**1) Servidor php:**

1. No terminal entrar na pasta do projeto
2. Digitar o comando php -S localhost:8080
3. No navegador acessar o endereço localhost/...

**2) Função is\_numeric($valor)** - verifica se a variavel é um numero ou uma string numerica. Se na string tiver um numero, ele faz a conversao.

**3) Métodos mágicos**:

* sempre inicia com \_\_
* \_\_construct
* \_\_set
* \_\_get
* \_\_toString - só retorna string e qd implementamos na classe não podemos retorna o objeto ou um array.

**4) Namespace:**

* Para o namespace funcionar precisamos implementar o autoload.
* vamos criar o arquivo autoload.php que será responsável por carregar todas as classes do nosso projeto. O ganho é que carregamos apenas as classes necessárias.

function load ($namespace) {

$namespace = str\_replace("\\", DIRECTORY\_SEPARATOR, $namespace);

$caminhoArquivo = \_\_DIR\_\_ . "/" . $namespace . ".php";

return include\_once $caminhoArquivo;

}

// toda vez que instanciamos um objeto o spl... carrega a classe correspondente.

spl\_autoload\_register(\_\_NAMESPACE\_\_ . "\load");

* Dentro das nossas classes, vamos criar o namespace, que vai ser o caminho até chegar o arquivo. Ex: A classe Funcionário está dentro da pasta classes e dentro da pasta funcionário, o namespace será:

namespace\classes\funcionario;

* No index antes de instanciarmos a classe precisamos usar o **use** e em seguida o caminho até do arquivo da classe.

Ex: use classes\funcionarios\Diretor; Agora

* O tipo do objeto passa a ser o namespace, isso evita que tenhamos classes com mesmo nome e isso cause conflito no nosso projeto.

Obs: só precisamos o usar o use.. quando a classe esta em um diretorio diferente de onde eu estou chamando.

* Caso tenhamos uma classe com mesmo nome porém em um diretório diferente, quando fomos usar o use, temos que colocar um apelido nela para não confudir o autoload.

Ex: use classes\Diretor as teste;

* Quando usamos heranca, na classe que está herdando precisamos incluir o use e colocar o namespace da classe que será herdada.

**5) Polimorfismo -** quando subescrevemos um metodo da classe pai.

**6) Classe abstrata-** não pode ser instanciada. Um metodo abstrato deve ser implementado pelas classes filhas não abstratas.

**7) Interface** - os metodos de uma interface devem ser publicos.

**8) Final** - Qd colocamos final antes de uma classe ou metodo, as classes que herdam não podem sobrescrever o metodo ou a classe.

**9) Parâmetros/assinatura de uma função:**

* Podemos passar um valor default para um argumento.

Ex: O salário já vai ter um valor default.

public function \_\_construct($cpf, $salario = self::$pisoSalarial){}

* Para saber numero de argumentos de uma funcao, usamos:

func\_num\_args()

**10) Retorno de uma função:**

* Para retorna um float ou null, podemos declarar:

public function x (): ?float {}

**11) Arrays:**

* Sort($array) ele ordena o array em ordem crescente.
* O foreach não consegue iterar sobre strings, somente sobre arrays ou estender a interface Traversable.
* função explode: pega uma variavel q contem strings e transforma em um array.

ex: $nomes = "a, b, c, d".

explode(",", $nomes);

* função implode: pega um array e tranforma em string.
* função array\_search: recebe como argumentos o $elemento, o $array e opcionalmente true caso queiramos trabalhar com tipos estritos, ou seja, se nessa busca vamos querer comparar o tipo e o valor do elemento. Essa função retorna a posicao do elemento.

OBS: caso passemos por parametro um elem q nao existe no array, o type juggling vai converte o false para 0 e vai retornar o indice do primeiro elemento do array. Entao para contornar isso verificamos se a $posicao existe e é um numero, com a função is\_int.

* unset: apaga um elemento do array. É necessário saber o índice do array.

**8) Método estatico:** não precisamos instancia-lo.

public static metodo(){}

chamada de um metodo estatico: nomeClasse::metodo;

**9) Declare(stric\_types):** O php faz conversoes automaticas dos tipos que ele recebe como argumento. Para travar isso, podemos logo após abrir o php colocar a seguinte declaração:

<?php

declare(strict\_types = 1);

Fazendo isso acabamos com os types juglings ( que convertem dinamicamente os tipos recebidos nos parâmetros da função).