública de Ensino Técnico em São Paulo cresça com a qualidade atual e possa contribuir para o desenvolvimento econômico e social do Estado e, consequentemente, do País.

Almério Melquíades de Araújo
Coordenador do Ensino Médio e Técnico

Centro Estadual de Educação Tecnológica Paula Souza

Diretora Superintendente
Laura Laganá

Vice-Diretor Superintendente
César Silva

Chefe de Gabinete da Superintendência
Luiz Carlos Quadrelli

REALIZAÇÃO
Unidade do Ensino Médio e Técnico

Coordenador
Almério Melquíades de Araújo

Centro de Capacitação Técnica, Pedagógica e de Gestão – Cetec Capacitações

Responsável
Lucília dos Anjos Felgueiras Guerra

Responsável Brasil Profissionalizado
Silvana Maria Brenha Ribeiro

Professores Coordenadores de Projetos
Carlos Eduardo Ribeiro
Fabricio Braoios Azevedo
Tiago Jesus de Souza

Autor
Adriano Cilhos Doimo

Projeto de formação continuada de professores da educação profissional do Programa Brasil Profissionalizado - Centro Paula Souza - Setec/MEC

PHP BÁSICO

Administração Central

Cetec Capacitações

Atualização em Programação para WEB

PHP Básico

Prof. Adriano Cilhos Doimo

Apresentação

Assuntos que serão abordados:

1 Introdução 5
1.1 Primeiros passos após a instalação 6
1.2 Editores PHP 10
1.3 Primeiro código 11
1.4 Múltiplos Blocos 13
1.5 Comentários no Código 13
2 Exibindo dados no navegador 14
3 Tipos de dados 16
3.1 Tipos de dados compostos 16
3.2 Type Casting 16
3.3 Tipo Juggling 17
3.4 Variáveis e Declaração 17
4 Escopo de variáveis 18
5 Variáveis Super Globais 21
6 Constantes 22
7 Expressões 22
7.1 Operadores 23
8 Estrutura de controle 23

8.1 Comandos de Seleção 24
8.1.1 Expressão if 24
8.1.2 Expressão if-else 24
8.1.3 Expressão elseif 25
8.1.4 Expressão switch 25
8.2 Comandos de Repetição 27
8.2.1 Expressão while 27
8.2.2 Expressão do..while 28
8.2.3 Expressão for 29
8.2.4 Expressão foreach 30
8.3 Outras expressões 31
8.3.1 Expressão break 31
8.3.2 Expressão goto 31
8.3.3 Expressão continue 32
9 Função 32
9.1 Função com argumento/parâmetro 33
9.2 Função com argumento e retorno 34
9.3 Biblioteca de funções 34
9.4 Formas de incluir arquivos 35
9.5 Expressão include condicionada 37
9.6 Expressão include_once 38
9.7 Expressão require 39
10 Array 40
10.1 Array associado 41
10.2 Função is_array() 41
10.3 Adicionando valores no vetor 42
10.3.1 Função array_unshift() 42
10.3.2 Função array_push() 43
10.4 Removendo valores do vetor 44
10.4.1 Função array_shift() 44
10.4.2 Função array_pop() 45
10.5 Localizando elementos no vetor 46

10.5.1 Função in_array() 46
10.5.2 Função array_key_exists() 46
10.5.3 Função array_search() 46
10.5.4 Função array_keys() 47
10.5.5 Função array_values() 47
10.6 Movendo o ponteiro de um array 48
10.7 Recuperando a chave de um ponteiro 48
10.8 Classificando arrays 49
10.8.1 Função array_reverse() 49
10.8.2 A função sort() 50
10.8.3 Função rsort() 50
10.8.4 Função natsort() 51
10.8.5 Função natcasesort() 51
11 Operações com array 52
11.1 Função array_merge() 52
11.2 Função array_combine() 53
11.3 Função array_slice() 53
11.4 Função array_intersect() 54
11.5 Função array_diff() 54
11.6 Função shuffle() 55
11.7 Função array_sum() 56
12 Array Multidimensional 56
13 Funções com strings 58
14 Funções de Data e Hora 59
15 Formulário 62
15.1 Propriedade action 62
15.2 Propriedade *name* 63
15.3 Elemento *input* e a propriedade *type* 64
15.4 Limitando a entrada de dados 66
15.5 Evento *onclick* 67
15.6 Botões fora do formulário 68
15.7 Caixa de seleção (select) 68

15.8 Elemento *textarea* 70
16 Formulário - Recuperando dados 71
16.1 Uso do $_POST[] 72
16.2 Uso do $_*GET*[] 80
16.3 Uso do $_*REQUEST*[] 81
17 Uso do $_*GET*[] com URL 82
17.1 Uso de variáveis para compor a URL 84
18 Sessão 86
18.1 Trabalhando com sessão 86
18.2 ID de uma sessão 87
18.3 Bloqueando páginas por sessão 87
18.4 Identificando uma sessão 89
19 Implementando os formulários 90

1 Introdução

O **PHP** (Hypertext Preprocessor) é uma linguagem interpretada gratuita, usada originalmente apenas para o desenvolvimento de aplicações presentes e atuantes no lado do servidor, capazes de gerar conteúdo dinâmico na web.

Uma página em PHP possui extensão **.php**. Podemos também encontrar nos arquivos php, conteúdo de código HTML. Quanto ao funcionamento, sempre que o servidor receber páginas com a extensão .php ele irá identificar a linguagem de programação na página e interpretar códigos HTML e PHP de qualquer maneira, uma vez que ambas são linguagens interpretadas e não compiladas.

Para funcionar, o php precisa estar rodando em um servidor devidamente configurado, que iremos comentar na sequência deste material. A maioria das hospedagens de sites possui o php configurado e pronto para ser usado. Para usar de forma local (em sua própria máquina), existem algumas opções de softwares para rodar as aplicações que iremos desenvolver neste material.

Dentre os programas que podemos utilizar, vamos citar apenas alguns mais comuns:

WampServer

http://www.wampserver.com/en/

USBWebserver 8

http://www.usbwebserver.net/en/

XAMPP

https://www.apachefriends.org/pt_br/index.html

No caso de nossas atividades, estaremos utilizando o XAMPP. Mas nada impede que utilizem outros softwares.

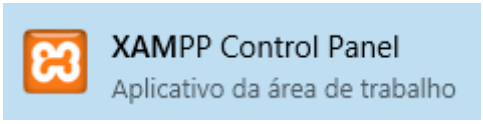
Após a instalação, devemos localizar o diretório padrão do serviço PHP que foi instalado, lembrando que cada tipo de servidor pode especificar um caminho diferente.

XAMPP = C:\xampp\htdocs
WAMP SERVER = C:\wamp\www
USBWebserver 8 = root

1.1 Primeiros passos após a instalação

Logo após a instalação (no caso vamos usar o XAMPP), devemos abrir o painel de controle do XAMPP para inicializar os serviços necessários para que o servidor funcione corretamente.

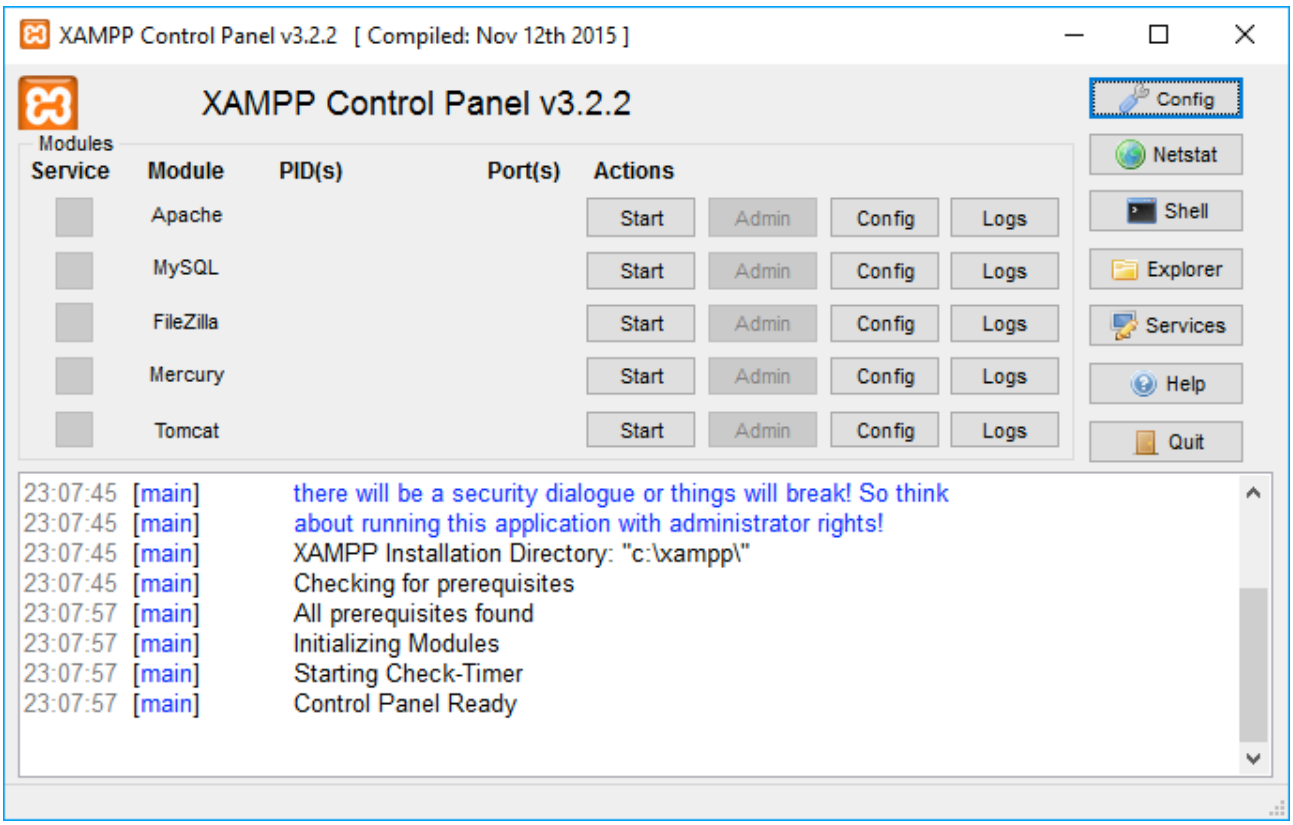
No caso deve ter um atalho na área de trabalho do seu computador:



Após dar um duplo clique, aparecerá a seguinte janela da aplicação:

Administração Central

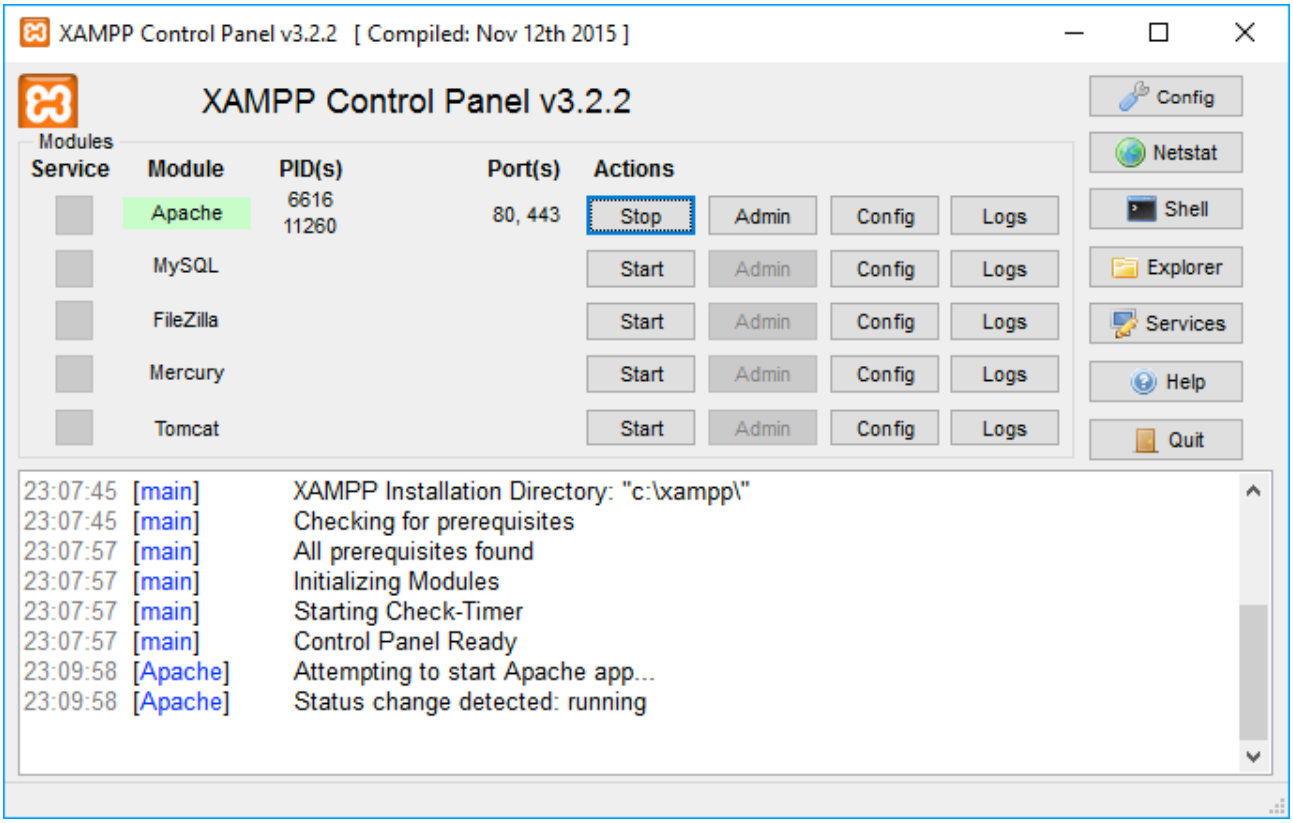
Cetec Capacitações



Agora devemos inicializar o serviço de aplicação, clicando em **start** do **Apache**. No caso após inicializar o servidor, conseguimos identificar as portas que está sendo utilizada para comunicação.

Administração Central

Cetec Capacitações



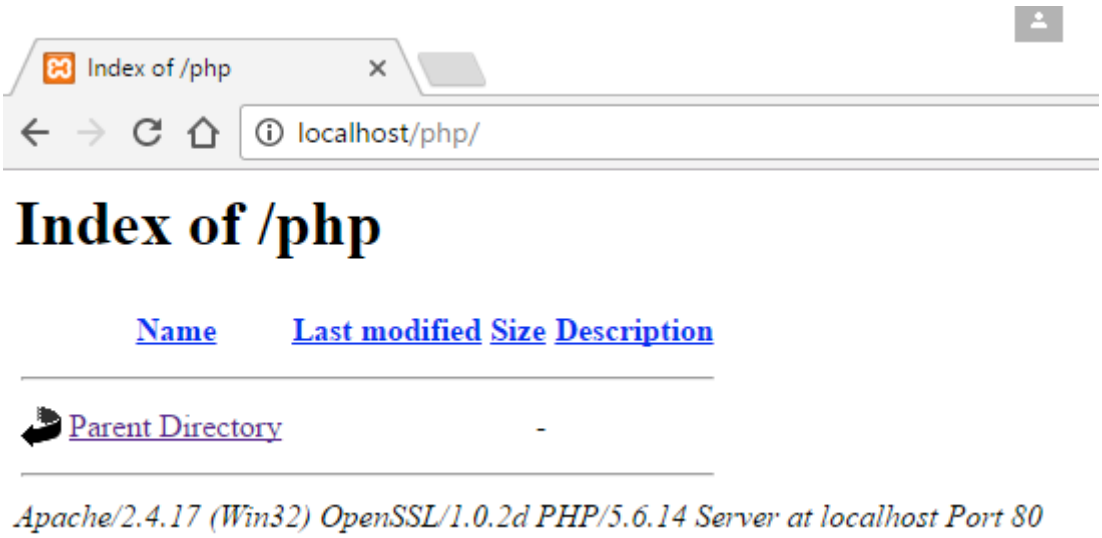
No navegador (qualquer um), com o serviço ativo do PHP, digite na barra de endereço ou URL a seguinte expressão "**localhost**" ou "**127.0.0.1**".

A resposta vai depender do conteúdo da **pasta padrão**, podendo aparecer à expressão "**Index of**", (nenhum arquivo) ou a tela de apresentação do serviço ativo.

Como sugestões para organização das atividades deste material:

- Exclua todos os arquivos na pasta padrão, ou crie outra pasta e copie todas as informações;
- Crie uma pasta para identificar suas atividades. Sendo assim, foi criada uma pasta chamada **php**, dentro do diretório **c:/xampp/htdocs**, para armazenar os arquivos de exemplo deste material.

Para acessar o conteúdo da pasta, clique diretamente sobre ela ou informe o caminho na URL, exemplo:



Importante:

1. Arquivos de inicialização (**index.php** ou **default.php**), são executados automaticamente;
2. Para outros nomes de arquivos devemos clicar sobre o nome ou indicá-lo diretamente na URL para serem executados;
3. Execute sempre a sua aplicação via localhost;

1.2 Editores PHP

Qualquer editor de texto é minimamente suficiente para escrever as linhas de códigos PHP. Para o desenvolvimento das atividades será utilizado o editor de texto **Notepad++** (https://notepad-plus-plus.org/). Mas fica a seu critério utilizar qualquer editor de texto para programação. Não podemos esquecer que todos os arquivos de preferência devem ser salvos com a extensão **.php**.

1.3 Primeiro código

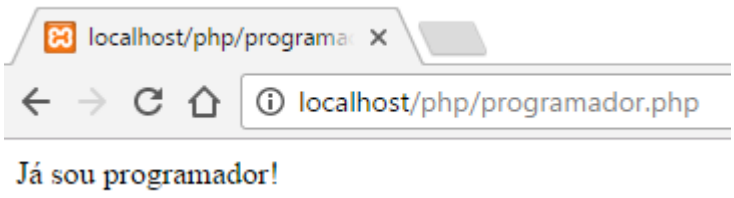
Para iniciarmos a codificação, vamos abrir o editor de texto, criar um arquivo e nomea-lo como **programador.php** e salve dentro da pasta padrão (c:/xampp/htdocs/php).

programador.php

```
1  <?php
2
3      echo "Já sou programador!";
4
5  ?>
```

Agora vamos abrir o navegador, digitar o caminho da pasta padrão e localize o arquivo que acabamos de criar. Encontrando-o, basta clicar sobre o arquivo **programador.php**.

O resultado que teremos no navegador assim que clicarmos no arquivo será:



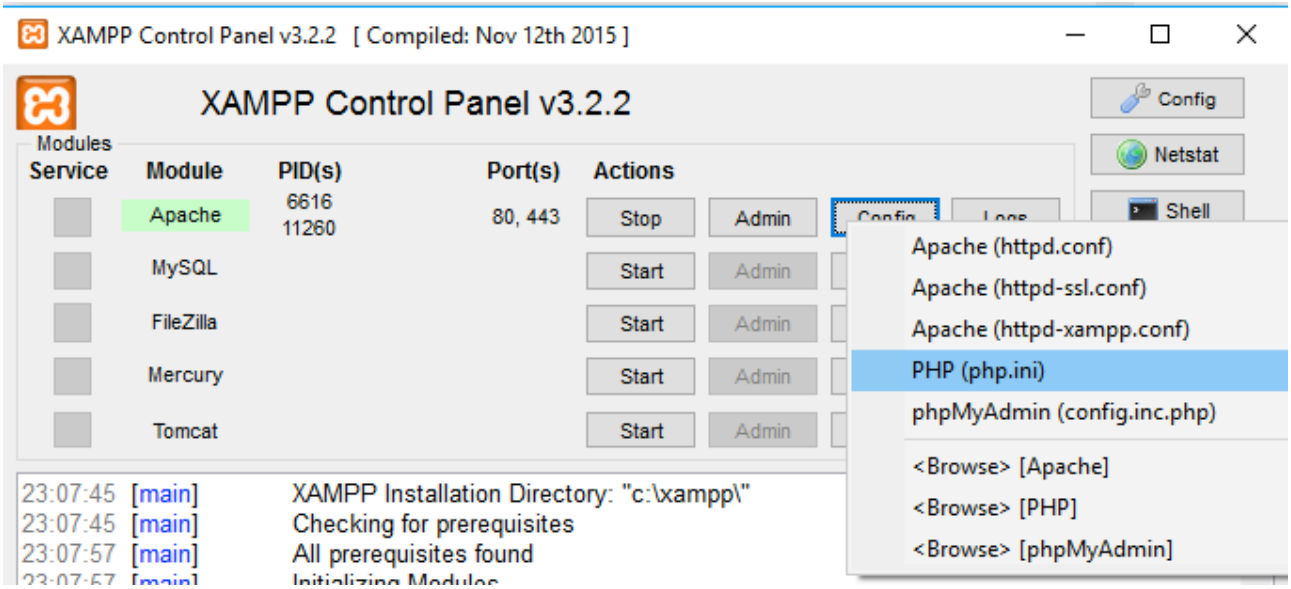
A sintaxe limitadora do PHP padrão, abre com **<?php** e termina com **?>**.

```php
<?php

    echo "Já sou programador!";

?>
```

Uma outra alternativa é utilizar a ***Short-Tags***, mas somente é funcional se estiver configurada na diretriz ***short_open_tag*** do PHP.

```php
<?
    echo "Já sou programador!";
?>
```

Para habilitar este recurso, vamos abrir o painel de controle do XAMPP e clicar no botão **Config** do **Apache** e depois selecionar **PHP** (**php.ini**)



Encontre a linha de código que contém **short_open_tag=Off**, altere o valor para **On** e reinicie o servidor Apache.

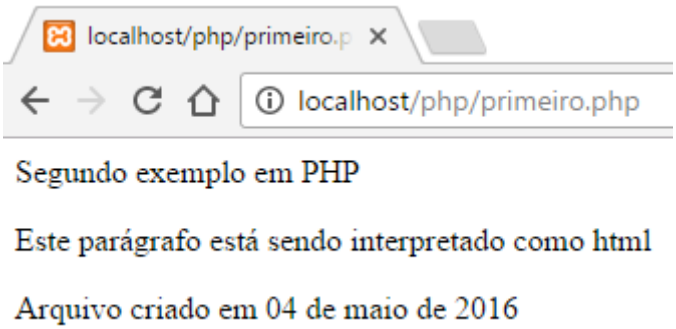
```
; Development Value: Off
; Production Value: Off
; http://php.net/short-open-tag
short_open_tag=On
; XAMPP for Linux is currently old fashion
;short_open_tag = On
```

Quando a ***short-tag*** estiver ativada, podemos utilizar uma saída rápida chamada ***short-circuit***.

```
<?="Já sou programador!"; ?>
```

1.4 Múltiplos Blocos

A inserção do código PHP dentro da página pode ser realizada quantas vezes forem necessárias, devendo sempre usar a sintaxe limitadora **<?php ?>**.



1.5 Comentários no Código

Na linguagem PHP podemos utilizar comentários no código-fonte, das seguintes maneiras:

// Comentário de uma única linha.
/* e ***/** para comentários de múltiplas linhas.
*(hash marks)* também é suportada para únicas linhas

```
<html>
<?php
    #Testando comentário hash marks
    #Adriano Cilhos Doimo
?>
<head>
    <title> PHP </title>
</head>
<body>
    <?php
        /*
            Este é um comentário
            com multiplas linhas.
        */
        echo "Segundo exemplo em PHP";
        $data = "04 de maio de 2016";

        //Este ´w um comentário de apenas uma linha
    ?>

    <p>Este parágrafo está sendo interpretado como html</p>

    Arquivo criado em <?php echo $data; ?>
</body>
</html>
```

2 Exibindo dados no navegador

O PHP oferece diversos métodos para exibir informações no navegado, que são elas:

- **Função print()** imprime dados para o navegador, retornando uma resposta se conseguiu exibir corretamente ou não
- **Função echo()** também imprime dados para o navegador, porém sem retorno de sucesso.
- **Função printf()** permite utilizar texto estático e dinâmico armazenado dentro de variáveis, usando formatação de casas decimais por exemplo.
- **Função sprintf()** é o mesmo do **printf(),** mas sua saída é para uma variável.

Administração Central

Cetec Capacitações

```php
<?php
    print "Usando print para exibir na tela <br />";

    $data = "04 de maio de 2016";
    $salario = 850.00 ;
    $cargo = "Estagiário";
    echo "Arquivo criado em $data <br />" ;

    printf("Salário mínimo: R$ %.2f <br />", $salario);

    $texto = sprintf("%s recebe R$ %.2f por mês", $cargo, $salario*2);

    echo $texto;
?>
```

Tabela de Padrões:

%b - O argumento é tratado com um inteiro e mostrado como um binário.

%c - O argumento é tratado como um inteiro e mostrado como o caractere ASCII correspondente.

%d - O argumento é tratado como um inteiro e mostrado como um número decimal com sinal.

%e - O argumento é tratado como notação científica (e.g. 1.2e+2).

%u - O argumento é tratado com um inteiro e mostrado como um número decimal sem sinal.

%f - O argumento é tratado como um float e mostrado como um número de ponto flutuante.

%F - o argumento é tratado como um float e mostrado como um número de ponto flutuante.

%o - O argumento é tratado com um inteiro e mostrado como um número octal.

%s - O argumento é tratado e mostrado como uma string.

%x - O argumento é tratado como um inteiro e mostrado como um número hexadecimal (com as letras minúsculas).

%X - O argumento é tratado como um inteiro e mostrado como um número hexadecimal (com as letras maiúsculas).

Fonte: http://php.net/manual/pt_BR/function.sprintf.php

3 Tipos de dados

O PHP oferece um conjunto bem amplo de dados, dentre eles, os tipos mais comuns incluem ***boolean***, ***integer***, ***float***, ***string*** e ***array***.

- **boolean** suporta penas 2 valores ***TRUE*** e ***FALSE*** (como alternativa pode usar o número **zero** para representar o ***FALSE*** e qualquer **outro valor** para representar o ***TRUE***).
- **integer** representa qualquer número inteiro.
- **float** representa qualquer número com casa decimal.
- **String** representa uma sequência de caracteres tratados como um grupo (arrays), delimitados por aspas simples (‘) ou duplas (“).

3.1 Tipos de dados compostos

Permitem múltiplos itens do mesmo tipo agregados em uma única entidade administrativa.

- **Array** é formalmente definida como uma coleção indexada de valores de dados.
- **Objetos** conceito central da programação orientada a objeto, o qual deve ser explicitamente declarado e instanciado na programação.

3.2 Type Casting

Ao indicarmos um operador de tipo antes da variável a ser impressa, podemos modificar seu tipo inicial, usando ***type casting***.

Operadores	Conversão
(array)	Array
(bool) ou (bololean)	Boolean
(int) ou (integer)	Integer

(int64)	Integer de 64 bits (PHP 6)
(object)	Object
(real) ou (double) ou (float)	Float
(string)	String

Considere alguns exemplos:

$valor = (double)968 // $valor = 98.0

$custo = (int) 23.53 // $custo= 23

3.3 Tipo Juggling

O PHP considera as variáveis automaticamente dependendo das circunstâncias nas quais estão referenciadas, ou seja, identifica seu tipo de acordo com o valor atribuída a variável de forma dinâmica.

```php
<?php

    $data = "04 de maio de 2016";    //string
    $salario = 850.00 ;              //float
    $cargo = "Estagiário";           //string
    $idade = 18;                     //integer
    $resultado = true                //boolean

    if ($resultado) echo "Verdadeiro";

?>
```

Resposta: Verdadeiro

3.4 Variáveis e Declaração

Uma variável começa com o símbolo de dólar (**$**), que é então seguido pelo seu nome, o qual deverá começar com uma letra ou um *underscore* e pode ser constituída por letras, números e *underscore.* São validas:

declararvar.php

```php
<?php
    // declarações corretas
    $cor = "azul";
    $cor_de_fundo = "amarelo";
    $_cor = "vermelho";

    // as variáveis abaixo não possuem nenhum relação uma com a outra
    $fundo = "azul";
    $Fundo = "amarelo";
    $FUNDO = "vermelho";
    echo $fundo."-".$Fundo."-".$FUNDO;
?>
```

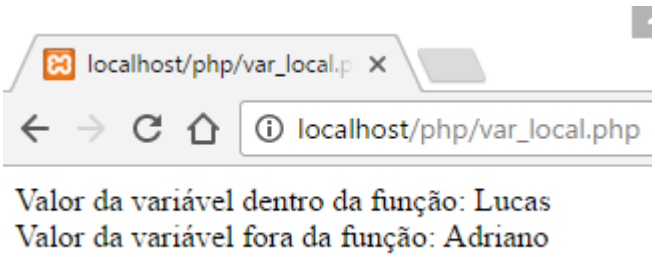
4 Escopo de variáveis

Podemos declarar as variáveis em qualquer lugar de um script PHP, mas a sua localização influencia no domínio em que pode ser acessado, esse domínio é conhecido como escopo. Vamos identificar a seguir cada escopo de variável e entender sua abordagem:

- **Variáveis Locais** são aquelas declaradas dentro do *script* ou das funções.

```php
<?php
    $nome = "Adriano";  //variável local no script

    //implementação de uma função
    function exibir(){
        $nome = "Lucas";    //variável local na função
        echo "Valor da variável dentro da função: ".$nome;
    }

    exibir();   //chamada para função
    echo "<br /> Valor da variável fora da função: ".$nome;
?>
```

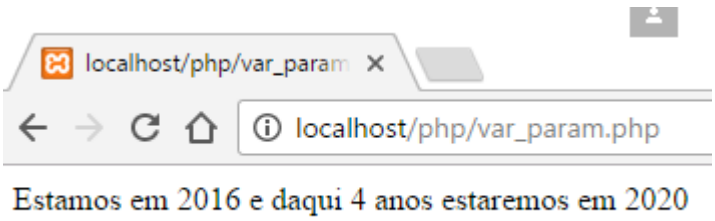
O resultado apresentado no navegador, exibe os respectivos valores atribuídos à elas, sendo que a primeira mensagem que está dentro da função e a segunda mensagem a que está fora da função. Embora as variáreis tenham o mesmo nome, como estão declaradas em locais distintos, são diferentes no escopo do código.



- **Parâmetros de Função** são declaradas depois no nome da função e entre parênteses.

```php
<?php
    $ano = 2016;  //variável local no script

    //implementação de uma função
    function exibir($parametro){    //variável como parâmetro
        $parametro = $parametro + 4;
        return $parametro;
    }

    echo "Estamos em ".$ano." e daqui 4 anos estaremos em ".exibir($ano);
?>
```

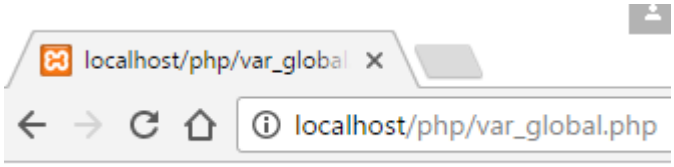
O resultado apresentado no navegador, mostra o valor da variável **ano** e também o resultado do retorno na função. Uma função pode ter vários parâmetros.



- **Variáveis Globais** pode ser acessada em qualquer parte do sistema.

```php
<?php
    $ano = 2016;

    //implementação de uma função
    function exibir(){
        GLOBAL $ano;          //definindo variável global
        $ano++;
        return $ano;
    }

    echo "<br /> Ano: ".$ano;
    echo "<br /> Ano: ".exibir();
    echo "<br /> Ano: ".exibir();
    echo "<br /> Ano: ".$ano;
?>
```

O resultado apresentado no navegador, mostra o valor inicial da variável **ano** e na sequência foram realizadas duas chamadas da função **exibir**, e podemos identificar que o valor da variável foi alterado no escopo global, tanto dentro como fora da função.

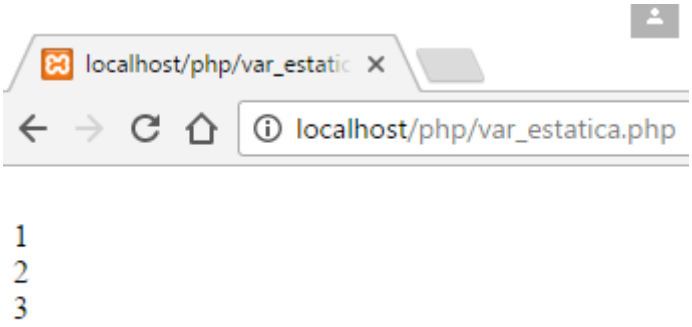


Ano: 2016
Ano: 2017
Ano: 2018
Ano: 2018

- **Variáveis Estáticas** supondo uma variável declarada dentro de uma função, quando esta função é fechada a variável não perde o seu valor;

```php
<?php

    function exibir(){
        STATIC $ano;          //definindo variável estática
        $ano++;
        echo "<br />". $ano;
    }

    echo exibir();
    echo exibir();
    echo exibir();

?>
```

O resultado apresentado no navegador, mostra que a variável **ano** não teve seu valor perdido, após a função ter sido executada por três vezes.



5 Variáveis Super Globais

São variáveis nativas que estão sempre disponíveis em todos escopos. Elas estão disponíveis em todos escopos para todo o script. Não há necessidade de fazer **global $variavel;** para acessá-lo dentro de funções ou métodos. As variáveis superglobais são $GLOBALS, $_SERVER, $_GET, $_POST, $_FILES, $_COOKIE, $_SESSION, $_REQUEST *e* $_ENV. Podemos identificar no código a seguir, exemplos de como utilizar as variáveis superglobais, como identificar o IP da máquina e o navegador (browser).

```php
<?php

    echo "<br /> Olá! Seja Bem Vindo".$_SERVER['REMOTE_ADDR']."<br />";
    echo "Você está utilizando o navegador: ".$_SERVER['HTTP_USER_AGENT'];

?>
```

6 Constantes

É um identificador para um valor único, ou seja, esse valor não pode mudar durante a execução do script.. Utilizando boas práticas, os identificadores de constantes são sempre em maiúsculas. A função **define()** registra uma constante a partir de um nome.

```php
<?php

    define("PI", 3.14);
    define("DISC", "Matemática");

    echo "<br /> Valor de PI: ".PI;
    echo "<br /> Disciplina: ".DISC;

?>
```

7 Expressões

Uma expressão é uma frase representando uma ação em particular, constituída de um operando e um ou mais operadores. Todas as instruções/linhas de código que envolvem algum valor, podemos defini como expressões e operadores são sinais (veremos a seguir).

Operandos, são as entradas de uma expressão, ou seja, as variáveis que contém dados.

```
$ano++                  // $ano é um operando.
$soma = $n1 e $n2       // $soma, $n1 e $n2 são operandos.
```

7.1 Operadores

É um símbolo que especifica uma ação em particular em uma expressão, e sua precedência e associação devem ser respeitadas. Um operador é algo que recebe um ou mais valores (ou expressões) e que devolve outro valor.

Associação	Operador	Objetivo
NA	new	instanciação do objeto
NA	()	subgrupos de expressão
Direita	[]	índice de posição
Direita	++ --	incremento e decremento
Direita	@	supressão de erro
Esquerda	/ * %	divisão, multiplicação e módulo
Esquerda	+ - .	adição, subtração e concatenação
NA	< <= > >=	menor, menor igual, maior, maior igual
NA	== != === <>	igual, não é igual, idêntico, diferente
Direita	= += -= *= /= %=	atribuição de operadores
Esquerda	&& ||	AND e OR (booleano)
Esquerda	AND XOR OR	AND, XOR e OR (booleano)
Esquerda	, (virgule)	separação de expressões

Fonte: http://php.net/manual/pt_BR/language.operators.precedence.php

8 Estrutura de controle

Estruturas de controle determinam o fluxo de códigos dentro de uma aplicação. A estrutura de controle é dividida em duas partes: comandos de seleção (if e switch) e comandos de repetição (for, while, do/while e foreach).

8.1 Comandos de Seleção

8.1.1 Expressão if

Podemos dizer que é expressão mais utilizada em qualquer linguagem de Programação, pois basicamente na maioria das vezes, temos que realizar comparações, que devem resultar verdadeiro ou falso.

Sintaxe:

```
if(expressão){
        comando(s)
}
```

Podemos observar que na sintaxe, se o resultado da expressão for verdadeira, executará o bloco de comandos (delimitado em chaves). O exemplo a seguir, demonstra a utilização de uma condição simples, utilizando apenas *if*.

Exemplo:

```php
<?php

    $nota = 8.0;
    if ($nota >= 7.0){
        echo "APROVADO";
    }

?>
```

8.1.2 Expressão if-else

Como pudemos observar, a estrutura *if* trata apenas se a condição for verdadeira. Neste caso, podemos tratar o resultado da condição falso, usando *else*. Ou seja, se o valor testado na condição if for verdadeira, executa os comandos dentro do bloco do *if*, senão, executa o bloco de comandos do *else*.

```php
<?php

    $nota = 8.0;
    if ($nota >= 7.0){
        echo "APROVADO";
    }
    else{
        echo "REPROVADO";
    }

?>
```

8.1.3 Expressão elseif

A expressão **if-else** funciona muito bem quando a situação é "um-ou-outro", ou seja, somente dois resultados são permitidos, a expressão **elseif** permite identificar novas possibilidade na sequência de testes.

```php
<?php

    $nota = 5.0;

    if ($nota < 0.0 && $nota > 10.0){
        echo "NOTA INVÁLIDA";
    }
    elseif ($nota >= 7.0){
        echo "APROVADO";
    }
    else{
        echo "REPROVADO";
    }

?>
```

8.1.4 Expressão switch

Pode ser utilizada como uma variante da combinação **if-elseif-else**, usando quando existe um grande número de valores para serem testados. Um exemplo muito comum de utilização é em programas com menu de opções, onde o usuário informa a opção desejada e somente as instruções relacionadas ao bloco de comandos daquela opção escolhida, serão executadas.

Sintaxe:

```
switch (expressão){
    case num1:
      comandos;
      break;

    case num2:
     comandos;
     break;

    case num3:
     comandos;
     break;

    case num4:
     comandos;
     break;

    default:
     Comandos;
     break;
 }
```

No exemplo a seguir, foi atribuída na variável *opcao* o valor 5, e neste caso, como não existe o *case 5*, será executada a instrução do *default*, pois seria a negação para o caso de não encontrar nenhum ***case*** como valor da opção desejada.

```php
<?php

    $opcao = 5;

    switch($opcao){

        case 1: echo "A opção 1 foi selecionada";
        break;

        case 2: echo "A opção 1 foi selecionada";
        break;

        case 3: echo "A opção 1 foi selecionada";
        break;

        default: echo "Opção incorreta";
        break;

    }
?>
```

8.2 Comandos de Repetição

As estruturas de repetição oferecem uma maneira simples para realizar a repetição de uma ou várias tarefas de acordo com uma condição. Também são conhecidos como estruturas de repetição, iterações ou loops, mantendo a execução de blocos de comandos, até que seu argumento seja falso.

8.2.1 Expressão while

A expressão ***while*** especifica uma condição no início do comando, onde será executado o bloco de comando, somente se esta condição for verdadeira, ou seja, só executa as instruções se a expressão for verdadeira.

Sintaxe:

```
while (expressão){
        comandos(s)
}
```

No exemplo a seguir, será exibido no navegador todos os números de 0 a 10 em ordem crescente. Se o valor inicial de *n* for um valor maior que 10, nenhum resultado será exibido no navegador, pois a condição será falsa.

```php
<?php

    $n = 0;
    while($n <= 10){
        echo "<br />".$n;
        $n++;
    }

?>
```

No exemplo a seguir, será exibido no navegador todos os números de 1 a 100 em ordem decrescente. Se o valor inicial de *n* for um valor menor que 0, nenhum resultado será exibido no navegador, pois a condição será falsa.

```php
<?php

    $n = 100;
    while($n > 0){
        echo "  ".$n;
        $n--;
    }

?>
```

8.2.2 Expressão do..while

A expressão ***do..while*** executa pelo menos uma vez os comandos do bloco de repetição e realiza o teste da condição no final do bloco, ou seja, realiza as instruções no mínimo uma vez e enquanto a expressão for verdadeira.

Sintaxe:

```
do {
        comando(s)
} while (expressão);
```

Neste exemplo a seguir, serão exibidos os números pares de 500 até 550 em ordem crescente. Se alterarmos o valor inicial de *n* para 800, teremos como resultado de saída apenas o valor 500, pois executará pelo menos uma vez o bloco de instruções.

```php
<?php

    $n = 500;
    do{
        echo "<br />".$n;
        $n+=2;
    }while($n <= 550);

?>
```

Neste exemplo a seguir, serão exibidos os números de 10 e 10, de 9 até 150 em ordem decrescente. Se alterarmos o valor inicial de *n* para 2, teremos como resultado de saída apenas o valor 150, pois executará pelo menos uma vez o bloco de instruções.

```php
<?php

    $n = 150;
    do{
        echo "<br />".$n;
        $n-=10;
    }while($n >= 9);

?>
```

8.2.3 Expressão for

A expressão **for** oferece um mecanismo de repetição mais completo, indicando a expressão inicial, a expressão condicional e expressão que determinada a operação do looping. Uma das grandes vantagens desta estrutura é que podemos determinar em uma única linha de instruções, a inicialização, a condição de parada e o incremente ou decremento, diminuindo as chances de looping infinito.

Sntaxe:

```
for (expressão 1; expressão 2; expressão 3){
        comando(s);
}
```

Descrevendo as expressões do comando for:

- Expressão 1: determina a inicialização da variável
- Expressão 2: determina a condição de parada do bloco de instruções
- Expressão 3: determina o incremento ou decremento da variável

Neste exemplo, será exibido no navegador todos os valores de 0 a 10 em ordem crescente.

```
<?php

    for($x=0; $x <= 10; $x++){
        echo "<br />".$x;
    }

?>
```

Neste próximo exemplo, será exibido no navegador todos os valores de 0 a 10 em ordem decrescente.

```
<?php

    for($x=10; $x >= 0; $x--){
        echo "<br />".$x;
    }

?>
```

8.2.4 Expressão foreach

É uma simplificação do comando *for* para se trabalhar em coleções de dados, ou seja, vetores e matrizes. É extremamente útil, pois com ele você executa um grupo de comandos para cada elemento de um array.

Sintaxe:

```
foreach (array_expressao as $valor){
        comando(s)
}
```

Neste exemplo a seguir temos a declaração de um vetor de nomes e a estrutura *foreach* irá percorrer as posições do vetor e atribuir os valores na variável *item*, para exibir na tela os valores.

```php
<?php

    $nomes = array('Adriano', 'Jorge', 'Paulo', 'Mateus');

    foreach ($nomes as $item) {
        echo "<br />".$item;
    }

?>
```

8.3 Outras expressões

8.3.1 Expressão break

Permite finalizar a execução do bloco de comandos de uma estrutura. Vimos a utilização no comando switch, mas também podemos utilizar em blocos de repetição, para interromper o bloco de comandos quando uma determinada condição for satisfeita.

No exemplo a seguir, foi utilizada a expressão break, para interromper o laco de repetição quando o valor da variável for igual a 50.

```php
<?php

    $x=0;
    while ($x < 100) {
        echo "<br />".$x;
        if ($x == 10)
            break;
        $x++;
    }

?>
```

8.3.2 Expressão goto

Permite pular para uma localização específica fora do looping. O ponto de destino é definido por um rótulo seguido de dois pontos, e a instrução é usada como *goto* seguida do rótulo de destino desejado

Neste exemplo o fluxo será desviado, quando o valor da variável *x* for igual a 10.

```php
<?php

    for ($x=1 ; $x < 100; $x++) {

        if ($x == 10)
            goto desvio;

        echo "<br />".$x;
    }

    desvio:
    echo "<br /> Pronto, aqui está o desvio";
?>
```

8.3.3 Expressão continue

Permite que a execução da repetição do loop atual termine e inicie na próxima repetição., ou seja, o looping não é encerrado, apenas encerramos a **iteração**, volta do looping, atual e iniciamos a próxima.

No exemplo a seguir, irá aparecer no navegador todos os números ímpares de 1 a 100, pois sempre que a condição testar um número e ele for par, irá pular para a próxima iteração..

```php
<?php

    for($x=1; $x <= 100; $x++ ){
        if($x % 2 == 0){
            continue;
        }
        echo "<br />".$x;
    }
?>
```

9 Função

Pode ser definida como blocos de código com um objetivo específico, identificados por um nome através do qual pode ser referenciado a partir de várias partes do código. Essa é uma das principais técnicas utilizadas para garantir a reutilização de código, tornando a programação mais prática, diminuir linhas de código e organizado.

Podemos criar funções customizadas ou até mesmo bibliotecas inteiras de funções, para fazer isso uma função deverá ser definida usando o modelo pré-definido.

Sintaxe:

```
function nome(parâmetros){
        Corpo da função (comando(s));
}
```

Neste exemplo, foi criada uma função chamada mensagem, que irá exibir apenas um texto e logo depois foi realizada uma chamada para esta função ser executada.

```php
<?php

    function mensagem(){
        echo "<br /> Minha primeira função.";
    }

    mensagem();
?>
```

As funções podem ser classificadas quanto ao seu retorno como vazias (void) ou com retorno. As funções void ou sem retorno apenas executam uma série de comandos sem a obrigação de devolver um valor específico como resultado. Já as funções com retorno, ao serem executadas, resultam diretamente em um dado valor, sem se preocupar com o tipo do dado.

9.1 Função com argumento/parâmetro

Podemos determinar que uma função seja executada, com base em ~dados passados para ela, ou seja, recebendo valores. Esse recurso é muito importante, pois assim conseguimos identificar exatamente o que é necessário para que uma determinada função funcione corretamente.

Neste exemplo, foi criada uma função chamada dobro, preparada para receber um valor como parâmetro, realizar o cálculo do dobro de um número e exibir o resultado na tela. Observe que isso tudo está dentro da função.

```php
<?php

    function dobro($n){
        $d = $n * 2;
        echo "<br /> O Dobro de ".$n." = ".$d;
    }

    dobro(6);

?>
```

9.2 Função com argumento e retorno

A sintaxe praticamente e a mesma, porém devolve um resultado para o código, usando a palavra-chave *return*.

Para exemplificar usaremos como base o exemplo anterior, porém agora a função além de receber um parâmetro e realizar o cálculo, também irá retornar o resultado do cálculo.

```php
<?php

    function dobro($n){
        $d = $n * 2;
        return $d;
    }

    $x = 6;
    echo "<br /> O Dobro de ".$x." = ".dobro($x);
?>
```

9.3 Biblioteca de funções

Pensado em reusabilidade de código, podemos criar várias funções em um único arquivo e usar em posteriormente com outros arquivos, ou seja, escrevemos uma vez e utilizamos quantas vezes forem necessárias, inclusive para projetos diferentes.

Para isso, primeiro vamos criar um arquivo chamado ***biblioteca.php***, que será nossa biblioteca de funções.

```php
<?php

    function dobro($n){
        return $d * 2;
    }

    function quadrado($n){
        return $n * $n;
    }

?>
```

Agora vamos criar outro arquivo que será a nossa página principal, onde iremos incluir a identificação desta biblioteca no código, para que as funções possam ser executadas. Para incluir o arquivo de bibliotecas, usaremos ***include***.

```php
<?php
    include("biblioteca.php");

    $x = 6;
    echo "<br /> O Dobro de ".$x." = ".dobro($x);

    echo "<br /> O Quadrado de ".$x." = ".quadrado($x);

?>
```

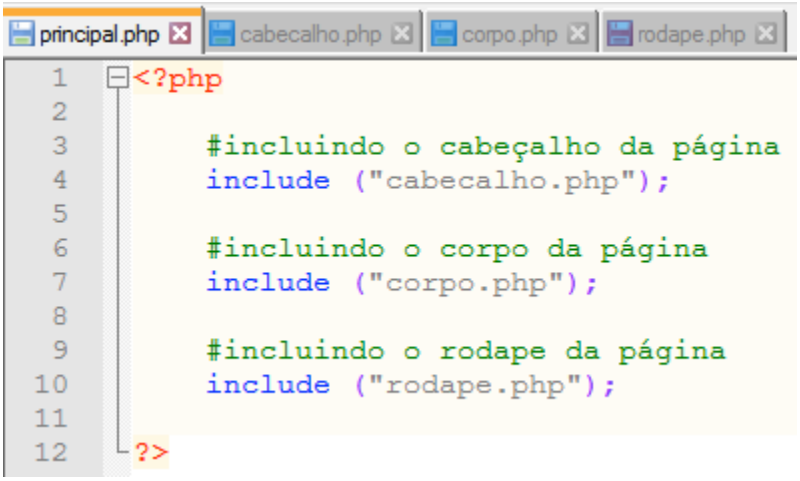
9.4 Formas de incluir arquivos

Podemos incluir arquivos no código de uma página, isso significa copiar os dados deste arquivo para dentro do código principal. Existem algumas formas de se fazer isso:

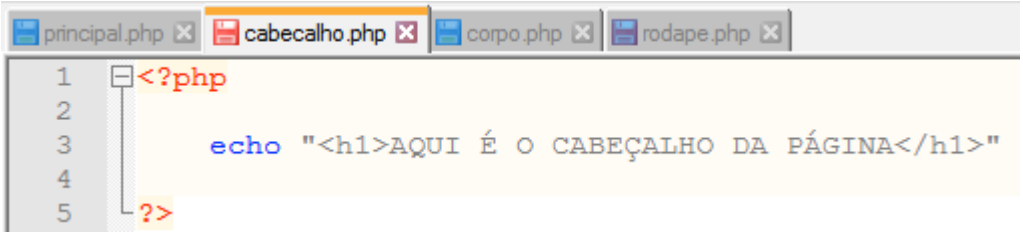
- **include** tenta incluir uma página. Caso de algum erro, o script retorna um warning (aviso) e prossegue com a execução do script.
- **require** tenta incluir uma página. Caso de algum erro, o script retorna um fatal error e aborta a execução do script.
- **include_once** e **require_once** possuem as mesmas finalidades das anteriores, porém se o arquivo referenciado já foi incluso na página anteriormente, a função retorna 'false' e o arquivo não é incluído.

Como exemplo uma prática muito utilizada é criar uma página principal e carregar nela o cabeçalho, corpo e rodapé. Desta forma, os arquivos ficam bem organizados e fácil de manutenção conforme a necessidade.

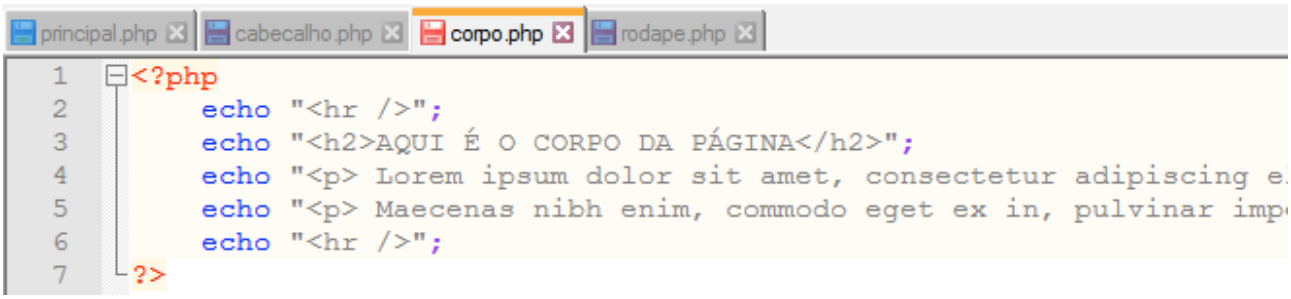
Arquivo: **principal.php**



Arquivo: **cabecalho.php**



Arquivo: **corpo.php**



Arquivo: **rodape.php**

principal.php | cabecalho.php | corpo.php | rodape.php

```php
<?php

    echo "<h3>AQUI É O RODAPÉ DA PÁGINA</h3>"

?>
```

9.5 Expressão include condicionada

A expressão **include** também pode estar condicionada, ou seja, um arquivo será incluído quando uma expressão for verdadeira. No exemplo a seguir, temos duas condições na página principal, onde são chamadas páginas específicas dependendo do valor da variável *idade*.

Arquivo: **votacao.php**

votacao.php | votacao_menor.php | votacao_16.php | votacao_maior.php

```php
<?php
    include("cabecalho.php");

    $idade = 16;

    if ($idade < 16)
        include ("votacao_menor.php");
    elseif ($idade < 18)
            include ("votacao_16.php");
        else
            incluede ("votacao_maior.php");

    include("rodape.php");

?>
```

Arquivo: **votação_menor.php**

votacao.php | votacao_menor.php | votacao_16.php | votacao_maior.php

```php
<?php

    echo "<br /> Menor de 16 não pode votar";

?>
```

Arquivo: **votação_16.php**

votacao.php | votacao_menor.php | votacao_16.php | votacao_maior.php

```php
<?php

    echo "<br /> Com no mínimo 16 anos já pode votar <br />";
    echo "<br /> Mas o voto não é obrigatório <br />";

?>
```

Arquivo: **votação_maior.php**

votacao.php | votacao_menor.php | votacao_16.php | votacao_maior.php

```php
<?php

    echo "<br /> Para maiores de 18 o voto é obrigatório <br />";

?>
```

9.6 Expressão include_once

Tem a mesma função da expressão include(), mas verifica se o arquivo já foi incluído.

```php
<?php

    #inclui o cabeçalho da página apenas uma única vez
    include_once ("cabecalho.php");
    include_once ("cabecalho.php");
    include_once ("cabecalho.php");

    #inclui o rodape da página 3 vezes
    include ("rodape.php");
    include ("rodape.php");
    include ("rodape.php");

?>
```

9.7 Expressão require

Sua operação é semelhante ao **include**(), mas pode falhar caso não localize o arquivo.

```php
<?php

    require ("cabecalho.php");

    require ("corpo_errado.php");

    require ("rodape.php");

?>
```

10 Array

Também tradicionalmente conhecidos como vetores, são variáveis que servem para guardar vários valores de forma uniforme na memória. É um grupo de itens que compartilham certas características, como similaridade e tipo de dados. Até o momento, se quiséssemos armazenar o nome de 10 pessoas, teríamos que declarar 10 variáveis ($nome1, $nome2, $nome3 ,..., $nome10). Mmas utilizando vetor, bastaríamos declarar apenas 1 variável do tipo array e armazenar os 10 nomes nesta variável.

Em PHP não precisa que se defina o tamanho do array no momento da criação, nem menos que você declare o array antes de usá-lo.

Declaração básica de um array: $vetor = array();

Declaração de um array com inicialização de valores:

$vetor = array{ "Maria" , "José" , "Pedro" };

Podemos também inicializar cada valor, em cada posição do vetor:

```
$vetor[0] = "Maria";
$vetor[1] = "José";
$vetor[2] = "Pedro";
```

Para exibir dos dados do vetor, temos que indicar a posição do vetor que queremos acessar:

```
echo $vetor[0];
echo $vetor[1];
echo $vetor[2];
```

Uma outra forma de atribuir, é não precisando indicar o índice/posição:

```
$vetor[] = "Maria";
$vetor[] = "José";
$vetor[] = "Pedro";
```

10.1 Array associado

Podemos também criar um array associado, ou seja, capaz de fazer associações por meio dos índices, que geralmente em vez de números, são usadas strings par definir a posição no vetor.

```
$estado["MA"] = "Maria";
$estado["ZE"] = "José";
$estado["PE"] = "Pedro";
```

Podemos também inicializar um array associativo da seguinte forma:

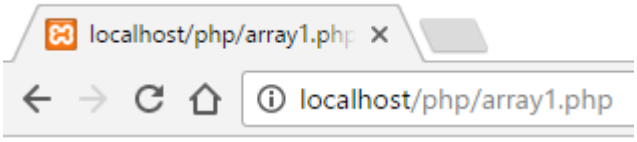
```
$vetor = array ("MA" => "Maria", "ZE" => "José", "PE" => "Pedro");
```

10.2 Função is_array()

Esta função serve para fazer um teste e saber se a variável é do tipo array.

```php
<?php

    $variavel = "Ana";
    $nomes = array ("MA" => "Maria", "ZE" => "José", "PE" => "Pedro");

    if (is_array($variavel))
        echo "<br />variavel é um array";
    else
        echo "<br />variavel não é um array";

    if (is_array($nomes))
        echo "<br />nomes é um array";
    else
        echo "<br />nomes não é um array";

?>
```

Como resultado no navegador, temos:



variavel não é um array
nomes é um array

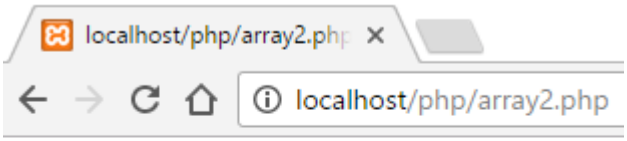
10.3 Adicionando valores no vetor

10.3.1 Função array_unshift()

Esta função tem a finalidade de adicionar elementos na frente (início) do array.

```php
<?php

    $nomes = array ("Maria", "José", "Pedro");
    array_unshift ($nomes, "Ana");

    foreach ($nomes as $item){
        echo "<br />".$item;
    }

?>
```

Podemos notar, que o nome Ana, foi inserido na primeira posição do vetor.



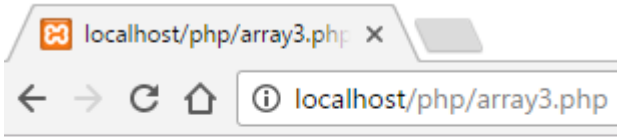
Ana
Maria
José
Pedro

10.3.2 Função array_push()

Esta função tem a finalidade de adicionar elementos no final do array.

```php
<?php

    $nomes = array ("Maria", "José", "Pedro");
    array_push ($nomes, "Ana");

    foreach ($nomes as $item){
        echo "<br />".$item;
    }

?>
```

Podemos notar, que o nome Ana, foi inserido na última posição do vetor.



Maria
José
Pedro
Ana

10.4 Removendo valores do vetor

10.4.1 Função array_shift()

Esta função remove e retorna o item encontrado na primeira posição do vetor.

```php
<?php

    $nomes = array ("Maria", "José", "Pedro");
    $excluido = array_shift ($nomes);

    foreach ($nomes as $item){
        echo "<br />".$item;
    }

    echo "<br /><br /> Nome removido: ".$excluido;
?>
```

Podemos notar, que o nome Maria que constava na primeira posição do vetor, foi removido.