Programação Avançada / Programação Mobile

Alexandre Erdmann Silva

O que faz o PHP?

- O PHP pode ser dividido em duas vertentes:
- **Programação backend:** criação de sites dinâmicos, conexão e interação com banco de dados, geração de gráficos, documentos de XML e PDF;
- **Scripts de linha de comando:** rodar scripts para que ações sejam executadas no computador ou remotamente, administração de sistema ou até mesmo CRONs;

O que é de fato PHP?

- Uma **linguagem de programação** que foi desenvolvida para deixar o **HTML dinâmico**;
- Linguagem de **script** e **open source**;
- O foco de PHP é o **desenvolvimento web**;
- Os programas são executados em **server side** (lado do servidor, backend);
- PHP significa: **Personal Home Page** (P) e **Hypertext Preprocessor** (HP)

Uma breve história

- O criador da linguagem foi **Rasmus Lerdorf**;
- O ano de lançamento foi **1994**;
- A linguagem já está **quase na versão 8**, diversas melhorias foram implementadas;

- A **ideia principal era deixar o HTML dinâmico**, não havia pretensão de ser o que é hoje;
- **Quase 80%** dos sites hoje (2020) contém PHP;

Instalação VS Code

- O **VS Code** é com certeza um dos editores mais utilizados atualmente;
- Ele também facilita muito a nossa vida com o **terminal integrado** e as suas extensões;
- Por estes e outros motivos, será o **editor utilizado no curso!**
- Podemos seguir as instruções do site: **code.visualstudio.com**

Instalação PHP Windows

- O PHP pode ser instalado no Windows **sem a instalação de ferramentas**, porém a própria documentação recomenda o **XAMPP**;
- Que é um pacote que contém os principais recursos para desenvolvimento web: **Apache, MySQL, Pearl e PHP**;
- Com o XAMPP conseguimos **simular um servidor web** de modo fácil e também executar qualquer código PHP;

Executando PHP no Windows

- **Após a instalação do XAMPP**, devemos colocar os arquivos que serão executados em uma pasta específica;
- O software vem configurado para rodar na pasta **htdocs**, dentro do local de sua instalação;
- Arquivos que estão lá podem ser acessados pelo navegador no endereço **localhost**;
- Lembrando que **o servidor precisa estar ligado**;

Instalação PHP Linux

- No Linux podemos instalar o **PHP de forma independente**, porém é interessante fazer a instalação da **pilha LAMP**;
- Linux, Apache, MySQL e PHP;
- Com isso poderemos criar e executar não só arquivos de PHP mas também **projetos web completos**;

Executando PHP no Linux

- **Após a instalação do LAMP**, podemos inserir arquivos na pasta **var/www/html**;
- Os arquivos desta pasta podem ser acessados no navegador pelo endereço de **localhost**;

O que é localhost?

- É a forma de **acessar o servidor local da nossa máquina**;
- Equivale ao IP **127.0.0.1**, ou seja, nosso PC;
- Assim podemos simular como se o site ou arquivo que estamos acessando é **processado em um servidor**;
- IP significa Internet Protocol;

A sintaxe do PHP

- O PHP vai interpretar um bloco de código em nosso arquivo apenas se ele estiver **entre as tags de PHP**;
- A abertura é: **<?php**
- E o fechamento: **?>**
- Coloque **;** a cada instrução;
- Todo o código dentro destas tags será executado pelo PHP, e após isso será impresso na tela;

PHP e suas dependências

- Podemos checar como o **PHP e suas dependências** estão instaladas no nosso computador;
- O nome da função é **phpinfo()**
- Ela exibe as **versões** também dos pacotes instalados;
- Útil para saber como o servidor está configurado;

Case sensitivity

- Significa sensibilidade a casas maiúsculas e minúsculas;
- Para **instruções PHP não temos essa diferença**, ou seja: echo = ECHO;
- Porém para **variáveis são case sensitive**;
- Ou seja, $nome != $NOME;
- Obs: veremos variáveis em detalhes mais adiante;

Instruções de código

- As instruções simples de PHP são **separadas por ponto e vírgula**;
- Instruções simples são instruções de uma linha;
- Quando há uma **instrução maior**, como de condição ou repetição, a definição da mesma é dada por **abertura e fechamento de chaves**;
- Nestes casos **não precisaremos** de ponto e vírgula;

Espaços em branco

- Para interpretação do código em PHP **o espaço em branco é ignorado**;
- Isso acontece pois o mesmo **é removido antes da execução**;
- A **quebra de linha também também é ignorada**;
- Porém se utilizada de má forma pode gerar erros inesperados no código;

Comentários

- Os comentários servem para dar **informações e direções importantes** de como o código funciona;
- Iniciamos um comentário com **//**
- Todo conteúdo que está em um **comentário é ignorado** na execução;
- **Não insira informações sensíveis** nos comentários;
- Outra forma de inserir comentários é com **#**
- Comentários multi linhas são feitos com: **/* comentário */**

Palavras reservadas

- Algumas palavras são **reservadas da linguagem** e já tem suas funcionalidades definidas, então não podemos utilizar em nossos programas;
- Pois caso fosse possível **poderíamos substituir** a sua função original;
- **Alguns exemplos são:** echo, insteadof, else, interface, namespace,pow, __DIR__, __FILE__, endif, print, private, protected, and, require, public, as, break, case, for, finally, switch, throw e etc;

Tipos de dados

Introdução da seção

Inteiros (integers)

- Os inteiros são os **números inteiros** da matemática, como: 1, 2, 15;
- Incluindo os **números negativos**;
- Os números positivos **não precisam** de um sinal de + na frente;
- Já os números negativos devem ser descritos assim, ex: -12;

Checando número inteiro

- Podemos validar se um dado é inteiro com a função **is_int()**;
- Caso um número seja inteiro, será retornado **true** (um outro tipo de dado);
- Caso não seja, receberemos um retorno de **false** (tipo de dado também);
- Precisamos utilizar uma estrutura **if** para validar o valor;

Números decimais (floats)

- Os floats são todos os números com **casas decimais**;
- Como o padrão universal é da língua inglesa, temos a separação de casas **com . e não ,**
- Exemplos de floats: 2.123, 0.04, -12.8

Checando se é float

- Podemos utilizar a função **is_float()** para verificar se um dado é um float;
- A função recebe um **valor como parâmetro**;
- Novamente receberemos **true or false**, dependendo do dado enviado;
- Precisamos utilizar uma estrutura **if** para validar o valor;

Textos (strings)

- Os textos são conhecidos como **strings**;
- Em PHP podemos escrever textos em **aspas simples ou duplas**, não há diferença para texto puro;
- As aspas duplas **interpretam variáveis**;

Checando se é string

- Podemos utilizar a função **is_string()** para verificar se um dado é uma string;
- A função recebe um **valor como parâmetro**;
- Novamente receberemos **true or false**, dependendo do dado enviado;
- Precisamos utilizar uma estrutura **if** para validar o valor;

Booleanos

- O boolean é um tipo de dado que só possui **dois valores**:
- **True** - verdadeiro;
- **False** - falso;
- Alguns valores são considerados como falsos: 0, 0.0, “0”, [ ], NULL;

Checando se é booleano

- Podemos utilizar a função **is_boolean()** para verificar se um dado é um boolean;
- A função recebe um **valor como parâmetro**;
- Novamente receberemos **true or false**, dependendo do dado enviado;
- Precisamos utilizar uma estrutura **if** para validar o valor;

Arrays (conjunto, lista)

- O array é um tipo de dado que serve para **agrupar um conjunto de valores**;
- Podemos inserir **qualquer tipo de dado** na lista;
- A sintaxe é: [1, 2, 3, 4, 5];
- Sempre entre **[ ]**, dados separados por **,**
- Veremos arrays em mais detalhes futuramente, é uma estrutura de dados muito importante e muito utilizada;

Array Associativo

- O **array associativo** é basicamente um array, porém com **chave e valor**;
- A **estrutura base é a mesma**, mas vamos construir dessa maneira:
- $arr = [‘nome’ => ‘Matheus’, idade => 29]
- Chave entre aspas, seta para apontar o valor e valor;

Exercício 7

- Crie um arquivo PHP;
- Crie um array associativo com características de uma pessoa;
- Desafio: faça um if checando se ela é maior de idade e imprima uma mensagem, caso seja;

Objetos

- PHP possui o paradigma de **orientação a objetos**;
- Podemos criar **classes e objetos**, e o objeto é considerado um tipo de dado;
- Objetos possuem **métodos** que são suas ações e **propriedades** que são suas características;
- Veremos objetos em maiores detalhes futuramente no curso;

Null

- O tipo de dado Null tem apenas um valor, o **NULL**;
- Um caso de uso do Null seria checar se uma variável tem ou não valor;
- Podemos checar se um valor é null com **is_null()**;

Variáveis

Introdução da seção

Sobre as variáveis

- São a forma que temos para **declarar um valor e salvá-lo na memória**;
- Uma variável em PHP tem o **$** na frente do seu nome;
- Ex: $nome = “Matheus”;
- Podemos salvar **qualquer tipo de dado**;
- Podemos **alterar o valor de uma variável** no decorrer do programa;
- Podemos imprimir o valor de uma variável com **echo**;

Variável de variável

- Podemos criar uma **variável por meio do nome de outra variável**, com um valor diferente;
- O símbolo para esta função é o **$$**

  $x = “teste”;

  $$x = 5;
- Após a execução do código, a variável teste (conteúdo de $x), será criada com o valor 5;

Variável por referência

- Podemos criar uma **variável com referência a outra**;
- O símbolo é **=&**;
- Se mudamos a variável de referência a referenciada muda o valor e ao contrário também gera a mudança;

```
$x = 2;

$y =& $x;
```

Escopo

- Como em outras linguagens em PHP também temos escopo de variáveis;
- **Local:** variável declarada em uma função;
- **Global:** variáveis declaradas fora de funções;
- **Static:** variável declarada dentro da função, porém o seu valor permanece salvo entre chamadas da função;
- **Parâmetros de função:** variáveis passadas para uma função, podendo ser utilizadas ao longo da mesma;

Variável Local

- A variável local tem seu **escopo definido dentro de uma função**;
- Ela **não é acessível** fora da mesma;
- O **seu valor sempre é resetado** quando a função é finalizada;
- Obs: veremos funções em detalhes futuramente;

Variável Global

- A principal característica da variável global **é ser declarada fora de funções**;
- Por comportamento padrão **não são acessíveis dentro de funções**;
- Precisamos utilizar a palavra **global** para isso;
- Essa função da variável global não ser acessível dentro de funções, previne muitos problemas no software;

Variável Estática

- A variável estática é declarada com a instrução **static**;
- O valor da mesma **é mantido e alterado a cada execução de uma função**;
- É interessante este comportamento pois as variáveis de **escopo local sempre são resetadas**;

Parâmetros de função

- Os parâmetros de função **também são considerados tipos de variáveis**;
- Este recurso nos ajuda a **criar funções com valores dinâmicos**;
- Podendo **alterá-los a cada invocação** da mesma;
- Podemos passar mais de um parâmetro para uma função;

Expressões e Operadores

Introdução da seção

O que é uma expressão?

- Uma **instrução de código** que será avaliada **e resultará em um valor**;
- Uma **simples impressão de um texto** é uma expressão;
- **Uma soma ou operação matemática mais complexa** também;
- Na programação realizaremos **diversas expressões** durante nosso código, para formar nosso software;

O que é um operador?

- Operadores são **recursos que utilizamos para compor expressões mais complexas**;
- Alguns deles: +, -, **, /, ++, >, <, >=, <= e etc…
- Estas operações podem matemáticas ou até mesmo comparações;
- A ideia principal é que um **novo valor é gerado** ou também um **booleano pode ser retornado**;

Ordem dos operadores

- O PHP e as linguagens de programação **executam os operadores na mesma ordem que na matemática**;
- Ou seja em: 2 + 2 * 4, teremos o resultado de **10**;
- Pois **a multiplicação é avaliada antes da soma**;
- Mesmo que a primeira operação seja soma;
- Podemos utilizar **( )** para separar operações;

Mudança de tipo implícito

- O PHP em certas operações **muda o tipo de dado** de forma implícita;
- Por exemplo 5 / 2 = 2.5 (gera um **float**)
- E 5 . 5 resulta em 55 (gera uma **string**, o . é o operador de concatenação)
- Por isso, temos que **tomar cuidado** com algumas expressões que podem gerar resultados indesejados;
- Este recurso é chamado de **auto cast**;

Operadores aritméticos

- Temos os **operadores básicos** da matemática em PHP;
- Soma: +
- Subtração: -
- Divisão: /
- Multiplicação: *

Operador de módulo

- O operador de módulo é inserido no código pelo símbolo de **%**
- Sua função é realizar **uma divisão**;
- Mas como resultado ele **apresenta apenas o resto** da mesma;

Exponenciação

- Podemos realizar o cálculo de potência com o símbolo **;
- Exemplo: 5 ** 2;
- Desta maneira teremos o resultado de **5 elevado a 2**;

Operador de concatenação

- Em PHP podemos concatenar valores com **.** (ponto)
- Concatenar é o ato de **juntar vários textos e/ou números** em apenas uma string;
- **Não há limites** de quantas expressões podem ser concatenadas;

Auto incremento e auto decremento

- Podemos incrementar um valor ou decrementar com os operadores: **++** e **--**;
- Exemplo: $n++ ou $x--
- Onde n e x são variáveis, e **terão seus valores alterados com +1 e -1**;
- Estes operadores são muito utilizados em **estruturas de repetição**;

Operadores de comparação

- As operações com operadores de comparação resultarão em true or false;
- Igualdade: **==**
- Idêntico a: **===**
- Diferença: **!=**
- Não idêntico a: **!==**
- Maior e maior ou igual a: **> e >=**
- Menor e menor ou igual a: **< e <=**

Operador de igualdade

- Com o **operador de igualdade** verificamos se um valor é igual ao outro;
- O símbolo é: **==**
- Exemplo: 5 == 4 # false
- Exemplo: 3 == 3 # true

Operador idêntico a

- Com o **operador idêntico a** verificamos se um valor é igual ao outro, avaliando o seu tipo também;
- O símbolo é: **===**
- Exemplo: 5 === 5 # true
- Exemplo: 3 === “3” # false

Operador de diferença

- Com o **operador de diferença** verificamos se um valor é diferente de outro;
- O símbolo é: **!=**
- Exemplo: 5 != 5 # false
- Exemplo: 10 != 5 # true

Operador não idêntico a

- Com o **operador não idêntico a** verificamos se um valor é diferente de outro, avaliando o seu tipo também;
- O símbolo é: **!==**
- Exemplo: 5 !== 4 # false
- Exemplo: 3 !== “3” # true

Operador maior e maior ou igual

- Com o **operador maior que** verificamos se um valor é maior que outro;
- O símbolo é: **>**
- Exemplo: 5 > 4 # true
- Com o **operador maior ou igual a** verificamos se um valor é maior ou igual a outro;
- O símbolo é: **>=**
- Exemplo: 5 >= 5 # true

Operador menor e menor ou igual

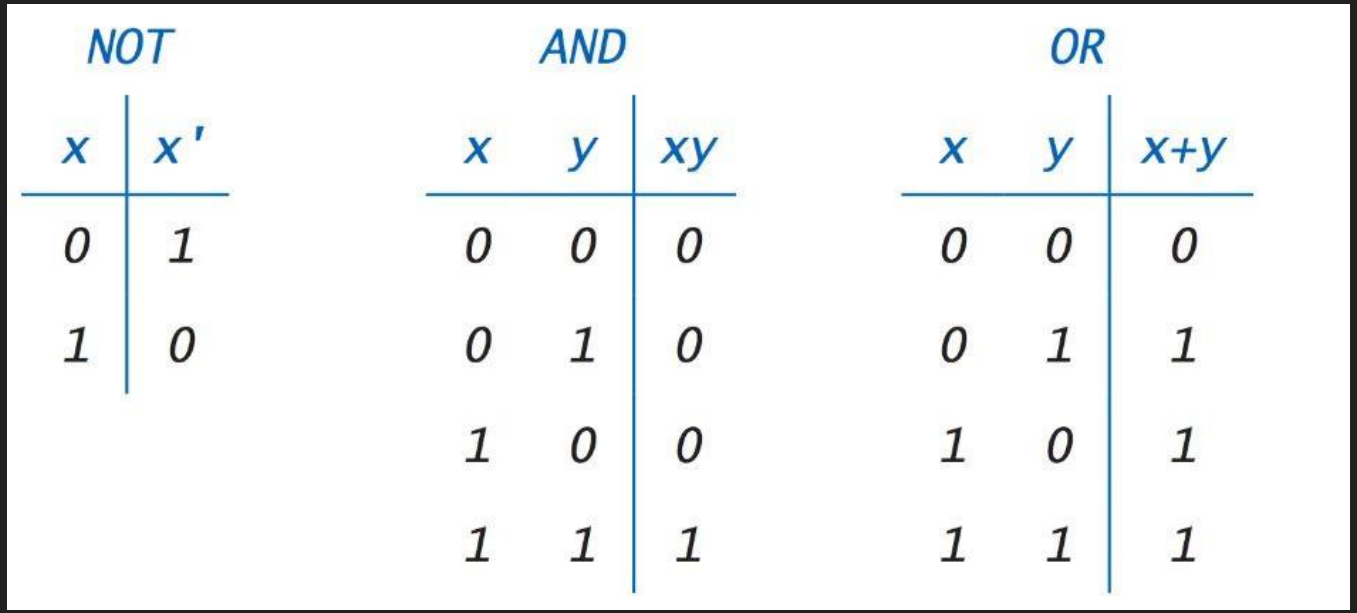
- Com o **operador menor que** verificamos se um valor é menor que outro;
- O símbolo é: **<**
- Exemplo: 5 < 4 # false
- Com o **operador menor ou igual a** verificamos se um valor é menor ou igual a outro;
- O símbolo é: **<=**
- Exemplo: 11 <= 12 # true

Operadores lógicos

- Com os operadores lógicos podemos **encadear várias comparações**;
- Operador AND: **&&**
- Operador OR: **||**
- Operador NOT: **!**

Tabela verdade

- Com a tabela verdade, temos um resumo dos operadores lógicos:



NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

fonte: https://introcs.cs.princeton.edu/java/home/

Operador lógico AND

- Os operadores lógicos em conjunto dos de comparação **também retornam uma booleano** (true ou false);
- No caso de **AND** temos **true** apenas quando **as duas comparações são verdadeiras**;
- Símbolo: &&
- Ex: 5 > 2 && 10 < 100 # true

Operador lógico OR

- O operador lógico OR resulta em **verdadeiro** caso **qualquer um dos lados da operação seja verdadeiro**;
- E só resulta em **falso** caso os **dois lados sejam falsos**;
- Símbolo: ||
- Exemplo: 5 > 15 || “teste” == “teste” # true

Operador lógico NOT

- O operador lógico **NOT** apenas **inverte o resultado booleano** de uma operação, se é true vira false e se é false vira true;
- Símbolo: **!**
- Exemplo: !true # false
- Exemplo: !(5 > 2) # false

Operadores de conversão (cast)

- Com os **operadores de conversão** podemos **forçar uma variável ser de um determinado tipo**;
- **Nem todos são úteis**, os mais utilizados são para converter uma string em número;
- Operadores: int, bool, float, string, array, object e unset;
- Exemplo: $a = (float) “5.34243” # string é convertida para float

Operadores de atribuição

- Com estes operadores podemos **atribuir valor a uma variável**;
- O mais conhecido é o **=**, porém temos algumas variações do mesmo;
- Operadores: **+=, -=, /=, *= e %=**;
- Cada um destes fará uma **operação antes da atribuição**;

Operador ternário

- Este operador constitui uma **estrutura de condição resumida**;
- **Na maioria dos casos** vamos optar por if/else;
- Porém em situações simples podemos utilizar o ternário;
- Exemplo: 5 > 2 **?** echo “5 é maior que dois” **:** echo “5 é menor que 2”
- A primeira interrogação vem **antes da comparação**;
- E o : é utilizado para uma segunda situação, caso a primeira seja falsa;

Estrutura if

- A estrutura **if** checa se uma expressão é verdadeira;
- Podemos incluir **operadores lógicos nas expressões**;
- Exemplo: if(expressão) { // bloco de código }

Estrutura else

- A estrutura **else** pode executar um outro bloco de código, isso acontece quando a expressão de if é falsa;
- Em else **não inserimos expressões**;
- Exemplo:

```
if (exp) {

} else {

}
```

If aninhado

- Podemos também inserir **um if dentro de outro**;
- Neste caso o segundo bloco precisa apenas ficar dentro do primeiro if;
- Exemplo:

```
if(exp) {
        if(exp) {
        }
}
```

Else if

- Com o **else if** podemos criar um **novo bloco de expressão**;
- Este bloco **será executado caso o primeiro if seja falso**;
- O else if fica entre o if e o else;
- Exemplo:

```
if(exp) {

} else if(exp) {

}
```

Switch

- O **switch** é uma estrutura de condição, que pode substituir o if em alguns casos;
- Podemos adicionar a instrução **break**, para ele não ser mais executado;
- Há a possibilidade também de adicionar a instrução **default**, que é executada caso nenhuma condição seja satisfeita;

Estruturas de Repetição

Introdução da seção

While

- O while é uma **estrutura de repetição**, pode executar um código n vezes;
- Até **satisfazer a sua condição**;
- Geralmente é necessário um **contador** para atingir a condição;
- Exemplo:

```
while(condicao) {
        código
}
```

Saindo de loop

- Podemos sair de um loop while **antes do seu fim**;
- Para isso é necessário adicionar a instrução **break**;
- Após interpretada, **o loop será automaticamente finalizado**;
- Geralmente inserimos esta instrução em uma **condição if**;

Loop dentro de loop

- Como nas estruturas de if, podemos **adicionar um loop dentro de outro**;
- O **contador deve ser único**, para que um loop não afete o outro;
- O loop interno será executado tantas vezes quanto o loop externo for;
- E em cada uma das suas execuções, serão passadas todas as suas etapas;

A instrução continue

- O **continue** pula uma execução do loop;
- Ou seja, quando o interpretador encontrar esta instrução, **a próxima etapa do loop será executada**;
- Novamente costumamos aplicar **dentro de uma estrutura de condição**;

Do while

- O **do while** é também uma estrutura de repetição;
- Porém **menos utilizada** que o while;
- A sintaxe é invertida, veja um exemplo:

```
do {
        codigo
} while(condicao);
```

A estrutura for

- A **for** é com certeza a estrutura de repetição mais utilizada;
- Sua **sintaxe é mais organizada**, em apenas uma linha e aparenta ser mais difícil, ao primeiro olhar;
- Exemplo:

```
for(contador; condicao; incremento) {
        codigo
}
```

Loop infinito

- O **loop infinito** é um erro que pode ser ocasionado quando uma estrutura de repetição não tem uma condição de término que seja possível;
- **Por exemplo:** x > 10 e a variável de referência tem um decremento, não um incremento;
- Isso vai fazer **o software travar**, e pode ser um grande problema caso usuários estejam acessando o mesmo;

Foreach

- A **foreach** também é uma estrutura de repetição;
- Porém **ela é orientada a um array**, devemos utilizar um para que a estrutura repita em todos os elementos do mesmo;
- Exemplo:

```
foreach($array as $item) {
        codigo
}
```

Inclusão de código

Introdução da seção

Include

- Com o **include** inserimos um arquivo de PHP, ou até mesmo um HTML, em outro;
- Podendo assim **utilizar tudo que está declarado no arquivo incluído**;
- O include **não gera erro fatal** se o arquivo não existir, e sim um **warning**;
- Exemplo:

  include “arquivo.ext”

Require

- Com o **require** inserimos um arquivo de PHP, ou até mesmo um HTML, em outro;
- Podendo assim **utilizar tudo que está declarado no arquivo incluído**;
- O require **gera erro fatal** se o arquivo não existir, parando o script;
- Exemplo:

  require “arquivo.ext”

include_once require_once

- Os dois **funcionam da mesma maneira** que require e include;
- Porém **impedem que o mesmo arquivo seja adicionado mais de uma vez** na página;
- Este **pode ser o método mais indicado** quando estamos montando templates com PHP;

Short tags

- A **short tag** é uma funcionalidade para adicionar código PHP em uma página;
- Este recurso **depende de uma configuração do servidor** para funcionar;
- Por isso é desencorajado seu uso, **pode ser que o código não funcione**;
- Ex:

  <? echo “teste”; ?>

Exibição de conteúdo

- Com uma **técnica semelhante ao short tags,** podemos exibir conteúdo sem o echo;
- Ótima estratégia para resumir as chamadas PHP **apenas para exibição de valores**;
- Ex:

  <?= “teste”; ?>

Inserindo PHP ao HTML

- Como abordado nas seções iniciais, **esta é uma das principais funcionalidades PHP**;
- Podemos **inserir código dinâmico entre nossas tags**;
- As extensões para este tipo de arquivo podem ser de **.php ou .phtml**;
- Ex:

```
<h1><?= $titulo ?></h1>
```

Funções

Introdução da seção

O que são funções?

- São **blocos de códigos** que **possuem nomes**;
- **Realizam uma ação** e **podem ser reaproveitadas** (chamadas novamente) ao longo do programa;
- Podemos passar parâmetros para funções, que moldam a sua execução;
- A criação de funções **reduz a duplicidade** de código;
- E também **melhora a manutenção** do mesmo;
- O PHP possui **diversas funções prontas**, que podemos utilizar;

Chamando funções

- Para chamar uma função basta colocar o seu **nome e abrir e fechar parênteses**;
- Exemplo: **funcaoTeste()**
- Algumas funções **exigem parâmetros**;
- O ato de chamar uma função também é conhecido como **invocar**;
- O PHP tem diversas funções para utilizarmos no nosso código, exemplos: strlen, strtoupper, strtolower, print_r, var_dump

OBRIGADO!