
# Curso de PHP

## do Básico ao Avançado

# 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;

# Exercício 1

- Crie um arquivo PHP;
- Este arquivo deve ter uma instrução que imprime o seu nome;
- Execute o arquivo;

# 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;

# Como tirar o máximo proveito

- Faça todos os exemplos com o editor aberto e **codifique junto**;
- Crie **seus próprios exemplos** com o que foi aprendido na aula;
- Crie também **projetos pessoais**;
- Faça **anotações** dos pontos mais importantes;
- **Dica bônus:** assista a aula e depois execute;


# Introdução

## Conclusão da seção


# Conceitos básicos

## Introdução da seção

# 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 */**

# Exercício 2

- Crie um arquivo PHP;
- Neste arquivo descreva características básicas de uma pessoa ou objeto utilizando comentários;
- Pelo menos três, em linhas separadas;

# 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;


# Conceitos básicos

## Conclusão da seção


# 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;

# Exercício 3

- Crie um arquivo PHP;
- Imprima três números inteiros;

# 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;

# Exercício 4

- Crie um arquivo PHP;
- Imprima três floats;
- Utilize a função is_float em um deles;

# 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;

# Exercício 5

- Crie um arquivo PHP;
- Imprima textos com aspas duplas e também com aspas simples;

# 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 6

- Crie um arquivo PHP;
- Crie um array com características de um carro;
- Imprima duas características;

# 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()**;


# Tipos de dados

## Conclusão da seção


# 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**;

# Exercício 8

- Crie um arquivo PHP;
- Crie três variáveis com tipos de dados diferentes;
- Imprima estas variáveis;

# Exercício 9

- Crie um arquivo PHP;
- Crie duas variáveis com números;
- Cria uma terceira com a soma destes dois números;
- Lembrando: a soma pode ser feita com o símbolo +;
- Ex: 2 + 4

# 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;


# Variáveis

## Conclusão da seção


# 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;

# Exercício 10

- Crie um arquivo PHP;
- Crie uma operação que utiliza subtração ( - ), divisão ( / ) e multiplicação
- Armazene todos os valores em variáveis;
- Imprima o resultado final na tela;

# 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**;

# Exercício 11

- Crie um arquivo PHP;
- Teste a expressão “5” * 12;
- Utilize a função **gettype()** com o resultado como parâmetro para checar o tipo resultante da operação;

# Operadores aritméticos

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

# Exercício 12

- Crie um arquivo PHP;
- Crie uma operação com cada um dos operadores básicos;
- Cada operação deve estar em uma variável diferente;
- Imprima cada uma das etapas;
- Ex: soma -> multiplicação -> divisão -> subtração;

# 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;

# Exercício 13

- Crie um arquivo PHP;
- Teste o operador de resto em duas divisões;
- Uma não exata e outra exata;

# 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;

# Exercício 14

- Crie um arquivo PHP;
- Crie uma variável saudação, nome e outra de sobrenome;
- Imprima com echo a concatenação de saudação, nome e sobrenome;

# 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

# Exercício 15

- Crie uma operação que retorne falso com igualdade;
- Crie uma operação que retorne verdadeiro com igualdade;

# 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

# Exercício 16

- Insira o valor 5 em uma variável, e o valor 3 em outra;
- Teste os operadores de: igualdade, diferença, idêntico e não idêntico;

# 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

# Exercício 17

- Verifique as seguintes operações com AND;
- 15 > 5 AND “João” === “João”
- “teste” > 5 AND 1
- 2 == 3 AND 5 >= 3

# 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

# Exercício 18

- Verifique as seguintes operações com OR;
- 12 < 5 OR “João” === “João”
- 1 > 5 OR 1
- 20 === “20” AND 51 >= 31

# 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

# Exercício 19

- Converta os seguintes dados para int com o operador de cast;
- “testando”
- 12.9
- true
- [1, 2, 3]
- E veja os resultados

# 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;

# Exercício 20

- Atribua dois números a variáveis distintas;
- Faça uma comparação de menor ou igual com o operador ternário;
- Imprima resultados para ambas as possibilidades;


# Expressões e Operadores

## Conclusão da seção


# Estruturas de Controle

Introdução da seção

# Estrutura if

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

# Exercício 21

- Faça as seguintes verificações em estruturas if:
- 5 é maior que 2?
- Matheus é diferente de Pedro
- 12 é menor ou igual a 11
- Você deve inserir os valores em variáveis;

# Exercício 22

- Crie variáveis que recebem idades;
- Cheque se as idades são maiores ou iguais a 18;
- Se sim, imprima uma mensagem que a pessoa é maior de idade;

# 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 {
}
```

# Exercício 23

- Complemente o exercício 22;
- Insira um else com uma mensagem para as pessoas que são menores de idade;

# Exercício 24

- Crie algumas variáveis com tipos de dados diferentes: string, int e boolean, por exemplo;
- Cheque se a variável é um inteiro;
- Caso sim, apresente uma mensagem confirmando o tipo de dado;
- Caso não, apresente outra mensagem;

# Exercício 24 b

- Crie uma variável que recebe um peso;
- Caso seja maior que 80, imprima a mensagem que está pesado demais;
- Se não, prima “Peso dentro do limite”;

# 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) {
        }
}
```

# Exercício 25

- Crie variáveis com números e outras com string;
- Faça um if checando se é um número;
- Caso for, atribua a multiplicação deste número por 2 em outra variável;
- E crie um outro if, que checa se o novo número é maior que 100;
- Se sim, imprima uma mensagem;

# 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) {
}
```

# Exercício 26

- Crie uma variável que recebe uma velocidade de um carro;
- Depois crie uma estrutura if que verifica essa velocidade;
- Se a velocidade for menor que 40, imprima que o motorista está na velocidade correta;
- Se igual a 40, imprima uma mensagem para o motorista tomar cuidado;
- Se for maior de 40, imprima uma mensagem de multa;

# Exercício 26 b

- Crie uma variável que recebe um peso de uma pessoa;
- E outra que recebe uma altura;
- Calcule o IMC (peso / altura ²)
- Faça 3 checagens;
- Menor que 20 = subpeso;
- 20 a 25 = normal;
- Maior que 25 = sobrepeso;

# 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 Controle

## Conclusão da seção


# 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
}
```

# Exercício 27

- Crie um array com alguns valores (pelo menos 10) de tipos de dados diferentes;
- Faça um loop while para exibir apenas os dados que são strings;

# 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**;

# Exercício 28

- Crie um Loop que vai até o número 30;
- O contador deve iniciar como 4;
- Faça incrementos de 2 em 2 no contador;
- Utilize o break para parar o loop quando chegar no número 24;

# 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**;

# Exercício 29

- Crie um array com valores inteiros de 10 a 100, com incremento de 10;
- Aplique um loop neste array;
- Quando entrar os valores 30 ou 40, pule para a próxima execução;

# 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
}
```

# Exercício 30

- Crie um array com números de 1 a 20;
- Crie um loop for para este array;
- Imprima apenas os pares;

# Exercício 30 b

- Crie um array de 1 a 10;
- Utilize um loop for para criar este array;
- Dica: você pode utilizar o método array_push(arr, elemento) para inserir um elemento em um array;
- Imprima o array criado com print_r;

# Exercício 30 c

- Crie um array de 10 a 20 com for;
- Faça um loop em cima do array criado dinamicamente;
- Imprima apenas os números ímpares;

# 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
}
```


# Estruturas de Repetição

## Conclusão da seção


# 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>
```


# Inclusão de código

## Conclusão da seção


# 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

# Exercício 31

- Crie um array com strings;
- Utilize a função implode no array;
- Primeiro argumento: “,”
- Segundo argumento: o seu array
- Atribua a invocação da função a uma variável
- Exiba o resultado

# Definindo uma função

- Para definir uma função vamos utilizar o nome **function**;
- Depois precisamos **escolher um nome**;
- **Abrir e fechar parênteses** e adicionar o bloco de multi linha com chaves;
- Este bloco é o **corpo da função**;

```
function nome() {
    // codigo
}
```

# Exercício 32

- Crie uma função;
- Define três variáveis numéricas dentro dela;
- Exiba a multiplicação destes números com um echo;

# Exercício 32 b

- Crie uma função;
- Define uma variável nome e sobrenome;
- Imprima os valores concatenados;

# Função com parâmetro

- Podemos passar **parâmetros** para a função;
- Estes **parâmetros são como variáveis**, que são utilizados dentro da função para moldar a sua execução;
- **Não há número máximo** de parâmetros;

```
function teste(param, param2) {
    // codigo
}
```

# Exercício 33

- Crie uma função;
- Ela deve receber um parâmetro de nome e idade;
- Imprima “Olá eu sou o NOME e tenho X anos”;

# Exercício 34

- Crie uma função que verifica se um número é par ou ímpar;
- Se for par imprima uma mensagem;
- Se for ímpar imprima uma mensagem;

# Retorno de funções

- Normalmente funções retornam algo, para isso utilizamos a instrução **return**;
- O objetivo é **armazenar o valor de retorno em uma variável** e utilizá-lo posteriormente no código;

```
function x(a, b) {
        return algumaCoisa;
}
```

# Exercício 35

- Crie uma função que recebe um número;
- Retorne o valor deste número ao quadrado;

# Relembrando escopo e funções

- Nas funções temos um escopo específico chamado de **local**, onde as suas variáveis são exclusivamente delas;
- Podemos utilizar as variáveis globais com a instrução **global**;
- E também há o **static**, onde podemos manter um valor após a execução de uma função, o que normalmente é resetado;

# Exercício 36

- Crie uma função que recebe um array de números;
- Crie um novo array com apenas os números que são maiores que 7;
- Retorne este novo array e imprima na tela;

# Parâmetros default

- Podemos passar parâmetros que já possuem um valor pré-determinado;
- Então caso você não passe este parâmetro, o valor **default** entra em cena;
- A **função será executada normalmente** com o valor definido;

```
function teste($a = “padrão”) {
        // codigo
}
```

# Exercício 37

- Crie uma função chamada defineCorCarro;
- Onde há um parâmetro chamado cor, com valor default de vermelha;
- Retorne a cor do carro;
- Imprima o retorno tanto com parâmetro default, como também definindo a cor;

# Descobrindo argumentos

- No PHP temos duas funções interessantes para aprender mais sobre funções;
- **func_get_arg** = retorna uma lista com os argumentos de uma função;
- **func_num_args** = retorna o número de argumentos de uma função;

# Exercício 38

- Crie uma função que recebe um array de itens de supermercado;
- Retorne este array em forma de string, separado em vírgulas;

# Retornando múltiplos valores

- Caso seja necessário retornar vários valores em uma função, podemos **formar um array para retorno**;
- E então **acessar os índices de forma isolada** com a nova variável que contém o retorno;

```
function teste() {
    return [“a”, 10, true];
}
```

# Depuração de valores

- Utilizamos duas funções para verificar dados formatados:
- **print_r** e **var_dump**;
- As duas apresentam os dados de forma semelhante;
- Porém var_dump exibe de uma forma “**human readable**”, traduzindo seria algo como “para humanos lerem”;


# Funções

## Conclusão da seção


# Strings

## Introdução da seção

# Interpolando variáveis

- Podemos **interpolar variáveis em strings** de duas formas;
- Utilizando **aspas duplas e colocando a variável** e também **com chaves e o nome da variável**;
- **Não há diferença** em ambas as formas;

  “Interpolando a variável **$teste**”

  “Interpolando a variável **{$teste}**”

# Valores de escape

- Podemos utilizar alguns valores que **executam funções especiais em strings**;
- Precisamos utilizar **aspas duplas**;
- Exemplos: **\n** = nova linha;
- **\t** = tab;
- \\ = barra invertida;
- **\$** = sinal de dólar;

# Exercício 39

- Crie uma função que recebe características de algum objeto como argumento (carro, sofá, cafeteira), em array associativo;
- O array deve conter nome => preco;
- Retorne apenas os itens que custam mais que R$10;
- Imprima o retorno;

# Função print

- A função **print** tem a funcionalidade semelhante de echo;
- Pode **imprimir uma string que foi passada como argumento**;
- Exemplo:

  print(“testando”);

# Função printf

- A função **printf** tem a funcionalidade semelhante a print;
- Porém podemos imprimir valores de forma dinâmica utilizando o símbolo **%**;
- Exemplo:

  print(“Número %d”, 1);

# Tamanho da string

- Utilizando a função **strlen** com uma string como parâmetro, vamos receber o tamanho da string;
- Ou seja, a **quantidade de caracteres** da mesma;
- Exemplo:

  strlen($string);

# Percorrendo uma string

- Podemos percorrer cada um dos **caracteres de uma string**;
- Para isso vamos utilizar uma **estrutura de repetição**;
- E o método **strlen**, para saber o **número de caracteres**;
- Com isso podemos iterar pela string completa;

```
for($x = 0; $x < strlen($str); $i++) {
        // codigo
}
```

# Exercício 40

- Percorra a string: O rato roeu a roupa do rei de Roma, a partir de um loop;
- Imprima o número de letras “a” desta string;

# Limpando uma string

- Podemos remover os espaços em branco de uma string com funções de PHP;
- **trim** - limpa espaços antes e depois da string;
- **ltrim** - limpa espaços da parte inicial da string;
- **rtrim** - limpa espaços da parte final da string;
- Desta forma conseguimos remover os espaços desnecessários inseridos pelos usuários;

# Alterando o case

- Podemos alterar as strings para maiúsculas ou minúsculas com funções de PHP;
- **strtolower** - todas as letras minúsculas;
- **strtoupper** - todas as letras maiúsculas;

# Alterando o case de palavras

- Podemos alterar o case apenas das palavras com funções de PHP;
- **ucfirst** - primeira letra da string em maiúscula;
- **ucwords** - primeira letra de cada palavra em maiúscula;

# Exercício 41

- Transforme a string “este item está em promoção”;
- Em “Este item está em PROMOÇÃO”;
- Obs: você pode separar as strings, mas não pode escrever em caixa alta ou baixa manualmente, só com funções;

# Removendo tags HTML

- Podemos remover as tags de HTML de uma string com a função **strip_tags**;
- Geralmente para salvar dados no banco removemos as tags;

# Resgatando uma parte da string

- Com a função **substr**, podemos resgatar apenas uma parte da string;
- Ex: substr(**str**, **início**, **fim);**
- **Str** é a string que vamos procurar algo;
- **Início** é o índice inicial da palavra ou texto;
- **Fim** é o índice final da palavra ou texto;

# Exercício 42

- Na frase “Cadê o meu queijo? Ele estava aqui em cima”
- Resgate apenas a palavra **queijo**;

# String reversa

- Podemos com PHP inverter uma string, a função **strrev** realiza esta ação;
- Ela recebe a string que será invertida como parâmetro;

# String reversa

- Podemos com PHP inverter uma string, a função **strrev** realiza esta ação;
- Ela recebe a string que será invertida como parâmetro;

# Repetição de string

- Com a função **str_repeat** você pode repetir n vezes uma determinada string;
- O primeiro argumento é a string que será repetida;
- O segundo é o número de repetições;

# String para array

- Podemos converter uma string em array com a função **explode**;
- Passamos primeiro o separador como argumento;
- Depois a string que vai ser convertida;

# Exercício 43

- Converta a seguinte string para array:
- carro - navio - helicóptero - barco - jangada

# Array para string

- Podemos converter um array em string com a função **implode**;
- Passamos primeiro o separador como argumento;
- Depois a string que vai ser convertida;

# Exercício 44

- Converta a seguinte array para uma string:
- [“O”, “PHP”, “é”, “muito”, “legal”]

# Encontrando a primeira ocorrência

- Com a função **strpos** podemos encontrar algum texto na string;
- Se recebermos **algum valor** é que o texto foi encontrado, e este valor é o índice inicial;
- Se for retornado **false**, o texto não está na string;

# Encontrando a última ocorrência

- Com a função **strrpos** podemos encontrar a última ocorrência de um texto na string;
- Se recebermos **algum valor** é que o texto foi encontrado, e este valor é o índice inicial;
- Se for retornado **false**, o texto não está na string;

# Retornando o resto da string

- Com a função **strstr** podemos encontrar um texto em uma string;
- Se algo for encontrado, a função vai retornar o resto da string após o texto encontrado;
- Se não encontrar nada retorna false;

# Decompor uma URL

- Com a função **parse_url** podemos decompor uma URL;
- Vamos receber um array com todas as partes que a URL tem;
- Alguns elementos que podem ser retornados são: protocolo, host, parâmetros;


# Strings

## Conclusão da seção


# Arrays

## Introdução da seção

# Adicionando dados a um array

- Podemos **criar novos índices** com dados em um array;
- Basta por o nome do array com o novo índice em colchetes e atribuir um valor;
- Ex: $arr[1] = “teste”;
- E em arrays associativos basta utilizar o nome da nova chave com a atribuição de valor;

# Adicionando valor ao fim do array

- Podemos adicionar valor ao fim de um array utilizando a **atribuição sem determinar um índice**;
- Então o valor atribuído será enviado para o último e novo índice do array;
- Exemplo: $arr[] = 5;

# Criar array rapidamente

- Podemos utilizar a função **range** para criar um array de forma rápida;
- Exemplo: range(1, 10);
- Um array de 1 a 10 será criado, podemos atribuir este valor a uma variável;

# Exercício 45

- Crie um array com a função range de 10 a 45;
- Imprima todos os números com uma soma de 6;
- Se passar de 30 a soma, imprima também que o número é muito alto;

# Número de elementos

- Podemos obter o número de elementos de um array com a função **count**;
- Basta passar o array como argumento;
- Um inteiro será retornado;

# Array multidimensional

- Quando inserimos arrays dentro de arrays formamos um **array multidimensional**, também conhecido como matriz;
- Para acessar este tipo de array também utilizamos índices, acessando o externo e depois os internos;
- Ex: $arr[1][0] => Primeiro elemento do segundo array;

# Exercício 46

- Crie um array multidimensional com 3 arrays que tem 4 elementos cada;
- Imprima todos os elementos de cada um dos arrays;
- Imprima também quando está mudando de array;

# Criando muitas variáveis

- Podemos criar muitas variáveis com base em um array;
- Para isso vamos utilizar a função **list**;
- Ex: list($nome, $idade, $profissao) = $pessoa;

# Exercício 47

- Crie um array com os seguintes valores: jaguar, 3.0, azul, 18, Teto solar, automático;
- Chame este array de carro;
- Crie variáveis com base neste array;

# Resgatando elementos de array

- Com a função **array_slice** podemos resgatar uma faixa de elementos de um array;
- Passamos 3 parâmetros: o array, índice inicial e quantos elementos queremos resgatar a partir do índice;

# Dividindo arrays

- Podemos dividir um array grande em diversos arrays de número de elementos iguais;
- Vamos utilizar a função **array_chunk**;
- Passamos o array como argumento e também o número de elementos que cada array deve ter;

# Chaves e valores

- Com a função **array_keys** recebemos um array apenas com as chaves de um array;
- Com a função **array_values** recebemos um array com apenas os valores de um array;

# Verificando se valor existe

- Com a função **array_key_exists** podemos verificar se há um valor em uma respectiva key de um array;
- Podemos fazer essa checagem em um if;
- Ex: array_key_exists(“nome”, $arr)
- Outra função que podemos utilizar para este fim é a **isset**;

# Removendo elementos

- Podemos remover elementos de um array com a função **array_splice**;
- Passamos como parâmetro o array, índice inicial e quantos elementos queremos remover;
- Ex: array_splice($arr, 2, 1) => A partir do índice 2, remove 1 elemento;

# Exercício 48

- Crie um array com os valores: batata, maçã, pera, feijão, arroz;
- Remova pera e feijão;

# Criando variáveis com extract

- Com a função **extract** podemos criar variáveis rapidamente de arrays associativos;
- O nome da chave será o nome da variável;
- Se houver uma variável já criada com o nome da chave, a mesma será sobrescrita;s

# Criando array com compact

- Com a função **compact** podemos criar um array a partir de variáveis;
- Passamos para a função o nome das variáveis em string;
- E então um novo array é criado, podemos atribuir a uma variável;

# Exercício 49

- Crie variáveis com característica de algum objeto ou animal;
- Depois crie um array com compact com estas mesmas variáveis;
- Faça um loop no array e imprima os valores;

# Foreach e arrays

- Anteriormente vimos a estrutura **foreach** com arrays, podemos iterar facilmente com ela;
- Utilizando a notação de **chave => valor**, temos acesso rápido também a arrays associativos;
- Exemplo:

  foreach($itens as $key => $value) { }

# Exercício 50

- Crie um array associativo com nomes e idades;
- Imprima estes dados em uma tabela de HTML;
- Dica: utilize as tags do elemento table

# Reduce em arrays

- A função **array_reduce** tem como objetivo reduzir um array a apenas um valor;
- Podemos passar uma **segunda função como parâmetro**, para algum processo ser executado;

  array_reduce($arr, $funcao);

# Buscando em arrays

- A função **in_array** verifica se um item passado por parâmetro está no array;
- O retorno é true se encontrar o item e false se não encontrar;
- Vamos passar dois argumentos para a função, exemplo: in_array(“item”, $arr)

# Ordenação de arrays

- Para ordenar em ordem crescente podemos utilizar a função **sort** em um array;
- Para ordenar de forma inversa utilizamos **rsort**;

# Ordenação de arrays associativos

- Para ordenar em ordem crescente pelo valor das chaves, podemos utilizar a função **arsort**;
- Se quisermos ordenar o array pelas chaves, utilizamos o valor **ksort**;

# Exercício 51

- Crie um array associativo com chaves com valor de nomes, e valores com uma pontuação;
- Ordene os dados do maior para o menor;
- Exiba uma lista, simulando um ranking, em HTML;

# Invertendo arrays

- Com a função **array_reverse** podemos obter o array ao inverso;
- Passamos apenas o array como argumento;
- O retorno será um array invertido do original;

# Ordem aleatório de itens

- Com a função **shuffle** podemos reorganizar os itens em ordem aleatória;
- Passamos apenas o array como parâmetro;
- Temos um array retornado em ordem aleatória;

# Somando itens de um array

- Para somar os itens de um array utilizamos a função **array_sum**;
- Ela nos retorna a soma de todos elementos numéricos do array que passamos como argumento;

# Unindo arrays

- Podemos unir arrays, a ação também é conhecida como merge;
- A função que vamos utilizar é a **array_merge**;
- Que como argumento aceita um número indeterminado de arrays;

# Diferença entre arrays

- Podemos verificar qual a diferença entre dois ou mais arrays com PHP;
- A função que vamos utilizar para isso é a **array_diff**;
- Esta função aceita um número indeterminado de arrays;


# Arrays

## Conclusão da seção


# Introdução a OOP

## Introdução da seção

# O que são objetos?

- Objetos são entidades que possuem **comportamentos e características**;
- As características são conhecidas como **propriedades** (variáveis);
- Os comportamentos como **métodos** (funções);
- Os objetos interagem entre si e sistemas são escritos orientados a objetos (paradigma de **Orientação a Objetos**);
- No PHP podemos desenvolver neste paradigma;

# O que são classes?

- Classes são os “pais” dos objetos;
- Em PHP sempre que vamos criar ou **instanciar** um objeto vamos precisar de uma classe;
- A classe contém o **molde do objeto**, ou seja, seus métodos e suas propriedades;
- Podemos mudar o valor para cada objeto criado, mas ele parte do que a classe impõe;

# Declarando uma classe

- Para iniciar uma classe vamos precisar da palavra reservada **class**, e também dar um nome para a classe;
- Por convenção, a **inicial do nome é sempre em maiúscula,** ex: User;
- Como é um bloco de código, a classe é envolvida por **{ }**;
- Exemplo:

```
class User {
}
```

# Instanciando objeto

- Para instanciar um objeto vamos utilizar a palavra **new** em conjunto do nome da classe;
- A partir daí uma entidade com as características da classe será criada;
- Normalmente encapsulamos este valor em uma variável;
- Exemplo:

  $matheus = new User;

# Exercício 52

- Crie uma classe Car;
- Instancie três objetos com esta classe;

# Declarando métodos

- Para declarar um método vamos utilizar a **sintaxe de function**, porém **dentro de uma class**;
- O restante é exatamente igual a sintaxe de função;
- Podemos retornar ou imprimir dados, dependendo da nossa regra de negócios;

# Exercício 53

- Crie uma classe Cachorro;
- Crie o método latir e andar;
- Execute o método em novas instâncias da classe;

# Declarando propriedades

- Para declarar propriedades vamos basicamente criar uma **variável dentro de uma class**;
- Porém precisamos definir a sua privacidade, como por exemplo **public**;
- Uma propriedade pública pode ser acessada fora do escopo do objeto;
- Exemplo:

  public $idade = 29;

# Exercício 54

- Crie uma classe Pessoa;
- Crie a propriedade nome e idade;
- E também um método andar;

# Conhecendo o $this

- O **$this** se refere a instância atual do objeto;
- Podendo assim alterar um valor de uma propriedade do objeto com:

  **$this->proriedade = “x”**;
- Tradução literal = este;
- Podemos invocar um método do objeto com this também;

# Exercício 55

- Crie uma classe Carro;
- Crie algumas propriedades e também a propriedade velocidade_maxima;
- Crie o método setVelocidadeMaxima, onde é possível alterar a velocidade máxima do carro;
- e também o getVelocidadeMaxima onde é possível imprimir a velocidade do carro;

# Constantes em classes

- As **constantes são parecidas com variáveis**, salvam valores em memória;
- Porém o seu **valor não pode ser alterado**;
- Exemplo de sintaxe:

  public const CHAVE_API = “ASO793mJJs39”;

# Visibilidade

- Temos três formatos de visibilidade: **public, protected e private**;
- **public:** A propriedade ou método pode ser acessada de qualquer forma;
- **protected:** A propriedade ou método pode ser acessada apenas pela classe de origem ou as que recebem a mesma de herança;
- **private:** a propriedade ou método pode ser acessada apenas pela classe que foi criada;

# Herança

- A herança é o recurso da OOP que dá a possibilidade de uma classe **herdar métodos e propriedades de outra**;
- A palavra reservada é **extends**;
- Exemplo:

```
class Programador extends Pessoa {
}
```

# Exercício 56

- Crie uma classe Humano com algumas propriedades e o método falar;
- Crie uma outra classe Professor que herda de humano, crie também as propriedades e métodos particulares desta classe;
- Exiba os valores das propriedades da classe pai e também utilize os métodos;

# Checando ancestralidade

- Para checar a ancestralidade de uma classe utilizamos o operador **instanceof**;
- Podemos inserir essa operação em um if, pois vai retornar um booleano;
- Exemplo:

  $objeto instanceof Humano

# Interfaces

- As interfaces criam um **modelo de definição de uma classe**;
- Então toda classe que implementar uma interface, deverá implementar também suas propriedades e métodos, obrigatoriamente;
- A palavra reservada é **implements**;
- Exemplo:

  class Humano implements Caracteristicas

# Traits

- As **traits** permitem o reuso do código sem hierarquia de classes, ou seja, sem herança;
- Podemos assim utilizar os métodos da classe que foi feita a trait;
- Utilizamos a palavra reservada **use**;
- Exemplo;

```
class Teste {
        use ClasseTrait
```

# Classes abstratas

- As **classes abstratas** não podem ser instanciadas;
- Podemos ter métodos abstratos, que devem ser implementados obrigatoriamente se uma classe herdar a abstrata;
- A palavra reservada tanto para classes como para métodos é **abstract**;
- Exemplo:

  abstract class ClasseAbstrata { }

# Construtores

- Pelos construtores **podemos inicializar objetos com valores** de propriedades únicos para cada objeto;
- Passamos como argumentos os valores das propriedades;
- Exemplo:

```
function __construct($portas, $motor, $teto_solar) {
}
```

# Exercício 57

- Crie uma classe Cachorro com propriedades;
- Inicie as propriedades via constructor;
- Crie um método para exibir cada um das propriedades que você criou

# Classes anônimas

- As **classes anônimas** são criadas em uma variável e não possuem nome;
- Elas **funcionam como qualquer outra classe**;
- Precisamos fechar ela com “;”
- Exemplo:

```
$anonima = new class() { };
```

# Verificando classes

- Em PHP temos alguns métodos que nos ajudam a entender as classes;
- **class_exists()** => verifica se uma classe existe;
- **get_class_methods()** => verifica os métodos de uma classe;
- **get_class_vars()** => mapeamento das propriedades de uma classe;

# Verificando objetos

- Em PHP temos alguns métodos que nos ajudam a entender melhor os objetos;
- **is_object()** => verifica se uma variável é um objeto;
- **get_class()** => verifica a classe de um objeto;
- **method_exists()** => verifica se um método existe em um objeto;


# Introdução a OOP

## Conclusão da seção


# Trabalhando com datas

Introdução da seção

# Função date

- A função date recebe um parâmetro, que é o **formato da data**, e este é o primeiro parâmetro da mesma;
- A resposta será a **data atual**;
- Exemplo:

  date(“d/m/y”); // day / month / year

# Função strtotime

- **Recebe uma string** como parâmetro, que é um texto sinalizando tempo;
- A função tenta interpretar e transformar em data;
- Veja um exemplo de utilização:

  echo date(‘d/m/y’, strtotime(‘+2 years’)); // 2 anos a mais

# Função mktime

- A função mktime recebe em seus parâmetros: **hora, minuto, segundo, mês, dia e ano**;
- Assim podemos criar uma data a partir desta informação;
- Exemplo:

```
$date = mktime(01,18,00,03,12,2000);
echo date(‘d/m/y’, $date);
```

# Objeto DateTime

- O objeto **DateTime** permite tratar a data como um objeto;
- Podemos passar um parâmetro que será a data criada, se não passarmos nada a data será a atual;
- Podemos exibir as informações do objeto com **print_r**;
- Exemplo:

```
$dataAtual = new DateTime();
```

# Métodos format e modify

- Os métodos **format** e **modify** são da classe DateTime e nos ajudam a manipular os dados nestes objetos;
- **format** => Formata a data;
- **modify** => Altera a data;

# Métodos setDate e setTime

- Temos mais dois métodos interessantes em DateTime: **setDate** e **setTime**;
- setDate => Recebe ano, mês e dia, alterando completamente a data;
- setTime => Recebe hora, minuto e segundo, alterando o tempo da data;

# Diferenças entre datas

- Podemos calcular a **diferença entre duas datas** com o método **diff**;
- O resultado pode ser formatado com **format**;
- Exemplo:

```
$diferenca = $dateA->diff($dateB);
```

# Comparação de datas

- Datas que foram criadas com o objeto de DateTime podem ser comparadas utilizando os **operadores de comparação**;
- Operadores como: **>, < ou ==**
- Exemplo:

  $dataA > $dataB

# Alterando o fuso horário

- O PHP por padrão vai utilizar **o fuso horário da máquina que está sendo executado**, ou seja, do servidor;
- Porém podemos alterar manualmente o fuso com a função **date_default_timezone_set**;
- Esta função recebe como parâmetro o novo fuso horário em string;


# Trabalhando com datas

## Conclusão da seção


# PHP e a web

## Introdução da seção

# Introdução ao HTTP

- A web roda em cima do protocolo **HTTP** (HyperText Transfer Protocol);
- Quando um navegador solicita uma página web é feito um **request HTTP**;
- Esta requisição recebe uma resposta, ambos podem possuir um **body**;
- A resposta contém um **header** (cabeçalho), que é constituído pelo método (GET, POST), arquivo/path solicitado (index.php) e versão do protocolo HTTP (HTTP/1.x);
- Basicamente uma requisição é enviada e uma resposta é recebida;

# Status HTTP

- Após enviarmos a requisição, vamos receber uma resposta que contém um **status**, que são separados em algumas categorias:
- **100 - 199** => Respostas de informação;
- **200 - 299** => Respostas de sucesso;
- **300 - 399** => Redirecionamento;
- **400 - 499** => Erros do cliente (navegador, ex: 404);
- **500 - 599** => Erros de servidor (ex: 500);

# Métodos HTTP

- As requisições que enviamos também contém métodos, alguns deles são:
- **GET** => Solicita a apresentação de um recurso (ex: visualização de uma página);
- **POST** => Envio de dados ao servidor (ex: cadastro de usuário);
- **PUT** => Atualização de dados;
- **DELETE** => Remoção de dados;
- **PATCH** => Atualização de dado específico;

# Variáveis globais do PHP

- Para lidar com estas requisições o PHP nos dá algumas variáveis globais:
- **$_ENV** => variáveis de ambiente;
- **$_GET** = Parâmetros que foram enviados por request GET;
- **$_POST** => Parâmetros que foram enviados por POST;
- **$_COOKIE** => Valores de cookies;
- **$_SERVER** => Informações sobre o servidor;
- **$_FILES** => Informações sobre os arquivos que vieram por upload;

# Explorando $_SERVER

- Como dito anteriormente $_SERVER tem diversas informações importantes;
- **SERVER_SOFTWARE** => Identificação do servidor;
- **SERVER_NAME** => Hostname, DNS ou IP do servidor;
- **SERVER_PROTOCOL** => Protocolo do servidor;
- **SERVER_PORT** => Porta do servidor;
- **QUERY_STRING** => Argumentos após o ? na URL;

# Processamento de formulários teoria

- Vamos realizar processamentos de formulário de duas maneiras: via **GET** e via **POST**;
- Com o GET vamos processar os parâmetros que vem na query string, ou seja, na URL;
- Com o POST vamos processar as informações que vem na requisição, estas não aparecem na URL;
- Exemplos de uso: Buscas => GET, Registro de usuário => POST

# Teste de formulários com GET

- Vamos precisar criar um formulário e definir o **método como GET**, e também o arquivo ou rota que vamos acessar em **action**;
- No lado do servidor vamos acessar a variável **$_GET** que contém os parâmetros enviados para o servidor;
- Faremos o processamento e retornamos algo para o usuário;

# Teste de formulários com POST

- Vamos precisar criar um formulário e definir o **método como POST**, e também o arquivo ou rota que vamos acessar em **action**;
- No lado do servidor vamos acessar a variável **$_POST** que contém os parâmetros enviados para o servidor;
- Faremos o processamento e retornamos algo para o usuário;

# Autoprocessamento de páginas

- Podemos criar uma página que faz o **processamento dos dados e também exibe o input de informações**;
- Para isso devemos criar um if que checa se o método de requisição (**$_SERVER['REQUEST_METHOD']**) é **GET** ou **POST**;
- Depois criar as duas variações, para cada uma das possibilidades;
- Ou checar se algum parâmetro veio pela requisição e então criar as variações;

# Preenchimento de formulário

- Podemos preencher o formulário com dados que vieram da requisição;
- Isso acontece bastante em resultados de busca ou edições de registros;
- Podemos fazer uma **checagem de se o dado foi enviado e recebido** para a página, e utilizar o echo para exibir no **atributo value** do input;
- Exemplo:

  <input type=”text” value=”<?php echo $nome; ?>”>

# Parâmetro com mais de um valor

- Os inputs de checkbox **podem conter mais de um valor**;
- Para receber todos os eles no backend, precisamos adicionar uma **sintaxe de array** no name;
- Assim receberemos todos os inputs marcados;
- Exemplo:

name=”caracteristicas[]”

# Upload de arquivos

- Para enviar arquivos ao servidor vamos precisar mudar o enctype do formulário para: **multipart/form-data**;
- Também será necessário um input de **tipo file**;
- O tamanho do arquivo pode exaurir a memória do servidor;
- Depois do envio, todos os dados da imagem estarão em **$_FILES**;

# Validação de formulários

- A validação é uma parte importante do **recebimento de dados**;
- Devemos checar se os dados enviados **condizem com o que estamos esperando**;
- Para isso podemos criar condicionais fazendo as **verificações**;
- Caso alguma não atenda ou um campo obrigatório esteja vazio, retornamos uma mensagem ao usuário;
- **Obs:** é possível fazer validações com HTML e também JavaScript;

# Mantendo o estado

- O **HTTP** é um protocolo que não mantém o estado (**stateless**);
- Ou seja, após o fim da requisição **a conexão entre usuário e servidor é finalizada**, a próxima conexão não possui mais relação entre ambos;
- Para conseguir manter estes dados podemos utilizar os **cookies**;
- O problema desta abordagem é que alguns navegadores não permitem o uso de cookies ou o bloqueiam;

# Cookies

- Os cookies são strings que contém informações;
- A função para adicionar um cookie é **setcookie**;
- A função deve ser chamada antes do corpo da página, pois envia dados como **header** (cabeçalho);
- O cookie leva dados como: nome, valor e data de expiração;
- Podemos acessar os cookies de volta com **$_COOKIE**;

# Sobre as sessions

- Com **session** podemos criar uma variável que persiste em diferentes páginas e também perdura por várias visitas ao mesmo site;
- A session **utiliza recursos de cookies** para seu funcionamento, e se o recurso estiver desabilitado propaga a sessão via URL;
- Sessions são utilizadas para: autenticação, carrinho de compras e tudo o que precisa persistir de página em página;
- Acaba sendo o recurso mais utilizado, comparando o com o cookie;

# Iniciando com sessions

- Para iniciar a utilizar session precisamos utilizar a função **session_start**;
- Esta função vai carregar os dados existentes e também permitir salvar novos dados;
- Todas as variáveis salvas em session ficam em **$_SESSION**;
- Podemos salvar utilizando uma **chave e valor**, como em arrays;
- Para deletar todos os dados da session utilizamos **session_destroy**;

# Onde são salvas as sessions?

- As **sessions são salvas em arquivos** no computador que estamos ou no servidor da aplicação;
- O caminho para onde os arquivos são salvos fica em php.ini na configuração **session.save_path**;
- As sessions podem ser salvas em dois formatos: o próprio do PHP e também o Web Distributed Data eXchange (WDDX)

# Introdução ao SSL

- SSL vem de **Secure Sockets Layer**;
- O PHP não se importa muito e não tem vantagens sobre o SSL;
- Porém garantimos que as requisições de dados entre nosso site e servidor estejam mais seguras com a **encriptação dos dados**;
- Devemos apenas ter cuidado com os formulários, para que sejam **enviados para a URL com HTTPS**, pois alguns servidores bloqueiam a conexão HTTP;


# PHP e a web

## Conclusão da seção


# PHP e MySQL

## Introdução da seção

# PHP e Banco de Dados

- O PHP tem suporte para **mais de 20 bancos de dados**;
- O mais comum a ser utilizado é o **MySQL**;
- Há algumas formas de conexão a bancos disponíveis no PHP, a mais famosa é o **PDO** (PHP Data Objects), porém também temos a **mysqli**;
- PDO costuma ser implementado por causa da **abordagem orientada a objetos** e outras vantagens sobre a mysqli;

# Banco de dados Relacionais

- O banco de dado relacional tem sua principal característica **trabalhar com tabelas**;
- Onde ela **possui colunas** que categorizam os dados, que são inseridos nas tabelas;
- O PHP é muito utilizado com DBs relacionais, como o **MySQL**;
- A linguagem para operações com estes bancos é a **SQL**;
- As instruções em SQL costumam ser escritas em **letras maiúsculas**;

# Criando banco de dados

- Podemos criar bancos de dados manualmente em softwares como o **phpMyAdmin** ou por SQL;
- Neste curso, com o intuito de aprender SQL, vamos utilizar os **comandos de query** que vão servir ao longo da sua carreira de programação;
- Para criar bancos vamos utilizar o:

  CREATE DATABASE nomedobanco;

# Removendo bancos

- Podemos também **remover os bancos**, ou seja deletá-los do sistema;
- Isso fará com que **todos os dados e tabelas sejam perdidos**, então tome cuidado;
- O comando para deletar bancos é:

  DROP DATABASE nomedobanco;

# Principais tipos de dados

- Os tipos de dados do banco funcionam como os tipos de dados de variáveis, porém em **vários ‘níveis’ para a melhor performance**;
- **VARCHAR**: texto de 0 a 65535 caracteres;
- **TEXT**: texto com no máximo 65535 bytes;
- **INT**: números inteiros;
- **BIGINT**: números inteiros com maior proporção que o INT;
- **DATE**: data no formato YYYY-MM-DD

# Criando tabelas

- As **tabelas ficam dentro dos bancos**, e os **dados ficam dentro das tabelas**, ou seja, é uma parte fundamental do banco relacional;
- Podemos criar tabelas facilmente por SQL,o comando é:

```
CREATE TABELA nome (
        coluna tipodedado,
        coluna2 tipodedado
);
```

# Removendo tabelas

- Há também a possibilidade de **remover tabelas do banco**;
- Os **dados serão removidos** para sempre;
- O comando para deletar tabelas é:

  DROP TABLE nome;

# Alterando tabela

- Podemos **alterar uma tabela já criada**, com algumas operações: **adicionar coluna, remover coluna, modificar coluna**;
- Normalmente **o banco não costuma mudar após a sua criação**, a operação mais provável é a adição de colunas;
- Comandos de alterar tabelas começam com:

  ALTER TABLE nome

  ADD/DROP COLUMN/MODIFY COLUMN nome

# Constraints

- Constraints são **características que podem ser adicionadas na hora da criação** de uma tabela;
- Podemos definir: **campos que não podem ser  nulos, campos únicos, chaves primárias e mais**;
- O comando fica após o tipo da coluna:

  coluna tipodedado constraint,

# NOT NULL

- A NOT NULL é uma **constraint**;
- Esta constraint não permite que o dado adicionado a esta coluna esteja vazio;
- Exemplo:

  nome VARCHAR(100) NOT NULL

# UNIQUE

- A UNIQUE é uma **constraint**;
- Garante que todos os valores da coluna que foi adicionado sejam únicos;
- **Obs:** podemos unir várias constraints;
- Exemplo:

  nome VARCHAR(100) UNIQUE

  email VARCHAR(255) NOT NULL UNIQUE;

# PRIMARY KEY

- A PRIMARY KEY é uma **constraint**;
- As chaves primárias devem ter valores únicos e não podem ser nulas, geralmente colocadas na coluna de ID;
- Uma tabela **só pode ter uma PRIMARY KEY**;
- Exemplo:

  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY

# Inserindo dados

- Para inserir dados vamos utilizar a instrução **INSERT**;
- Devemos determinar a **tabela, colunas e também os dados** que serão inseridos;
- Exemplo:

  INSERT INTO tabela (coluna, coluna2) VALUES (valor, valor2);

# Selecionando dados

- Para selecionar dados vamos utilizar a instrução **SELECT**;
- Devemos determinar a **tabela e as colunas** que serão selecionadas;
- Podemos utilizar o * para selecionar todas as colunas;
- Exemplo:

  SELECT * FROM tabela;

# Selecionando dados WHERE

- O WHERE é uma cláusula utilizada para **filtrar registros**;
- Vamos poder **resgatar os dados que batem apenas com as características** que estamos procurando;
- Exemplo:

  SELECT colunas

  FROM tabela

  WHERE condição;

# AND, OR e NOT

- Os operadores **AND, OR e NOT** são semelhantes aos do PHP e podem auxiliar o WHERE a filtrar mais ainda os dados;
- Podemos utilizar os operadores em conjunto;
- Veja um exemplo:

  WHERE condicao AND condicao2

# ORDER BY

- Com o ORDER BY é possível **ordenar o retorno com base em alguma coluna**;
- Podemos ordenar de forma crescente (**ASC**) ou descendente (**DESC**);
- Exemplo:

  ORDER BY email ASC;

# Atualizando dados

- Para atualizar dados em uma tabela vamos utilizar a instrução **UPDATE**;
- Precisamos determinar a **tabela, colunas e valores**;
- **Obs:** Um UPDATE sem WHERE vai atualizar todos os dados da tabela;
- Exemplo:

```
UPDATE tabela
SET coluna1 = valor1
WHERE condicao;
```

# Deletando dados

- Para deletar dados de uma tabela vamos utilizar o **DELETE FROM**;
- **Obs:** DELETE sem WHERE vai deletar todos os dados da tabela;
- Exemplo:

  DELETE FROM tabela

  WHERE condicao;

# Criando usuários

- Podemos criar usuários no banco, depois utilizá-los para conectar e realizar as queries;
- Além de criar precisamos **adicionar os privilégios**;
- Exemplo:

  CREATE USER ‘usuario’@’localhost’ IDENTIFIED BY ‘senha’;

  GRANT ALL PRIVILEGES ON * . * TO ‘usuario’@’localhost’;

  FLUSH PRIVILEGES;

# mysqli x PDO

- O **mysqli** é uma extensão do próprio PHP para conectar ao banco MySQL, e tem uma proximidade do código nativo, sendo **mais rápida do que a PDO**;
- **PDO** é uma API para conexão de bancos de dados, **não limitada ao MySQL**, que abstrair alguns conceitos com código PHP tornando esta abordagem mais lenta;
- Ambas as formas podem ser utilizadas da forma orientada a objetos;

# Conectando com mysqli

- Para conectar ao MySQL com o mysqli é muito simples, precisamos utilizar a função **mysqli_connect**;
- Passar os parâmetros de: **host, usuário, senha e banco de dados**;
- Com a conexão feita podemos utilizar as queries;
- Exemplo:

  $conn = new mysqli(“host”, “user”, “pass”, “db”);

# Checando a conexão

- Para verificar se houve algum erro na conexão podemos utilizar a propriedade **connect_errno**;
- E para verificar o erro podemos utilizar o método **connect_error()**
- Podemos inserir a checagem em um if e mostrar a mensagem de erro com um echo, por exemplo;

# Executando uma query

- Para executar uma query vamos usar o método **query**;
- Ele deve ser utilizado **a partir do objeto que fez a conexão**;
- Vamos receber um determinado retorno como resultado, que podem ser os dados, caso seja um SELECT, por exemplo;
- É importante ao fim de todas as queries fechar a conexão, com o método **close**;
- Conexões abertas gastam recursos do servidor e prejudicam a aplicação;

# Criando e deletando tabelas com mysqli

- Para criar e deletar tabelas vamos utilizar as mesmas queries de SQL puro, porém com o auxílio do método **query**;
- **DROP TABLE** para deletar tabelas;
- **CREATE TABLE** para criar tabelas;
- Lembre de fechar a conexão!

# Inserindo dados com mysqli

- Para inserir dados com o mysqli vamos utilizar a mesma query do SQL puro e novamente o método **query**;
- A instrução para inserir dados é a **INSERT INTO**;
- Devemos passar a **tabela, colunas e valores**;

# Selecionando dados com mysqli

- Para resgatar dados com o mysqli vamos utilizar a mesma query do SQL puro e novamente o método **query**;
- A instrução para inserir dados é a **SELECT**;
- Vamos inserir o método query em uma variável, que é onde receberemos os resultados;
- Com o método **fetch_assoc**, transformamos os resultados em um array;

# Prepared statements teoria

- Prepared statements é quando criamos uma **query com placeholders** em vez dos valores reais;
- Aumentando a segurança e a performance da requisição;
- Neste caso o fluxo muda um pouco, vamos utilizar o método **prepare** para preparar a query;
- O **bind_param** para resgatar os parâmetros, e o **execute** para rodar a query;

# Inserindo dados com prepared

- Para inserir dados com **prepared statements** vamos seguir a ideia da aula anterior;
- prepare => bind_param => execute;
- Como teremos uma variável para guardar estes três passos, também devemos fechar a conexão desta variável;
- Ela é comumente chamada de **statement**; (declaração)
- Lembre-se de **fechar a conexão**;

# Selecionando dados com prepared

- Para selecionar dados com prepared statements devemos resgatar os dados com o método **fetch_all**;
- A sequência será: prepare => bind_param => execute => get_result => fetch_all;
- E depois devemos **fechar a conexão**;

# Resgatando apenas uma linha

- Para os selects que precisamos de apenas um dado retornado, podemos utilizar o **fetch_row**;
- Este método pode ser inserido depois de obter o resultado, ou seja, após o **get_result**;

# Atualizando dados com prepared

- Para atualizar dados vamos **seguir os mesmos passos de INSERT e SELECT**;
- Na hora de inserir o **SET** para atualizar os campos, vamos inserir os prepared statements;
- Sequência: prepare => bind_param => execute;

# Deletando dados com prepared

- Para atualizar dados vamos **seguir os mesmos passos de INSERT e SELECT**;
- Na hora de inserir o **WHERE** para remover os registros, vamos inserir os prepared statements;
- Sequência: prepare => bind_param => execute;
- Lembrando que DELETE sem WHERE, causa a remoção de todos os registros;

# Habilitando a PDO

- Antes de começar a de fato utilizar a PDO, é necessário checar se a lib está habilitada;
- Vamos checar no **php.ini** por duas linhas, e descomentar caso estejam:

  php_pdo

  php_pdo_mysql

# Conexão com PDO

- A conexão com **PDO** é um pouco diferente do **mysqli**, mas vamos informar basicamente os mesmos parâmetros;
- Que são: banco de dados, host, nome do banco, usuário e senha;
- Exemplo:

  $conn = new PDO(“mysql:host=localhost;dbname=teste”, $user, $pass);

# Inserindo dados com PDO

- Em PDO vamos utilizar uma abordagem parecida com o mysqli;
- Utilizaremos o método **prepare** para realizar a query com **prepared statements**;
- Depois **bind_param** para estabelecer os valores dos parâmetros;
- Por fim **execute** fará a execução da query;

```
$stmt = $con->prepare(“INSERT INTO x(a, b) VALUES(?, ?)”);
```

# Atualizando dados com PDO

- Para atualizar a abordagem também é parecida;
- Vamo seguir com a sequência: **prepare => bind_param => execute**
- E então a query persistirá no banco:

  $stmt = $con->prepare(“UPDATE x SET a = ?, b = ? WHERE c = ?”)

# Selecionando dados com PDO

- Para selecionar dados a abordagem também é parecida com mysqli;
- Vamo seguir com a sequência: **prepare => bind_param => execute**
- Porém para o resgate dos dados temos dois métodos:
- **fetch**: recebe apenas a primeira ocorrência;
- **fetchAll**: recebe todos os dados;


# PHP e MySQL

## Conclusão da seção


# Design Patterns e padrões

Introdução da seção

# O que é DAO?

- **DAO** = Data Access Object;
- Padrão de código utilizado para persistência de dados;
- Utilizada apenas em **abordagens orientadas a objetos**;
- Há uma classe DAO que será **responsável pelas interações ao DB**;
- Atua como um intermediário de aplicação e banco de dados;
- Separa a regra de negócio da interação com o banco de dados;
- Possibilita também a troca de bancos ou modelo de conexão facilmente;

# DAO na teoria

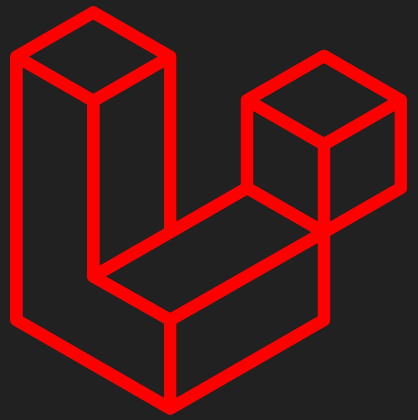
- Trabalharemos com duas classes;
- Exemplificando com uma classe de usuário:
- **UserDAO**: Manipulação de dados do banco;
- **User**: Todas as ações que não envolvem o banco;
- **Create:** User monta um novo usuário com seus campos necessários do banco, UserDAO recebe este objeto e insere o usuário no banco;

# Interface do DAO

- Normalmente também é criada uma **interface** para o DAO;
- Esta interface molda a classe DAO, definindo seus métodos;
- Desta maneira temos um esqueleto para seguir e implementar na classe que vai manipular o banco de dados;
- Os métodos principais da interface são pelo menos **os que constituem o CRUD**;
- A interface pode ser re-implementada em diversos bancos, por exemplo;


# Design Patterns e padrões

## Conclusão da seção

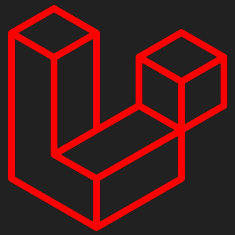


# Laravel

## Introdução da seção

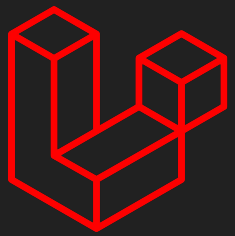
# O que é Laravel?

- Framework construído na linguagem **PHP**;
- Utiliza a arquitetura **MVC** (Model View Controller);
- Possui recursos muito interessantes que auxiliam o desenvolvimento de aplicações: **artisan**, **migrations**, **blade** e etc…
- Fácil de criar código, não é tão burocrático e flexibiliza bastante no desenvolvimento de aplicações;
- A estrutura de pastas é simples, deixando o projeto organizado;



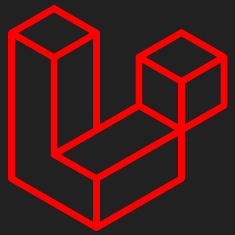
# Instalação das dependências

- Utilização do XAMPP para **MySQL** ou o serviço rodando na máquina;
- Instalação do **Composer**, para instalar o Laravel e suas dependências;
- Instalação do **Laravel**, por meio do Composer;
- Editor de código da sua escolha, indico o **VS Code**;



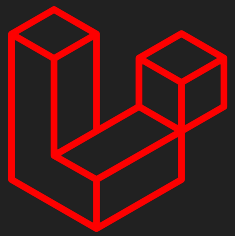
# Rotas e Views

- Vamos acessar as páginas do nosso projeto por meio de **rotas**;
- As rotas chamam as **views**, que são as representações gráficas das páginas;
- Nas views teremos os **templates**, onde há a estruturação da página por meio do HTML;
- Os templates também renderizam **dados dinâmicos** por meio do PHP;



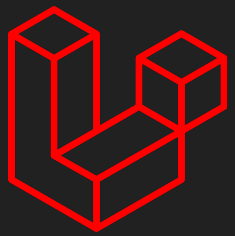
# Conhecendo o Blade

- **Blade** é a template engine do Laravel;
- Com ele, vamos deixar as **nossas views dinâmicas**;
- Inserindo tags de HTML e também **dados que são fornecidos pelo banco**;
- Podemos dizer que as Views serão responsabilidade do Blade;



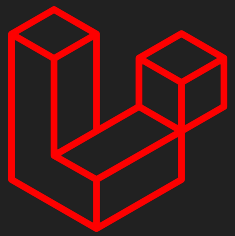
# Layouts com Blade

- A funcionalidade de criar um layout permite o **reaproveitamento de código**;
- Por exemplo: Podemos utilizar o **mesmo header e footer** em todas as páginas sem repetir código;
- Mas o layout do blade não se limita a isso, podemos criar seções do site por meio do layout e também mudar o title da página;



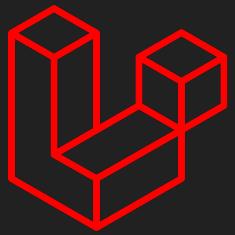
# Adicionando arquivos estáticos

- Uma aplicação web normalmente tem arquivos de **CSS, JS e imagens**;
- O Laravel proporciona uma **maneira muito fácil de inserir estes arquivos** no projeto;
- Todos os recursos ficam na pasta **public**, e tem acesso direto nas tags que trabalham com arquivos estáticos;



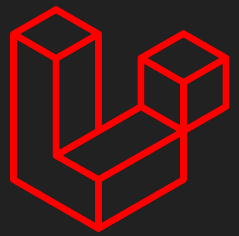
# Controllers

- Os **Controllers** são parte fundamental de toda aplicação em Laravel;
- Geralmente condensam a maior parte da lógica;
- Tem o papel de **enviar e esperar resposta do banco de dados**;
- E também **receber e enviar alguma resposta para as views**;
- Os Controllers podem ser criados via **artisan**;
- É comum retornar uma view ou redirecionar para uma URL pelo Controller;



# Conexão com o banco

- A conexão do Laravel com o banco é configurada pelo arquivo **.env**;
- Isso nos proporciona maior liberdade e também segurança na aplicação;
- O Laravel utiliza um **ORM** (Object-Relational Mapping) chamada **Eloquent**;
- E também para a criação de tabelas as **migrations**;



# Migrations

- As **migrations** funcionam como um versionamento de banco de dados;
- Podemos **avançar e retroceder** a qualquer momento;
- **Adicionar colunas e remover** de forma facilitada;
- Fazer o setup de DB de uma nova instalação em apenas um comando;
- Podemos verificar as migrations com **migrate:status**