**PHP**

1)lamp stack

linux os

apache server

mysql database

php scripting lang

2)mamp stack

mac os

apache server

mysql database

php scripting lang

3)wamp stack

window os

apache server

mysql database

php scripting lang

software

ampps support all using this

mamp support all

xammp mariadb window

wamp window mysql

|  |
| --- |
| **Php stsart with <?php ?>** |
| **Print msg in php**  **Echo “Hello world”;**  **print ‘hello world’;**  **<?= “hello world”; ?>** |
| **Comments**  **// single line**  **/\* \*/ multiple line**  **# single line** |
|  |
| **Client server model:-**  **it is architecture where server holds all the resources and client will access it whenever it required.**  **Client → internet → server** |
| **Request response pattern**  **client → request→ server**  **client← response← server** |
| **How php works?**  **php is dyanamic interperted lang used by server side.**  **web browser request the php file and web server exectes the php code and generates the htyml file and send it to web browser** |
| **Compare php with**  **asp.net :-- not easy to learn and setting and configure is hard created by microsoft lots of knowledge required**  **ruby on rails :-- ruby is the lang and rails is framework**  **python :--- desktop application bussiness work , can be used to more complex**  **java :--- not simple little complex for web program or same as python uses**  **perl:-- cga programming**  **php:- simplicity efficiency security flexibility familiarity ,neutral lang ,**  **books:-**  **murachs php and mysql**  **learning php mysql and javascript**  **php and mysql for dynamic websites** |
| **Php runs only in .php and not in .html** |
| **<?php echo “ heelo “; ?>**  **<?php <p <?= ‘style=color:#ff0000;” ’ ?>> sammmmojm </p>**  **<?php print<p> hekhdad </p> ?>**  **<p> <?php echo”sjajda” ?> </p>**  **<?= “<p>” ?>**  **sample 5 sfsd**  **<?= “</p>”?>**  **html always ignore white spaces in code or browser**  **php doesnot ignore white spaces** |
| **Escape character in php**  **\n newline <br> \’ single quotes**  **\r carriage <br> \” double quotes**  **\t tab &nbsp; \\ black slash**  **\$ doller** |
| **variable**  **$variable\_name = “value”;**  **echo $variable\_name**  **use “ “ when printing variable\_name** |
| **. dot is use to combine message**  **echo $name. “:” . $message .”<br>”; or**  **echo “$name: $message “;** |
| **functionname are not case sensitive**  **case sensitive for variable** |
| **Global variable**  **global $name;**  **$name = krunal;** |
| **Static variable**  **function counter(){**  **static $count=1; echo $count . ”<br>”;**  **$count = $count +1;**  **}**  **count();** |
| **Superglobal**  **$GLOBALS**  **$\_SERVER $\_COOKIE**  **$\_GET $\_SESSION**  **$\_POST $\_REQUEST**  **$\_FILES $\_ENV** |
| **VARIABLE VARIABLES**  **$male = “krunal”;**  **$gender=”male”;**  **echo $gender; //male**  **echo $$gender ; //krunal;** |
| **Predefined variable**  **go to url php.net** |
| **Isset() variable**  **echo isset($name) ? True : false ;** |
| **Constant**  **does not change the value through the program.**  **define(“name of constant .” , “value of the constant”);**  **define(“LANGUAGE” , “PHP”);**  **$lang= LANGUAGE;**  **echo = “this is $lang”;**  **echo = “this is”. LANGUAGE ;**  **echo constant(‘LANGUAGE’);** |
| **MAGICAL CONSTANT**  **\_\_LINE\_\_ echo \_\_ine\_\_**  **\_\_FILE\_\_** |
| **EXPRESSION AND OPERATOR**  **logical operator**  **== equal to check value int == string**  **=== data type and value identical both should be same int===int**  **!= , <> !== not equal to**  **and &&**  **or ||**  **!**  **= += -= .=**  **& |**  **^ = xor ~ = not**  **backtick ` `**  **echo `dir \*.php`;** |
| **ERROR CONTROL OPERATORS**  **@$value =1/0 ; //this line is error will not show in window it is handled by @**  **or**  **echo $php\_errormsg to check if error is generated or not** |
| **Scalar data types**  **integer**  **$marks = 77**  **$marks =0xHEXAVALUE**  **echo PHP\_INT\_SIZE ;**  **doubles**  **$marks= 44.45;**  **$price=4.55;**  **echo round($price)**  **strings**  **strlen($name)**  **strpos(content , find\_word); // content=”hello world” findword = “hello”**  **stripos(content , find\_word); // it is not case sensitive**  **str\_word\_count // number of words**  **str\_replace(search,replace,subject);**  **strrev() reverse string**  **str\_shuffle() shuffling**  **strtoupper()**  **strtolower()**  **wordwrap($content , 5); to wrap the content**  **trim() ; ltrim() rtrim(); remove whitespaces**    **null**  **unset() deletes the data of variable**  **is\_null()** |
| **Name spaces are case sensitive**  **declare name space in first line**  **only for classes interfaces constant**  **it is keyword**  **NameSpaces.php**  **namespace myconstants;**  **const FILE\_NAME = “NAMESPACES.php” ;**  **DefineNAmespaces.php**  **include “NAMESPACES.php”;**  **echo Myconstants\FILE\_NAME;**  **mynamespaces.php**  **<?php**  **namespace StudentConstants\Student1;**  **const TOTAL\_SUB =7;**  **const TOTAL\_MARKS =700;**  **?>**  **DefineNamespaces.php**  **<?php**  **include “MyNamespaces.php”;**  **echo “total marks:” StudentContsants\Student1\TOTAL\_MARKS; //700**  **echo “total subs:” StudentContsants\Student1\TOTAL\_SUB; //7** |
| **Control statement**  **if() { } else{ }**  **if() { } else if() { } else { }**  **switch() { case 1 : break; default: break; }**  **for(initialization ; condition ; increment){ }**  **while() { }**  **do { } while();**  **break; continue; return;**  **include embed php file in anthor php file**  **require if file is missiing it does not execute further**  **include\_once only include once**  **require\_once**  **goto label:** |
| **Heredoc in string**  **echo <<< DOC\_LABEL write anything here DOC\_LABEL;** |
| **Nowdoc in string**  **echo <<< ‘DOC\_LABEL’ DOC\_LABEL; anything insides prints as it is** |
| **ARRAYS**  **store any data type in array**  **empty array are allowed**  **$arr = [1,2,3];**  **var\_dump($arr); print all the element of array**  **print\_r($arr); print alll the elements of array**  **echo $arr[0] ;**  **count($arr); length of array**  **foreach( $arr as $values){**  **echo $values;**  **}**  **empty($arr)**  **$arr= [ [ ] ]; array in array** |
| **ASSOCIATIVE ARRAY**  **index=> value similar to key => value**  **$arr = [0=> “mon” , 1=> “tue” ];**  **foreach($arr as $value){**  **echo $value.;**  **}**  **or**  **foreach($arr as $key => $value){**  **echo $key. “ => “ . $value.;**  **}**  **print\_r(array\_keys($arr)); only keys**  **foreach(array\_keys($arr) as $key){ print all keys**  **echo $key ;**  **}** |
| **MULTIDIMENSIONAL ARRAY**  **$arr = [ [1,2,3]**  **[4,5]**  **] ;**  **foreach($arr as $singlearr){**  **echo”[“;**  **foreach($singlearr as $value){**  **echo values;**  **}**  **echo”]”;** |
| **ARRAY UNSET method to delete element**  **unset($arr[4] );**  **it deletes the element of array but when new element is added in array it continues with next index of delete element for eg if a[4] is deleted then new element will be adde dat a[5] and not aat a[4];** |
| **Sorting array**  **sort($arr);**  **rsort($arr);**  **for associative array**  **asort($arr);**  **arsort($arr);**  **ksort($arr);**  **krsort($arr);** |
| **Copy array**  **$arr1= $arr;** |
| **SPILT ARRAY**  **explode(“ , “ ,$student); string to array**  **$studentlist = implode(“,” $student); array to string**  **explode(“ , “ ,$student, 3); only 3 index array** |
| **FUNCTION**  **function function\_name (para 1, para2 )**  **{**  **statements;**  **return return-value;**  **}**  **function function\_name (para 1, para2 ) : return\_type**  **{**  **statements;**  **return return-value;**  **}**  **optional parameter at last**  **default parameters at first**  **declare(strict\_types=1); //mention datatypes** |
| **Dynamic fuctions**  **function add($a ,$b) : int{ return $a + $b,}**  **$addFunction = “add”;**  **echo $addFunction(1,2); //3** |
| **AnonymousFunction**  **$addfunction =function($a,$b):int{ $addition =$a+$b; return $addition;} ;**  **echo $addfunction(1,2);**  **$content = “krunal”;**  **$printcontent = function() use($content) { echo $content ;};**  **echo $printcontent;** |
| **Inbuild function**  **strlen (string $string) : int** |
| **Passing reference function**  **$result = 0;**  **function add($a ,$b ,&$result =null){**  **$result =$a +$b;**  **}**  **add(1,2,$result);**  **echo $result;** |
| **Try catch error**  **try{ }**  **catch(Exception $e) { }**  **function divide($x, $y):int { if($y <=0 ) {**  **throw new Exception(“Divided by zeros” );**  **result = $x/$y;**  **return result ; }**  **try{**  **$result = divide(1,0);**  **}catch(Exception $e){**  **echo ‘caught Exception’ . $e→getMessage();**  **}** |
| **Custom exception handling //donot need to write try catch block**  **function custom\_exception\_handler($exception) {**  **echo ‘caught Exception’ . $exception→getMessage() ; }**  **function divide($x, $y):int { if($y <=0 ) {**  **throw new Exception(“Divided by zeros” );**  **result = $x/$y;**  **return result ; }**  **set\_exception\_handler(‘custom\_exception\_handler’);**  **divide(1,0);**  **//here default php exception is changed to custom exception catching** |
| **Finally block**  **try( throw new Exception (“Exception is raised”); }**  **finally {**  **echo “this line is executed before exception is raised”**  **}**  **//safest block** |
| **Raise and catch error**  **function custom\_error\_handler($errno , $errstr){**  **echo $errstr; }**  **set\_error\_handler(‘custom\_error\_handler’);**  **echo(5/0);** |
| **Files**  **1)scandir = list all files in a Dirctory**    **$path =”Testfolder”;**  **$result = scandir($path);**  **var\_dump($result); OR //list all files in array format include . And ..**  **foreach($result as $dir){ // list all files without . And ..**  **if($dir != “.” && $dir != “..” ){**  **echo $dir ;**  **}**  **OR**  **$directory = array\_diff($result, [‘.’ , ‘..’]);**  **var\_dump($directory);**  **foreach($directroy as $dir){**  **echo $dir ; }** |
| **Check for files or directory is\_file or is\_dir**  **$path=”foldername”;**  **$result=scandir($path);**  **$result = array\_diff($result , [‘.’ , “..”]);**  **foreach($result as $dir)**  **{**  **if(is\_file($path . “/ “ .dir )){ //shows only file . Use is\_dir to show directory**  **echo $dir ; }** |
| **Glob method to find file through pattern easy**  **$result = glob(“\*.php”); //display all file with php extension**  **var\_dump($result) ;** |
| **Create a folder**  **if(!file\_exists(“NAmeof\_file” )) {**  **mkdir(“nameof\_file”); //file will be created in given path** |
| **Copy files**  **copy(“source file “ , “destination file add”);** |
| **Find files**  **$fileName = “Myfolder”;**  **if(file\_exists($fileName) ) {**  **echo “file exists” ;**  **if(is\_dir($fileName) ){**  **die(“it is directory”); } } //exits out of program with message**  **else{**  **echo “file /dire does not exist “**  **die(“no such files”);** |
| **Rename file**  **rename(“file1 “ , “renamedfile1” );** |
| **Delete file**  **unlink(“filename”);** |
| **Read and write files**  **open file read file close file**  **r read w write a append**  **$fileName = “file.txt”; //php default file system to read file**  **$content = file\_get\_contents($fileName);**  **echo $content ;**  **OR**  **$fileHandler = fopen(fileName , “r”);**  **$fileSize = filesize($fileName );**  **$ontent = fread($fileHandler , $fileSize)**  **echo $content**  **fclose($fileHandler):** |
| **Write in files**  **$fileHandler = fopen($filename , “w”) or die(“unable to write file”);**  **fwrite($fileHandler , ”this is modified”);**  **fclose($fileHandler);**  **or**  **$fileName = “NewFile.txt”**  **file\_put\_contents($filename , “this is modified content”);**  **best way**  **if (!file\_exits($filename) ) {**  **if( !is\_file($filename) ) {**  **die(“file does not exits “);**  **}**  **}** |
| **Configuration files**  **$setting = parse\_ini\_file(“test.ini”)**  **print\_r($setting);** |
| **Read and write csv files**  **$filename = “student.csv”;**  **$content = file\_get\_contents($filename);**  **print\_r($content);**  **or**  **$csvFile =file($fileName);**  **var\_dump($csvFile);**  **foreach($csvFile as $line){**  **echo $line ;}**  **or**  **$csvFile =file($fileName); // files in array of array format**  **foreach($csvFile as $line){**  **$data[] = str\_getcsv($line)**  **print\_r($data ;}**  **or**  **$csv = array\_map(‘str\_getcsv’ , file($fileName)) ;**  **print\_r($csv);** |
| **Date and time fomat**  **date(forma, timestamp)**  **d = day of month 01-31**  **D = day of week mon -sun**  **m =month in number 01-12**  **M = month in text jan-dec**  **y = year in two digit 08 – 20**  **Y = year in four digit 2008 – 2020**  **date(“d-m-y”)** |
| **h = hours in 12 hours format 01-12**  **H = hour in 24 hours format 00 – 23**  **i= minutes with 00 -59**  **s = seconds with 00-59**  **a = lowercase am pm**  **A= uppercase AM PM**  **echo “current date “ .date(“d-m-y h:i:s:a”)**  **echo “ time in milliseconds “ . time() ;**  **$time = time() + 1000;**  **echo “current date “ .date(“d-m-y h:i:s:a” , $time ) ; // custom time** |
| **$datearr = getdate();**  **foreach($datearr as $format => $val ){**  **echo “$format => $val” ; }**  **echo $datearr[“year”] or [“weekdays”]** |
| **Date function**  **echo date\_create(“now”)→format(“Y-m-d H:i:s”) ;**  **echo date\_create(“+1”)→format(“Y-m-d H:i:s”) ;**  **convert string of time to custom time**  **$strTime = strtotime(“2019-05-21 9:00:00”);**  **echo date (“d:m:y h:i:s “ , $strTime);** |
| **Other country timezone**  **echo date\_default\_timezone\_get()**  **echo date(“m/d/y h:i:s a”);**  **echo date\_default\_timezone\_get(“Australia/Melbourne”);** |
| **Time difference**  **$date = time();**  **sleep(10);**  **$date2 =time();**  **$diff =($date2-date1)/60; echo $diff**  **OR**  **$start= date\_create(“2019-05-26 12:01:00”);**  **$end= date\_create(“2019-05-26 13:01:00”);**  **$diff= date\_diff($end ,$start); print\_r($diff);** |

|  |
| --- |
| **Php oops** |
| **Class name First letter capital and file name should match class name**  **class Car{**  **public variable name ;**  **function function\_name() { }**  **}**  **$car1 = new Car();**  **$car1→name =”ferrai” ;**  **$car1→ sayHello();**  **echo “$car1→name”;** |
| **$this keywords**  **it allowed us to access properties and method of same class**  **$this indicates instance of this class**  **to access the property or method of same class in function we use $this keywords**  **class Car{**  **public $name =” “ ;**    **function getName() :string{ //getter method**  **return $this→name; }**  **function setName($name) : string{ //setter method**  **this→name = $name;**  **}** |
| **Constructor**  **function \_\_construct(){**  **this→name = “krunal” ; //initialise property of class**  **}**  **uses**  **iniyialize databse connection**  **check if file exists.**  **Open file before use**  **check internet connection**  **checkl api is reachable or not** |
|  |
| **Destructor**  **gets loaded automatically**  **function \_\_destruct(){ }**  **use to close database connection**  **perform cleanup activities**  **save log files**  **free up resources**  **save the cache** |
| **Chain methods and property**  **when we have to call two methods at a time**  **class ChainMathods{**  **function method1(){**  **echo “ this is method1”;**  **return $this; // this create the instance of the class for next method**  **}**  **function method2() {**  **echo “this is method 2 “ ; }**  **$cm = new ChainMethods();**  **$cm→ method1()→ method2();**  **//if $this is not use it will give error beacause to access methods of class we require instance of that class which is provided by $this keyword.** |
| **Public and private access modifiers**  **function default are public**  **private member cannot be access outside the class directly but through getter and setter method.** |
| **Inheritance - reusability**  **//multiple inheritance is not allowed**  **class University{ }**  **class Teacher extends University{ }** |
| **Imp concept to understand in short Eg**  **university.php**  **class University{**  **public uniname;**  **public unilocation;**  **function \_\_construct($uniname , Unilocation)**  **{**  **this→uniname =$uniname;**  **this→ unilocation = $unilocation;**  **}**  **function printUni(){**  **echo “this→uniname and this→unilocation”;**  **}}** |
| **Teacher.php**  **class Teacher extends University**  **{**  **public tname;**  **public tage;**    **function \_\_construct($tname ,$tage ,University $university){**  **$this→tname =$tname;**  **$this→tage =$tage;**    **$this→uniname = $uniname→uniname;**  **$this→unilocation = $unilocation→unilocation;**  **}**  **function printTeacher(){**  **echo “$this→tname and $this→tage”;**  **}** |
| **Inheritance.php**  **declare(strict\_types =1);**  **require\_once = “university.php”;**  **require\_once = “teacher.php”;**  **$uni1 = new University(“Boston “ , “london”);**  **teacher1 = new Teacher(“krunal “ , “21” , $uni1);**  **echo printTeacher();**  **echo printUni();** |
| **Protected specifier**  **can be accessed by child** |
| **Over riding**  **same function name in child class**  **if we want to access parent method then**  **parent:: method\_name** |
| **Multi Inheritance**  **class A { }**  **class B extends class A{ }**  **clacc C extends class C{ }** |
| **Final keyword**    **anti inheritance**  **it does not allow inheritance**  **it is used with class or method**  **class as final = prevent Inheritance**  **method as final = prevent overriding** |
| **Interface**  **interface Car{**  **public function break1();**  **public function increaseSpeed();**  **}**  **class SportsCar implement Car{**  **// must have the car function defined here.**  **public function break1(){ }**  **public function increaseSpeed(){ }**  **}** |
| **Abstract class**  **class that we donot want to create object**  **but we can access method of that class**  **abstract class Car{**  **abstract public function applyBreak();**  **public function increaseSpeed() { echo “increase Speed” ;}**  **}**  **class SportCar extends Car{**  **// must have applyBReak() method implemented**  **}** |
| **Static method and property**  **when you want to access class method or properties without an instance then thy ar marked as static**  **scope resolution ::**  **class Car{**  **static public $name=”Ferrai”;**  **static public function applyBreak(){**  **echo “break applied”; }**  **}**  **echo Car::$name;**  **Car::applyBreak();**  **“self” keyword using this we can access static property and method**  **class Calculator{**  **static public $result;**  **static function add($a,$b){**  **self::$result = $a +$b; //**  **} }**  **calculator::add(1,2);**  **echo Calculator::$result ;** |
| **Polymorphism**  **same parent same method different property different class** |
| **Data encapsulation**  **hide the properties and make them access only via methods**  **setter sets the value**  **getter gets the value** |
|  |
| **Magic methods**  **function \_\_construct()**  **\_\_call() invoked when undefiend method is called**  **\_\_toString () invoked when someone tried to print the object**  **\_\_debuginfo() invoked when vardump method is used to print object**  **\_\_set and \_\_get invoked when anyone try to set or get protected or private** |
| **Type hinting**  **function addStudent(string $name) : bool{ }** |
| **Dependency injection**  **relation between parent and child**  **eg**  **class Course{**  **public $cid;**  **function \_\_construct($cid){**  **$this→ cid = $cid; } }**    **class Lesson extends Courses{**  **public lid;**  **function \_\_construct($lid , Course $c){**  **$this→lid =$lid;**  **$this→cid = $c→ cid; }**  **}**  **$cou = new Course(1);**  **$les = new Lesson(10 , $cou);** |
| **Traits and code inclusion**    **// help to solve multiple inheritance**  **trait Database{**  **function getRecord(){**  **echo “this is record” ; }**  **}**  **class Student{**  **use Database;**  **function Connect() {**  **this→ getRecords(); } }**  **//use as many you want trait .. for multiple inheritance.** |
|  |
| **Working with object**  **clone of object**  **$post2 = clone $post1 //copy post 1 to post 2**  **serialised**  **$post1 = new Posts(“this is th first post “);**  **$serialised = serialize($post1);**  **echo $serialised;**  **file\_put\_content(“posts.txt” , $serialised);**  **unserialised**  **$unserialised = file\_get\_contents(“posts.txt” );**  **$post2 = unserialize($unserialised);**  **echo $post2→ post;** |
| **Caching of file** |
| **Comparing objects**  **identity operator // ===**  **Comparison operator // ==** |
| **Autoloading and Dynamic calling**  **spl\_autoload\_register(function($classname){ //similar to include but advance**  **echo “ loading the class :$classname “ ;**  **require\_once(“$classname.php”); }) ;**  **$file1 = new Files();**  **$file→hello();** |
| **Namespace in php**  **Hello.php file is saved in different folder ie Library**  **<?php namespace Library**  **function hello(){**  **echo “hello”; }**  **autoload.php //file name and class name should be same**  **above autoload code same as it is**  **$file1 = new Library\Hello()** |
| **S.O.L.I.D PRINCIPLE**  **S-RP : Single Responsibility Principle**  **one class – one Responsibility**  **O-CP : Open Closed Principle**  **open for extension and close for modification**  **L-sp : LISKOV SUBSTITUTION PRINCIPLE**  **function that use pointer to base classes must be able to use object of derived classes without knowing it**  **I-SP: INTERFACE SEGREGATION PRINCIPLE**  **make fine grained interface that are client specific**  **D-IP:Dependency inversion principle**  **reusable modules should be defined in abstraction.** |
| **Database with oops**  **make database.php which contain all the methods of database that can be used later in any project using oops concept and crud operation.** |
|  |