PHP Tutorial

- server scripting language
- making dynamic and interactive Web pages
- **Sample PHP Program**

-
  ```
  <!DOCTYPE html>
  <html>
  <body>

  <?php
  echo "My first PHP script!";
  ?>

  </body>
  </html>
  ```
- PHP code is executed on the server.
- PHP: Hypertext Preprocessor
- open source scripting language
- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can be used to control user-access
- PHP can encrypt data
- PHP supports a wide range of databases
- runs on various platforms
- compatible with almost all servers used today
- end with a semicolon (`;`).
- NO keywords (e.g. `if`, `else`, `while`, `echo`, etc.), classes, functions, and user-defined functions are case-sensitive.
- all variable names are case-sensitive!

Comments in PHP

- not executed as a part of the program
- `// This is a single-line comment`
- `# This is also a single-line comment`

-
  ```
  /*
  This is a multiple-lines comment block
  that spans over multiple
  lines
  */
  ```

PHP Variables

- no command for declaring a variable
- Rules for PHP variables

- variable starts with the $ sign, followed by the name of the variable
- variable name must start with a letter or the underscore character
- variable name cannot start with a number
- can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- case-sensitive
- **Example:**

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
</head>
<body>
  <?php
          $a=2;
          $b=3;
          print("This Example is for sum of two variable<br>");
          print("Sum of ".$a." and ".$b." is ".($a+$b));
          ?>
</body>
</html>
```

- PHP automatically associates a data type to the variable, depending on its value

Variables Scope

- local
- global
- static
- A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function
- **Example : For Global and Local sc ope**

```
<?php
            $a=2;
            print("This Example is for Global and Local Scope<br>");
            function display()
            {
                    $a=1;
                    print("a value in local scope is $a<br>");
                    //This will generate an error because a and b are in global scope
            }
            display();
            print("a value in Global scope is $a <br >");
            ?>
```

- variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function
- You can have local variables with the same name in different functions
- `global` keyword is used to access a global variable from within a function
- **`Example: Using of a Global Keyword`**

```
➢ <?php
➢               $a=2;
➢               print("This Example is for Global  Keyword<br>");
➢               function display()
➢               {
➢                         global $a;
➢                         print("a value in local scope is $a<br>");
➢                         //This will use the value of variable in global scope
➢               }
➢               display();
➢               print("a value in Global scope is $a <br >");
➢               ?>
```

- PHP also stores all global variables in an array called `$GLOBALS[index]`
- The *`index`* holds the name of the variable
- **`Example: Using of a Global Array`**

```
➢ <?php
➢               $a=2;
➢               $b=3;
➢
➢               print("This Example is for sum of two variable using global array<br>");
➢               function display()
➢               {
➢                         $GLOBALS['sum']=$GLOBALS['a']+$GLOBALS['b'];
➢                         print("Sum of ".$GLOBALS['a']." and ".$GLOBALS['b']." is ".$GLOBALS['sum']);

➢                         //Here we are creating a sum variable using a global array
➢               }
➢               display();
➢ ?>
```

static Keyword

- when a function is completed/executed, all of its variables are deleted
- sometimes we want a local variable NOT to be deleted
- use the `static` keyword when you first declare the variable

```
➢ <?php
➢               function increment()
➢               {
➢                         static $a=0;
➢                         print($a."<br>");
➢                         $a++;
➢               }
➢     increment();
➢     increment();
➢     increment();
➢               ?>
```

- variable is still local to the function

echo and print Statements

- `echo` has no return value while `print` has a return value of 1
- `echo` can take multiple parameters (although such usage is rare) while `print` can take one argument.

```
<?php
        print("Hello World<br>");
        echo("Hello Jeevan");
        ?>
```

Data Types

PHP supports the following data types:

- String
  - sequence of characters
- Integer
  - non-decimal number between -2,147,483,648 and 2,147,483,647.
  - Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation
- Float (floating point numbers - also called double)
  - number with a decimal point or a number in exponential form
- Boolean
  - Boolean represents two possible states: TRUE or FALSE
- Array
  - array stores multiple values in one single variable
- Object
  - object is a data type which stores data and information on how to process that data
  - class is a structure that can contain properties and methods
- NULL
  - can have only one value: NULL.
  - If a variable is created without a value, it is automatically assigned a value of NULL
  -
- Resource

- **Example: This is example for integer,float and string**

```
<?php
                $string="Jeevan";
                $a=5;
                $b=5.5;
                $c=Null;
                $names=array("Jeevan","PawanKalyan","Kanna");
                print(var_dump($a)."<br>");
                print(var_dump($b)."<br>");
```

- print("$string<br>");
- print(var_dump($names)."<br>");
- print(var_dump($c));
- ?>
- **Example:This is example for object**
- <?php
- class Name
- {
- function Name()
- {
- $this->name="Pawan Kalyan";
- }
- }
- $obj=new Name();
- print($obj->name);
- ?>

String Functions

- `strlen()` function returns the length of a string
- <?php
- print(strlen("Hello World")."<br>");
- print(str_word_count("Hello World")."<br>");
- print(strrev("Hello World")."<br>");
- print(strpos("Hello World", "World")."<br>");
- print(str_replace("World", "Jeevan", "Hello World"));
-
- ?>

PHP Numbers

- integer data type is a non-decimal number between -2147483648 and 2147483647
- 4 * 2.5 is 10, the result is stored as float
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)
- **Example: Verification that it is a number or not**
- <?php
- $a=5;
- var_dump(is_int($a));
- ?>

PHP Floats

- commonly store a value up to 1.7976931348623E+308 (platform dependent), and have a maximum precision of 14 digits
- <?php

- $a=2.5;
- print(is_float($a));
- ?>

PHP Infinity

- numeric value that is larger than PHP_FLOAT_MAX is considered infinite
- <?php
- $a=1.9e411;
- var_dump($a);
- ?>

PHP NaN

- NaN stands for Not a Number.

-
  ```
  <?php
  $x = acos(8);
  var_dump($x);
  ?>
  ```

PHP Numerical Strings

- returns true if the variable is a number or a numeric string
- <?php
- $x = 256;
- $y="256";
- $z=$x+$y;
- $a="Jeevan";
- print(var_dump($x));
- print(var_dump($y));
- print(var_dump($z));
- print(var_dump($a));
- print(is_numeric($x));
- print(is_numeric($y));
- print(is_numeric($a));
- print(is_numeric($z));
- ?>

Type Casting

- <?php
- $x = 256;
- $y=(int)"256";
- $z=$x+$y;
- print($z);
- ?>

PHP Constants

- once they are defined they cannot be changed or undefined.
- no $ sign before the constant name
- constants are automatically global
- **Example: creating a constant using define() function case sensitive**
- `define(`*`name, value, case-insensitive`*`)`
- <?php
- define("Name","Jeevan");//This declares a constant which is case sensitive
- function display()
- {
-     print(NAme."<br>");//This generates error
-     print(Name);
- }
- display()
- ?>
- **Example: creating a constant using define() function case insensitive**
- <?php
- define("Name","Jeevan",True);//This declares a constant which is case insensitive
- function display()
- {
-     print(NAme."<br>");//Both generates same output
-     print(Name);
- }
- display()
- ?>
- **Example: creating a constant array using define() function case insensitive**
- <?php
- define("Names",["Jeevan","Pawan","Kalyan"],True);//This declares a constant which is case insensitive
- function display()
- {
-     print_r(NAmes);//Both generates same output
-     print("<br>");
-     print_r(Names);
- }
- display()
- ?>

PHP Operators

Arithmetic Operators

- <?php
- $a=2;
- $b=3;

```
➢ function display()
➢ {
➢       global $a,$b;
➢       print("Sum of two variables ".($a+$b));print("<br>");
➢       print("Difference of two variables is ".($a-$b));print("<br>");
➢       print("Multiplication of two variables is ".($a*$b));print("<br>");
➢       print("Division of two variables is ".($a/$b));print("<br>");
➢       print("Modulus of two variables is ".($a%$b));print("<br>");
➢       print("Exponentiation of two variables is ".($a**$b));print("<br>");
➢ }
➢ display()
➢ ?>
```

Assignment Operators

```
➢ <?php
➢ $a=2;
➢ $b=NULL;
➢ function display()
➢ {
➢       global $a,$b;
➢       $b=$a;
➢       print("The value of b will be equal to a= ".$b);print("<br>");
➢       $a+=1;
➢       print("The value of a will be increment by 1 so a=".$a);print("<br>");
➢       $a-=1;
➢       print("The value of a will be decrement by 1 so a=".$a);print("<br>");
➢       $a*=2;
➢       print("The value of a will be multiplied by 2 then a=".$a);print("<br>");
➢       $a/=2;
➢       print("The value of a will be divided by 2 then a=".$a);print("<br>");
➢       $a%=2;
➢       print("The value of a will be modulous when divided by 2 then a=".$a);print("<br>");
➢ }
➢ display()
➢ ?>
```

Comparison Operators

➢ used to compare two values

```
➢ <?php
➢ $a=2;
➢ $b="2";
➢ $c=1;
➢ function display()
➢ {
➢       global $a,$b,$c;
➢       print("Both a and b are equal ".($a==$b));print("<br>");
➢       print("Both a and b are identical ".($a===$b));print("<br>");//returns 1 when both are of
  same type
➢       print("a is not equal to b ".($a!=$c));print("<br>");
➢       print("a is not equal to b ".($a<>$c));print("<br>");
➢       print("a is not identical to b ".($a!==$b));print("<br>");
➢       print("a is greater than b ".($a>$c));print("<br>");
➢       print("b is greater than a ".($c>$a));print("<br>");//In the sam way <= and >=
➢       print("Relation between a and b is ".($a<=>$b));print("<br>");
➢       print("Relation between a and c is ".($a<=>$c));print("<br>");
➢       //comparision between string and integer only works for equal,identical,not
   identical,spaceship operator
➢  }
➢  display()
➢  ?>
```

Increment / Decrement Operators

```
➢  <?php
➢  $a=2;
➢  function display()
➢  {
➢       global $a;
➢       $a=$a++;
➢       print("post increment ".$a);
➢       $a=++$a;
➢       print("pre increment ".$a);
➢       $a=$a--;
➢       print("post decrement ".$a);
➢       $a=--$a;
➢       print("pre decrement ".$a);
➢  }
➢  display()
➢  ?>
```

Logical Operators

- used to combine conditional statements
- Logical Operators are and,or,xor,&&,||,!

```
➢  <?php
➢  $a=1;
➢  $b=3;
➢  function display()
➢  {
➢       global $a,$b;
➢       if($a==1 and $b==2)
➢       {
➢               print("Your values are matched using AND");
➢               print("<br>");
➢       }
➢  }
➢  display()
➢  ?>
```

String Operators

```
➢ <?php
➢       function display()
➢       {
➢               $First_Name="Jeevan Sai";
➢               $Last_Name="K.";
➢               $Name1=$Last_Name;
➢               $Name1.=$First_Name;
➢               print($Name1);
➢               print("<br>");
➢               print($Last_Name.$First_Name);
➢       }
➢       display();
➢ ?>
```

Array Operators

➢ We have “+,==,===,!=,<>,!==” operators to compare

Conditional Assignment Operators

```
➢ <?php
➢       function display()
➢       {
➢               $a=6;
➢               $str=($a>3)?("The value is greater than 5"):("The value is lesser than 5");
➢               print($str);
➢               print("<br>");
➢               $b=Null??("This is the second statement");
➢               print($b);
➢       }
➢       display();
➢ ?>
```

Conditional Statements

if...elseif...else Statement

```
➢ <?php
➢       $a=-3;
➢       if($a>0)
➢       {
➢               print("The value is greater than 0");
➢       }
➢       else if ($a==0)
```

- {
- print("The value is equal to 0");
- }
- else
- {
- print("The value is less than 0");
- }
- ?>

switch Statement

-
  ```
  <?php
  $favcolor = "red";

  switch ($favcolor) {
      case "red":
          echo "Your favorite color is red!";
          break;
      case "blue":
          echo "Your favorite color is blue!";
          break;
      case "green":
          echo "Your favorite color is green!";
          break;
      default:
          echo "Your favorite color is neither red, blue, nor green!";
  }
  ?>
  ```

PHP Loops

- used to execute the same block of code again and again
- as long as a certain condition is true.
- Loops through a block of code as long as the specified condition is true

PHP while Loop

- executes a block of code as long as the specified condition is true
- <?php
- $a=1;
- while($a<6)
- {
- print($a);
- print("<br>");
- $a++;
- }
- ?>

PHP do...while Loop

- always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true
- `do...while` loop the condition is tested AFTER executing the statements within the loop

```
<?php
    $a=1;
    do
    {
            print($a);
            $a++;
            print("<br>");
    }while($a<6);
?>
```

PHP for Loop

- `for` loop is used when you know in advance how many times the script should run.

```
<?php
    for($a=1;$a<11;$a++)
    {
            print($a);
            print("<br>");
    }
?>
```

PHP foreach Loop

- `foreach` loop works only on arrays, and is used to loop through each key/value pair in an array
- **Example: iterates through only keys**

```
<?php
    $Names=array("Jeevan","Pawan","Kalyan");
    foreach($Names as $value)
    {
            print($value);
            print("<br>");
    }
?>
```

- **Example: iterates through both key and value**

```
<?php
    $Names=array("Jeevan"=>18,"Pawan"=>47,"Kalyan"=>48);
    foreach($Names as $key=>$value)
    {
            print(($key.":".$value));
            print("<br>");
```

- }
- ?>

PHP Built-in Functions

- PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.

PHP User Defined Functions

- function is a block of statements that can be used repeatedly in a program
- not execute automatically when a page loads, will be executed by a call to the function.
- Function names are NOT case-sensitive

PHP Function Arguments

- You can add as many arguments as you want, just separate them with a comma

Loosely Typed Language

- PHP automatically associates a data type to the variable, depending on its value
- PHP 7, type declarations were added
- If we use strict keyword then it will throw a "Fatal Error" if the data type mismatch
- **Without adding strict keyword**

```
<?php
    function add(int $a,int $b)
    {
            return $a+$b;
    }
    print(add(2,"3 types"))//This example is without using strict keyword
?>
```

- **With adding strict keyword**

```
<?php declare(strict_types=1);
    function add(int $a,int $b)
    {
            return $a+$b;
    }
    print(add(2,"3 types"))//This example is without using strict keyword
//This throws an fatal error
?>
```

Sending and Returning values

```
➢ <?php
➢     function display($name="Pawan Kalyan")
➢     {
➢             print("My name is ".$name."<br>");
➢     }
➢     function display1($name="Pawan Kalyan")
➢     {
➢             return $name;
➢     }
➢     display();
➢     print("My name is ".display1("Jeevan"));
➢     print("<br>");
➢     print(display());
➢ ?>
```

Return Type Declarations

➢ **Example: If we want to add two float numbers and and return float answer**

```
➢ <?php
➢     function add($a,$b)
➢     {
➢             return $a+$b;
➢     }
➢     print(add(1.2,3.2));
➢ ?>
```

➢ **Example: If we want to add two float numbers and and return int answer**

```
➢ <?php
➢     function add($a,$b):int
➢     {
➢             return (int)($a+$b);
➢     }
➢     print(add(1.2,3.2));
➢ ?>
```

PHP Arrays

➢ stores multiple values in one single variable

➢ **Example:Simple array**

```
➢ <?php
➢     $names=array("Jeevan","Sai","Kanna");
➢     print($names[0].','.$names[1].','.$names[2]);
➢ ?>
```

➢ can access the values by referring to an index number

➢ There are three types of arrays they are Indexed arrays ,Associative arrays , Multi Dimensional array

➢ **Example: Count number of elements in array and indexing array**

```
<?php
    $names=array("Jeevan","Sai","Kanna");
    print($names[0].','.$names[1].','.$names[2]);print("<br>");
    print("Number of elements in array is ".count($names));// This gives length of array
?>
```

- **Example: Loop Through an array**

```
<?php
    $names=array("Jeevan","Sai","Kanna");
    for($x=0;$x<count($names);$x++)
    {
            print($names[$x]);
            print("<br>");
    }
?>
```

- **Example: associated array have named keys**

```
<?php
    $names=array("Jeevan"=>18,"Sai"=>19,"Kanna"=>20);
    print($names["Jeevan"]);print("<br>");
    print($names["Sai"]);print("<br>");
    print($names["Kanna"]);print("<br>");
?>
```

- **Example: Iterating through associative array**

```
<?php
    $names=array("Jeevan"=>18,"Sai"=>19,"Kanna"=>20);
    foreach ($names as $key => $value) {
            print("(".$key.",".$value.")");
            print("<br>");
    }
?>
```

PHP - Multidimensional Arrays

- array containing one or more arrays
- two-dimensional $cars array contains four arrays, and it has two indices: row and column
- **Example: Iterating over MultiDimensional Arrays**

```
<?php
    $names=array(
            array("Jeevan"=>18,"Sai"=>19,"Kanna"=>20),
            array("Jeevan"=>18,"Sai"=>19,"Kanna"=>20),
            array("Jeevan"=>18,"Sai"=>19,"Kanna"=>20),
    );
    foreach($names as $key=>$value)
    {
            foreach ($value as $a => $b) {
                    # code...
            print("(".$a.",".$b.")");
            }
            print("<br>");
    }
```

- ?>

PHP - Sort Functions For Arrays

sort()

- Sort arrays in ascending order

```
<?php
     $names=array("Jeevan","PawanKalyan","Kanna");
     sort($names);//sorting in acending order
     for($i=0;$i<count($names);$i++)
     {
          print($names[$i]."<br>");
     }
?>
```

rsort()

```
<?php
     $names=array("Jeevan","PawanKalyan","Kanna");
     rsort($names);//sorting in descending order
     for($i=0;$i<count($names);$i++)
     {
          print($names[$i]."<br>");
     }
?>
```

asort()

```
<?php
     $names=array("Jeevan"=>18,"PawanKalyan"=>48,"Kanna"=>19);
     asort($names);//sorting in ascending order considering values of arrays
     foreach ($names as $key => $value) {
          print("(".$key.",".$value.")"."<br>");
     }
?>
```

- Replace asort with arsort to get descending order of array

ksort()

```
<?php
     $names=array("Jeevan"=>18,"PawanKalyan"=>48,"Kanna"=>19);
     ksort($names);//sorting in ascending order considering keys of arrays
     foreach ($names as $key => $value) {
          print("(".$key.",".$value.")"."<br>");
     }
?>
```

- Replace ksort with krsort to get descending order of array

Global Variables - Superglobals

- always accessible, regardless of scope

PHP superglobal variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

PHP $GLOBALS

- PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable
- used to access global variables from anywhere in the PHP script

```
<?php
    $x=2;
    $y=3;
    function add()
    {
            $GLOBALS['z']=$GLOBALS['x']+$GLOBALS['y'];
            print('z='.$z);//This doesnt give output because x is not a local variable
            print("<br>");
            print("z=".$GLOBALS['z']);//This generates a output because you called z as a global
varable
            print("<br>");
    }
    add();
    print('z='.$z);
?>
```

PHP $_SERVER

- holds information about headers, paths, and script locations

Element/Code	Description

$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)

$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page

$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script

$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

PHP $_REQUEST

- used to collect data after submitting an HTML form
- **HTML Code**
- <body>
-     <!--Here in this example we have an input field with a submit button and when the user submits it the form data sends the information to the file mentioned in the action part of the form and process the data there. Then we use super global varable $_REQUEST to collect value of input-->

```
      <form method="post" action="first.php">
              Name:<input type="text" name="fname">
              <input type="submit" name="submit">
      </form>
</body>
```

- **PHP CODE**

```
<?php
      if($_SERVER["REQUEST_METHOD"]=="POST")
      {
              $name=$_REQUEST['fname'];
              if(empty($name)==0)
              {
                      print("Name is empty");
              }
              else
              {
                      print("$name");
              }
      }

?>
```

PHP $_POST

- used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables
- we can use the super global variable $_POST to collect the value of the input field
- **HTML CODE**

- <body>
- 	<!--Here in this example we have an input field with a submit button and when the user submits it the form data sends the information to the file mentioned in the action part of the form and process the data there. Then we use super global varable $_POST to collect value of input-->
- 	<form method="post" action="first.php">
- 		Name:<input type="text" name="fname">
- 		<input type="submit" name="submit">
- 	</form>
- </body>
- **PHP CODE**
- <body>
- <?php
- 	if($_SERVER['REQUEST_METHOD']=='POST')
- 	{
- 		$name=$_POST["fname"];
- 		if(empty($name))
- 		{
- 			print("Name is empty");
- 		}
- 		else
- 		{
- 			print("$name");
- 		}
- 	}
-
- ?>
- </body>

PHP $_GET

- used to collect form data after submitting an HTML form with method="get"
- $_GET can also collect data sent in the URL
- We can use the same example which is written above just by replacing $_POST with $_GET.
- $_GET can input hyperlinks also
- That example is specified below
- **HTML CODE**
- <body>
- 	<a href="first.php?name=jeevan&password=pawankalyan">Go to PHP</a>
- </body>
- **PHP CODE**
- <?php
- 	print("Name: ".($_GET['name']));print("<br>");
- 	print("Password: ".($_GET['password']));print("<br>");
- ?>
- Simple HTML Form using POST method
- **HTML PAGE**

- <body>
-     <form action="first.php" method="post">
-         Name:<input type="test" name="name"><br>
-         E-mail:<input type="text" name="email"><br>
-         <input type="submit" name="submit">
-     </form>
- </body>
- **PHP PAGE**
- <body>
- Hello <?php print($_POST['name']);?><br>
- Your email ID <?php print($_POST["email"]);?> is verified <br>
-
- </body>
- **USING GET**
- **HTML PAGE**
- <body>
-     <form action="first.php" method="get">
-         Name:<input type="test" name="name"><br>
-         E-mail:<input type="text" name="email"><br>
-         <input type="submit" name="submit">
-     </form>
- </body>
- **PHP PAGE**
- <body>
- Hello <?php print($_GET['name']);?><br>
- Your email ID <?php print($_GET["email"]);?> is verified <br>
-
- </body>

GET vs. POST

- Both GET and POST create an array
- keys are the names of the form controls and values are the input data from the user
- $_GET is an array of variables passed to the current script via the URL parameters.
- $_POST is an array of variables passed to the current script via the HTTP POST method.
- GET also has limits on the amount of information to send. The limitation is about 2000 characters.
- GET possible to bookmark the page
- GET may be used for sending non-sensitive data.
- POST has no limits in sending information
- POST supports advanced functionality such as support for multi-part binary input while uploading files to server

PHP Form Validation

- **What is the htmlspecialchars() function?**

  The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

- **Main note on html security**
  - We are using PHP SELF in our code so it can be used by hackers by
  - PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute
  - Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users
  - Suppose if you use the following form code
  - `<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">`
  - Now if the user enter the following url it will be good
  - Assume that we are having the page in test_form.php
  - http://www.example.com/test_form.php if we use this link it will move to test_form.php but if we consider the folloing code
  - http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
  - The user enters the original address and by adding slash to it he continued with a XSS script so that when user efreshes his page he will get an alert popup.
  - Here hacker injected <script> code in our html page
  - The script code is following
  - `<form method="post" action="test_form.php/"><script>alert('hacked')</script>`

How To Avoid $_SERVER["PHP_SELF"] Exploits?

- `<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">`
- Use htmlspecialchars() function converts special charecters to HTML entities. So it wont be affected to our code and no javascript will be involved

Validate Form Data With PHP

- we will do is to pass all variables through PHP's htmlspecialchars() function
- Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
- Remove backslashes (\) from the user input data (with the PHP stripslashes() function)
- **HTML CODE**

```
<form action="first.php" method="post">
            Name:<input type="text" name="name"><br><br>
            E-Mail:<input type="text" name="email"><br><br>
            Website:<input type="text" name="website"><br><br>
            Comment:<textarea name="comment" rows="5"
cols="40"></textarea><br><br>
            Gender:<input type="radio" name="gender" value="male">Male
            <input type="radio" name="gender" value="female">Female
```

- 			<input type="radio" name="gender" value="other">Other
- 			<input type="submit" name="submit" value="submit">
- 		</form>
- **PHP CODE**
- <?php
- 			$name=$email=$gender=$comment=$website="";
- 			if($_SERVER["REQUEST_METHOD"]=="POST")
- 			{
- 				$name=test_input($_POST["name"]);
- 				$email=test_input($_POST["email"]);
- 				$website=test_input($_POST["website"]);
- 				$comment=test_input($_POST["comment"]);
- 				$gender=test_input($_POST["gender"]);
- 			}
- 			function test_input($data)
- 			{
- 				$data=trim($data);
- 				$data=stripslashes($data);
- 				$data=htmlspecialchars($data);
- 				return $data;
- 			}
- 			printf($name);printf("<br>");
- 			printf($email);printf("<br>");
- 			printf($website);printf("<br>");
- 			printf($comment);printf("<br>");
- 			printf($gender);printf("<br>");
-
- 			?>

PHP – Complete Form

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression
also allows dashes in the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

```
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name" value="<?php echo $name;?>">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email" value="<?php echo $email;?>">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website" value="<?php echo
$website;?>">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="female") echo "checked";?> value="female">Female
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="male") echo "checked";?> value="male">Male
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="other") echo "checked";?> value="other">Other
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

PHP Date() Function

- PHP `date()` function formats a timestamp to a more readable date and time
- **To get current Date**
- <?php
-     print(date("d-m-Y"));

- ?>
- **To get current greenwich Time**
- <?php
- print("The following time is GreenWich time<br>");
- print("Time in 24 Hr format ".date("H:i:s"));
- print("<br>");
- print("Time in 12Hr format ".date("h:i:sa"));
- ?>
- **Create a date with mktime()**
- <?php
-     $d=mktime(11,10,59,1,4,2020);
-     print("Mentioned Date is ".date("Y-m-d h:i:sa",$d));
- ?>
- Epoch date: January 1 1970
- **Date form a string**
- <?php
-     $d=strtotime("11:14am Jan 4 2020");
-     print("Date is ".date("Y-m-d h:i:sa",$d));
- ?>
- **Strtotime()**
- <?php
-     $d=strtotime("next Saturday");
-     print("Date is ".date("Y-m-d h:i:sa",$d));
-     print("<br>");
-     $d=strtotime("tomorrow");
-     print("Date is ".date("Y-m-d h:i:sa",$d));
-     print("<br>");
-     $d=strtotime("yesterday");
-     print("Date is ".date("Y-m-d h:i:sa",$d));
-     print("<br>");
-     $d=strtotime("+2 Days");
-     print("Date is ".date("Y-m-d h:i:sa",$d));
-     print("<br>");
-     $d=strtotime("-2 Days");
-     print("Date is ".date("Y-m-d h:i:sa",$d));
-     print("<br>");
-     $d=strtotime("next Month");
-     print("Date is ".date("Y-m-d h:i:sa",$d));
-     print("<br>");
-     $d=strtotime("last Month");
-     print("Date is ".date("Y-m-d h:i:sa",$d));
-     print("<br>");
-     $d=strtotime("-1 Months");
-     print("Date is ".date("Y-m-d h:i:sa",$d));
-     print("<br>");
-     $d=strtotime("+3 Months");
-     print("Date is ".date("Y-m-d h:i:sa",$d));

```
➢       ?>
```

➢ **Iteration through dates and counting time period**

```
➢ <?php
➢               $thisdate=strtotime("today");
➢               $enddate=strtotime("Feb 21");
➢               $daysleft=($enddate-$thisdate)/(60*60*24);
➢               print("days left to my birthday ".$daysleft);
➢               while($thisdate<$enddate)
➢               {
➢                       print("<br>".date("d-M-Y",$thisdate)."<br>");
➢                       $thisdate=strtotime("+1 day",$thisdate);
➢               }
➢       ?>
```

➢ **Prints consecutive weekdays**

```
➢ <?php
➢               $today=strtotime("today");
➢               $endday=strtotime("+1 week",$today);
➢               while($today<$endday)
➢               {
➢                       print(date("l",$today)."<br>");
➢                       $today=strtotime("+1 day",$today);
➢               }
➢       ?>
```

➢ PHP readfile() Function

➢ This function is how to read a text file using php in html

```
➢ <?php
➢               readfile("scrap.txt");
➢       ?>
```

➢ fread() and fclose()

```
➢ <?php
➢               $a=fopen("scrap.txt",'r') or die("Unable to open the file");
➢               print(fread($a,filesize("scrap.txt")));
➢               fclose($a);
➢       ?>
```

➢ Iterating file line by line

```
➢       <?php
➢               $a=fopen("scrap.txt",'r') or die("Unable to open the file");
➢               for($i=0;$i<3;$i++)
➢               {
➢                       print(fgets($a));
➢               }
➢               fclose($a);
➢       ?>
```

➢ Iterating upto end of the file

```
➢ <?php
➢               $a=fopen("scrap.txt",'r') or die("unable to open the file");
➢               while(!feof($a))
➢               {
➢                       print(fgets($a));
➢               }
➢               fclose($a);
➢       ?>
```
➢ **Reading character by character in a file**
```
➢ <?php
➢               $a=fopen("scrap.txt",'r') or die("unable to open the file");
➢               while(!feof($a))
➢               {
➢                       print(fgetc($a));
➢               }
➢               fclose($a);
➢       ?>
```
➢ **Creating a file and editing it and reading it character wise**
```
➢ <?php
➢               $a=fopen("newfile.txt","w") or die("unable to open the file!");
➢               $txt="hi i am pawan kalyan<br>";
➢               fwrite($a,$txt);
➢               $txt="hi i am jeevan<br>";
➢               fwrite($a,$txt);
➢               fclose($a);
➢               $a=fopen("newfile.txt","r") or die("unable to open the file!");
➢               while(!feof($a))
➢               {
➢                       print(fgetc($a));
➢               }
➢       ?>
```
➢ What is a Cookie?

➢ cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values

➢ **How to set a cookie**
```
➢ <!DOCTYPE html>
➢ <?php
➢ $cookie_name = "user";
➢ $cookie_value = "Jeevan";
➢ setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
➢ ?>
➢ <html>
➢ <body>
➢ <?php
➢ if(!isset($_COOKIE[$cookie_name])) {
➢     echo "Cookie named '" . $cookie_name . "' is not set!";
➢ } else {
➢     echo "Cookie '" . $cookie_name . "' is set!<br>";
➢     echo "Value is: " . $_COOKIE[$cookie_name];
➢ }
➢ ?>
```

- <p><strong>Note:</strong> You might have to reload the page to see the value of the cookie.</p>
- </body>
- </html>
- **How to delete a cookie**
- setcookie() function with an expiration date in the past
- use minus sign when you are setting a cookie
- **How to check cookies are enabled or disabled**
-
```
<?php
if(count($_COOKIE) > 0) {
    echo "Cookies are enabled.";
} else {
    echo "Cookies are disabled.";
}
?>
```

PHP Sessions

- Session variables hold information about one single user, and are available to all pages in one application
- In this example we are using sessions in one file we are declaring a variable and in the second file we are accessing it. It is quite different to cookies cookies acts for self page and for every account it shows cookies irrespective of login session
- But Sessions are like they start when we open the file and close the file or close the tab or login to different account
- **First.php**
- <?php
- session_start();
- ?>
- <!DOCTYPE html>
- <html>
- <head>
-     <title></title>
- </head>
- <body>
- <?php
-     $_SESSION["name"]="Jeevan";
-     print("Session variable is set");
-     ?>
- </body>
- </html>
- **Second.php**
- <?php
-     session_start();
- ?>
- <!DOCTYPE html>
- <html>
- <head>
-     <title></title>

- </head>
- <body>
-     <?php
-             print("Your name is ".$_SESSION['name']."<br>");
-             ?>
- </body>
- </html>
- **If we want to modify a variable in php sessions just over write as the way in particular page(second.php)**
- $_SESSION['name']="Pawan kalyan";
- **Destroy a php session**
- <?php
- session_start();
- ?>
- <!DOCTYPE html>
- <html>
- <head>
-     <title></title>
- </head>
- <body>
- <?php
-     $_SESSION["name"]="Jeevan";
-     print("Session variable is set");
-     session_unset();
-     session_destroy();
-     ?>
- </body>
- </html>

PHP Filters

- PHP filters are used to validate and sanitize external input
- <table border="1">
-     <tr>
-             <td>Filter name</td>
-             <td>Filter Id</td>
-     </tr>
-     <?php
-             foreach (filter_list() as $id => $filter) {
-                     print("<tr><td>".$filter."</td><td>".filter_id($filter)."</td><</tr>");
-             }
-             ?>
- </table>

Why Use Filters?

Many web applications receive external input. External input/data can be:

- User input from a form

- Cookies
- Web services data
- Server variables
- Database query results

filter_var() Function

- `filter_var()` function both validate and sanitize data
- `It takes two parameters one is the variable you want to check and other is type of check to use`

Sanitize a String

- Use **filter_var()** function to remove all HTML Tags in string

```
<?php
    $str="<b>Hi i am jeevan</b>";
    $newstr=filter_var($str,FILTER_SANITIZE_STRING);
    print($newstr);
?>
```

- **Validate a datatype**

```
<?php
    $int=10;
    if(filter_var($int,FILTER_VALIDATE_INT)==0||!filter_var($int,FILTER_VALIDATE_INT)==false)
    {
            print("Integer is Valid");
    }
    else
    {
            print("Integer is not valid");
    }
?>
```

- **Validate a IP address**

```
<?php
    $ip="125.34.53.6";
    if(!filter_var($ip,FILTER_VALIDATE_IP)==false)
    {
            print("$ip is valid IP address");
    }
    else
    {
            print("$ip is invalid IP address");
    }
?>
```

- **Sanitize and Validate an Email address**

```
<?php
    $email="kanaparthi_jeevan@srmap.edu.in";
    $email=filter_var($email,FILTER_SANITIZE_EMAIL);
    if(!filter_var($email,FILTER_VALIDATE_EMAIL)===false)
    {
```

```
                print("$email is valid");
        }
        else
        {
                print("$email is not valid");
        }
?>
```

- **Sanitize and validate a url**

```
<?php
        $url="https://www.facebook.com";
        $url=filter_var($url,FILTER_SANITIZE_URL);
        if(!filter_var($url,FILTER_VALIDATE_URL)==false)
        {
                print("$url is valid");
        }
        else
        {
                print("$url is invalid");
        }
?>
```

- **Check if the number is in given range or not**

```
<?php
        $int=1222;
        $min=1;
        $max=200;
        if(filter_var($int,FILTER_VALIDATE_INT,array("options"=>array("min_range"=>$min,"max_ra
nge"=>$max)))==true)
        {
                print("variable is within the given range");
        }
        else
        {
                print("variable is not within the range");
        }
?>
```

- **Check if the address is a valid IPv6 address**

```
<?php
        $ip="2001:0db8:85a3:08d3:1319:8a2e:0370:7334";
        if(filter_var($ip,FILTER_VALIDATE_IP,FILTER_FLAG_IPV6)==true)
        {
                print("$ip is a valid IPV6 address");
        }
        else
        {
                print("$ip is not a valid IPV6 address");
        }
?>
```

- **Valid url with query string**

```
<?php
        $url = "https://www.w3schools.com";
        if(filter_var($url,FILTER_VALIDATE_URL,FILTER_FLAG_QUERY_REQUIRED)==false)
```

```
    {
            print("$url is not a valid url with query string");
    }
    else
    {
            print("$url is valid url with query string");
    }
?>
```

- **Validate the string and removing all charecters which are greater than ascii value 127**

```
<?php
    $str = "<h1>Hello JeevanÆØÅ!</h1>";
    $newstr=filter_var($str,FILTER_SANITIZE_STRING,FILTER_FLAG_STRIP_HIGH);
    //it removes all the charecters whose ascii values > 127
    print($newstr);
?>
```

What is JSON?

- JSON stands for JavaScript Object Notation
- syntax for storing and exchanging data
- PHP has some built-in functions to handle JSON

json_encode()

- //In this example we are creating $age array and then we are encoding into json object called jsonobj using json_encode function and then we are decoding it using json_decode. Json_decode returns an object as default. When we want to decode it to as array then pass second parameter as true so that json object is decoded into arrays

```
<?php

$age=array("jeevan"=>19,"pawan"=>48,"kalyan"=>49);

print_r($age);//This prints an array

$jsonobj=json_encode($age);// this is an json object i which an array converted into json encode

print("<br>");

print_r(json_decode($jsonobj));

?>
```

json_decode()

```
<?php
    $jsonobj='{"jeevan":19,"pawan":48,"kalyan":49}';//this is json string
```

- print_r(json_decode($jsonobj,true));//here it is decoded agian to array from json object by setting the second parameter as true int the function
- ?>

Accessing the Decoded Values

```
➢ <?php
➢     $jsonobj='{"jeevan":18,"Pawan":47,"Kalyan":48}';
➢     $obj=json_decode($jsonobj);//accessing values from php object
➢     print(($obj->Pawan)+($obj->Kalyan));
➢ ?>

➢ <?php
➢     $jsonobj='{"jeevan":18,"Pawan":47,"Kalyan":48}';
➢     $obj=json_decode($jsonobj,true);//accessing values from php associated array
➢     print(($obj["Pawan"])+($obj["Kalyan"]));
➢ ?>
```

Looping Through the Values

- **Looping through objects**

```
➢ <?php
➢     $jsonobj='{"Jeevan":19,"Pawan":47,"Kalyan":48}';
➢     $obj=json_decode($jsonobj);
➢     foreach ($obj as $key => $value) {
➢             print($key."=>".$value."<br>");
➢     }
➢ ?>
```

- **Looping through array**

```
➢ <?php
➢ $jsonobj='{"Jeevan":19,"Pawan":47,"Kalyan":48}';
➢ $obj=json_decode($jsonobj,true);
➢ foreach ($obj as $key => $value) {
➢     print($key."=>".$value."<br>");
➢ }
➢
➢ ?>
```

PHP What is OOP?

- Object-Oriented programming is faster and easier to execute
- Procedural programming is about writing procedures or functions that perform operations on the data
- object-oriented programming is about creating objects that contain both data and functions
- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug

- OOP makes it possible to create full reusable applications with less code and shorter development time
- Classes and objects are the two main aspects of object-oriented programming
- **class is a template for objects, and an object is an instance of a class**
- When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties
- class is defined by using the `class` keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods goes inside the braces

Class Example

```
<?php
    class Student
    {
            //properties
            public $name;
            //methods
            function set_name($name)
            {
                    $this->name=$name;
            }
            function get_name()
            {
                    return $this->name;
            }
    }
    $stu1= new Student();
    $stu2= new Student();
    $stu1->set_name("Jeevan");
    $stu2->set_name("Pawan Kalyan");
    print($stu1->get_name());
    print("<br>");
    print($stu2->get_name());
?>
```

How can we change value of a variable in class

- ### Method 1:

```
<?php
    class student
    {
            public $name;
            function set_name($name)
            {
                    $this->name=$name;
➢               }
➢               function get_name()
➢               {
➢                       return $this->name;
➢               }
➢       }
➢       $stu1=new student();
➢       $stu1->set_name("Jeevan");// this is adding a name inside the class
➢       print($stu1->get_name());
➢   ?>
```

➢ Method 2:

```
➢   <?php
➢       class student
➢       {
➢               public $name;
➢               function set_name()
➢               {
➢                       $this->name=$name;
➢               }
➢       }
➢       $stu1=new student();
➢       $stu1->name="Jeevan";
➢       print($stu1->name);
➢   ?>
```

PHP - instanceof

```
➢   <?php
➢       class student
➢       {
➢               public $name;
➢               function set_name()
➢               {
➢                       $this->name=$name;
➢               }
➢       }
➢       $stu1=new student();
➢       var_dump($stu1 instanceof student);
➢   ?>
```

The __construct Function

```
➢
➢   <?php
➢       class Student
➢       {
```

```
            public $name;
            function __construct($name="Jeevan")
            {
                    $this->name=$name;
            }
            function get_name()
            {
                    return $this->name;
            }
    }
    $stu1=new Student("Pawan Kalyan");
    $stu2=new Student();// This stores a default value
    print($stu1->get_name());
    print("<br>");
    print($stu2->get_name());
?>
```

- **a __construct() function that is automatically called when you create an object from a class, and a __destruct() function that is automatically called at the end of the script**

```
<?php
    class student
    {
            public $name;
            function __construct($name="Ram")
            {
                    $this->name=$name;
            }
            function __destruct()
            {
                    print("My name is {$this->name}");
                    print("<br>");
            }
    }
    $stu2=new student();
    $stu3=new student("jyothi");
    $stu1=new student("Jeevan");
?>
```

PHP - Access Modifiers

- Properties and methods can have access modifiers which control where they can be accessed
- `public` - the property or method can be accessed from everywhere. This is default
- `protected` - the property or method can be accessed within the class and by classes derived from that class
- `private` - the property or method can ONLY be accessed within the class

- **Example1**
- <?php
-     class student
-     {
-             public $name;
-             public $color;
-             public $weight;
-             function set_name($n)
-             {
-                     $this->name=$n;
-                     return $this->name;
-             }
-             protected function set_roll($n)
-             {
-                     $this->roll=$n;
-             }
-             private function set_weight($n)
-             {
-                     $this->weight=$n;
-             }
-             function __destruct()
-             {
-                     print($this->roll);
-             }
-     }
-     $stu=new student();
-     print($stu->set_name("Jeevan"));
-     print("<br>");
-     //here set name function is public access modifier so we can access it from anywhere
-
-     $stu->roll="AP18110010266";
-     //print($stu->roll);// This doesnt generates an error because we used the property directly rather than using method or a function
-
-     //$stu->set_roll("AP18110010266");//This generates an error because that property in protected attribute
- ?>
- **Example 2:**
- <?php
-     class student
-     {
-             public $name;
-             function set_name($n)
-             {
-                     $this->name=$n;
-                     return $this->name;

- }
- function set_roll($n)
- {
- $this->roll=$n;
- }
- private function set_weight($n)
- {
- $this->weight=$n;
- }
- function __destruct()
- {
- print($this->roll);
- }
- }
- $stu=new student();
- print($stu->set_name("Jeevan"));
- print("<br>");
- //here set name function is public access modifier so we can access it from anywhere
- $stu->set_roll("AP18110010266");
- ?>

PHP - What is Inheritance?

- When a class derives from another class
- child class will inherit all the public and protected properties and methods from the parent class
- **Example 1:**
- <?php
- class Studentname
- {
- public $name;
- public function __construct($name)
- {
- $this->name=$name;
- }
- public function intro()
- {
- print("My name is {$this->name} <br>");
- }
- }
- class StudentId extends Studentname
- {
- public $id;
- public function set_id($id)
- {
- $this->id=$id;

```
➢           }
➢           public function message()
➢           {
➢                   print("Name: $this->name<br>");
➢                   print("Id: $this->id <br>");
➢           }
➢       }
➢       $stu=new StudentId("Jeevan");
➢       $stu->set_id("AP18110010266");
➢       $stu->intro();
➢       $stu->message();
➢   ?>
```

➢ **Example 2:**

```
➢   <?php
➢       class Studentname
➢       {
➢           public $name;
➢           public function __construct($name)
➢           {
➢                   $this->name=$name;
➢           }
➢           protected function intro()
➢           {
➢                   print("My name is {$this->name} <br>");
➢           }
➢       }
➢       class StudentId extends Studentname
➢       {
➢           public $id;
➢           public function set_id($id)
➢           {
➢                   $this->id=$id;
➢           }
➢           public function message()
➢           {
➢                   print("Name: $this->name<br>");
➢                   print("Id: $this->id <br>");
➢                   $this->intro();
➢           }
➢       }
➢       $stu=new StudentId("Jeevan");
➢       $stu->set_id("AP18110010266");
➢       //$stu->intro();
➢       $stu->message();
➢   ?>
```

➢ **Inherited methods can be overridden by redefining the methods (use the same name) in the child class**

- The __construct() and other methods in the child class will override the __construct() and other methods in the parent class
- **Example:**

```
<?php
    class Studentname
    {
            public $name;
            public function __construct($name)
            {
                    $this->name=$name;
            }
            protected function intro()
            {
                    print("My name is {$this->name} <br>");
            }
    }
    class StudentId extends Studentname
    {
            public $id;
            public function __construct($name,$sex="")
            {
                    $this->name=$name;
                    $this->sex=$sex;
            }
            public function set_id($id)
            {
                    $this->id=$id;
            }
            public function message()
            {
                    print("Name: $this->name<br>");
                    print("Id: $this->id <br>");
                    print("Sex: $this->sex <br>");
                    $this->intro();
            }
    }
    $stu=new StudentId("Jeevan","Male");
    $stu->set_id("AP18110010266");
    //$stu->intro();
    $stu->message();
?>
```

- **Example: This prevents inheritance.**

```
<?php
    final class Studentname
    {

    }
```

- class StudentId extends Studentname
- {
-
- }
- ?>
- **Example: This prevents method overloading**
- <?php
- class Studentname
- {
- final function stuname()
- {
-
- }
- }
- class StudentId extends Studentname
- {
- function stuname()
- {
-
- }
- }
- ?>

PHP - Class Constants

- Constants cannot be changed once it is declared
- case-sensitive
- class constant is declared inside a class with the `const` keyword
- it is recommended to name the constants in all uppercase letters
- We can access a constant from outside the class by using the class name followed by the scope resolution operator (`::`) followed by the constant name
- **Example:**
- <?php
- class Student
- {
- const name="Jeevan";
- public function name()
- {
- print(self::name);//we shouldnt use the command directly in the class we have to use function to present a data.
- }
- }
- print(Student::name);//this syntax is for getting constant which is inside the class.
- print("<br>");
- $stu=new Student();
- $stu->name();

- ?>

PHP - What are Abstract Classes and Methods?

- Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks
- if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private
- Rules when a child class method is inherited from an avstract class
    - The child class method must be defined with the same name and it redeclares the parent abstract method
    - The child class method must be defined with the same or a less restricted access modifier
    - The number of required arguments must be the same. However, the child class may has optional arguments in addition
- **Abstract classes example**
    - <?php
    - 	abstract class Student
    - 	{
    - 		public $name;
    - 		public function __construct($name)
    - 		{
    - 			$this->name=$name;
    - 		}
    - 		abstract public function id():string;
    - 	}
    - 	class Stu extends Student
    - 	{
    - 		public function id():string
    - 		{
    - 			return "my name is $this->name";
    - 		}
    - 	}
    - 	$stu=new stu("Jeevan");
    - 	print($stu->id());
    - ?>
- **Abstract class example**
    - <?php
    - 	abstract class Student
    - 	{
    - 		public $name;
    - 		public function __construct($name)
    - 		{
    - 			$this->name=$name;
    - 		}
    - 		abstract protected function id():string;
    - 	}
    - 	class Stu extends Student

```
o       {
o               public function id():string
o               {
o                       return "my name is $this->name";
o               }
o       }
o       $stu=new stu("Jeevan");
o       print($stu->id());
o ?>
```

Multiple Traits

```
➢ <?php
➢     trait message1
➢     {
➢             public function Task1()
➢             {
➢                     print("My name is Jeevan<br>");
➢             }
➢     }
➢     trait message2
➢     {
➢             public function Task2()
➢             {
➢                     print("I am fan of PawanKalyan");
➢             }
➢     }
➢     class welcome
➢     {
➢             use message1,message2;
➢     }
➢     $obj= new welcome();
➢     $obj->Task1();
➢     $obj->Task2();
➢ ?>
```

Static Methods

- can be called directly - without creating an instance of a class
- **Example: use the program and manage without static**

```
➢ <?php
➢     class name
➢     {
➢             public function Name()
➢             {
➢                     print("Hi! My name is PawanKalyan");
```

```
                }
        }
        $abc=new name();
        $abc->Name();
?>
```

- **Example: Using a static method**

```
<?php
        class Name
        {
                public static function myname()
                {
                        print("Hi! My name is PawanKalyan");
                }
        }
        //$abc=new name();   $abc->Name(); this generates error because it is not a
constructor
        print(Name::myname());
?>
```

- **Example:**

```
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}

class SomeOtherClass {
  public function message() {
    greeting::welcome();
  }
}
?>
```

PHP - Static Properties

```
<?php
        class Name
        {
                public static $name="Jeevan";
                protected function Names()
                {
                        print(self::$name);
                }
                public function __destruct()
                {
                        print(Name::Names());
                        print(Name::$name);
                }
```

- }
- print(Name::$name);
- $a=new Name();
- //$a->Names();//this gives a error because it is protected function or method
- ?>

- MySQL is the most popular database system used with PHP.

What is MySQL?

- database system used on the web or on the server
- ideal for both small and large applications
- very fast, reliable, and easy to use
- compiles on a number of platforms
- supported by Oracle
- data in a MySQL database are stored in tables
- table is a collection of related data, and it consists of columns and rows
- Both MySQLi and PDO have their advantages
- PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases
- Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security

Open a Connection to MySQL

```
<?php
    $servername="localhost";
    $username="root";
    $password="";
    try
    {
            $conn=new
PDO("mysql:host=$servername;dbname=test",$username,$password);
            $conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
            print("Connected Successfully");
    }
    catch(PDOException $e)
    {
            print("Connection Failed: ".$e->getMessage());
    }
    $conn=null;
?>
```

- If no database is provided then it throws an exception error.
- If an exception is thrown in try block then it automatically throws an error and stop excecuting try block and go to catch block.

Create a MySQL Database Using PDO

- The following PDO example create a database named "testDB"
- <?php
-     $servername="localhost";
-     $username="root";
-     $password="";
-     try
-     {
-             $conn=new PDO("mysql:host=$servername;dbname=test",$username,$password);
-             $conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
-             $sql="CREATE DATABASE testdb";
-             $conn->exec($sql);//use exec() because no results are returned
-             print("Connected Successfully");
-     }
-     catch(PDOException $e)
-     {
-             print($sql."<br>".$e->getMessage());
-     }
-     $conn=null;
- ?>

Create a MySQL Table Using PDO

- After the data type, you can specify other optional attributes for each column
- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
- AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

Create a Table with its Datatype in PDO

- <?php
-     $servername="localhost";
-     $username="root";
-     $password="";
-     try
-     {

- $conn=new PDO("mysql:host=$servername;dbname=test",$username,$password);
- $conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
- $sql="CREATE TABLE STUDENT(
- ID INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
- FIRSTNAME VARCHAR(30) NOT NULL,
- LASTNAME VARCHAR(30) NOT NULL,
- EMAIL VARCHAR(50),
- REG_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)";
- $conn->exec($sql);//use exec() because no results are returned
- $last_id=$conn->lastInsertId();//This gives the last row index
- print("Connected Successfully");
- }
- catch(PDOException $e)
- {
- print($sql."<br>".$e->getMessage());
- }
- $conn=null;
- ?>

Insert Data Into MySQL Using PDO

- $sql="INSERT INTO STUDENT (firstname,lastname,email) VALUES('JEEVAN','KANAPARTHI','kanaparthi_jeevan@srmap.edu.in')";
- $conn->exec($sql);
- Just replace these two lines in top php code so that you can insert values in the table.

Insert Multiple Records Into MySQL Using PDO

- <?php
- $servername="localhost";
- $username="root";
- $password="";
- try
- {
- $conn=new PDO("mysql:host=$servername;dbname=test",$username,$password);
- $conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
- $conn->beginTransaction();
- $conn->exec("INSERT INTO student (firstname, lastname, email)
- VALUES ('John', 'Doe', 'john@example.com')");
- $conn->exec("INSERT INTO student (firstname, lastname, email)
- VALUES ('Mary', 'Moe', 'mary@example.com')");
- $conn->exec("INSERT INTO student (firstname, lastname, email)
- VALUES ('Julie', 'Dooley', 'julie@example.com')");
- $conn->commit();
- print("Record entering is done");

- }
- catch(PDOException $e)
- {
- $conn->rollback();//rollback the transactionif something is failed
- print("Error: ".$e->getMessage());
- }
- $conn=Null;
- ?>

Prepared Statements and Bound Parameters

- used to execute the same (or similar) SQL statements repeatedly with high efficiency
- prepared statements have three main advantages
    - reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
    - Prepared statements are very useful against SQL injections
    - If the original statement template is not derived from external input, SQL injection cannot occur

Prepared Statements in PDO

```
<?php
    $servername="localhost";
    $username="root";
    $password="";
    $dbname="test";
    try
    {
    $conn=new PDO("mysql:host=$servername;dbname=$dbname",$username,$password);
    $conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);//pdo error mode
to exception
            $stm=$conn->prepare("INSERT INTO STUDENT (firstname,lastname,email)
VALUES(:firstname,:lastname,:email)");
            $stm->bindParam(':firstname',$firstname);
            $stm->bindParam(':lastname',$lastname);
            $stm->bindParam(':email',$email);

            //insert first row
            $firstname="Jeevan Sai";
            $lastname="Kanaparthi";
            $email="www.jeevansai2001@gmail.com";
            $stm->execute();
➢
➢                 //insert second row
➢                 $firstname="Jeevan Sai";
➢                 $lastname="Kanaparthi";
➢                 $email="kanaparthi_jeevan@srmap.edu.in";
➢                 $stm->execute();
➢
➢                 //insert third row
➢                 $firstname="Jeevan Sai";
➢                 $lastname="Kanaparthi";
➢                 $email="jeevansai2001@gmail.com";
➢                 $stm->execute();
➢
➢                 print("Record entered succesfully");
➢         }
➢         catch(PDOException $e)
➢         {
➢                 print("Error: ".$e->getMessage());
➢         }
➢         $conn=null;
➢     ?>
```

Select Data With PDO (+ Prepared Statements)

```
➢  <?php
➢  echo "<table style='border: solid 1px black;'>";
➢   echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";
➢
➢  class TableRows extends RecursiveIteratorIterator {
➢      function __construct($it) {
➢          parent::__construct($it, self::LEAVES_ONLY);
➢      }
➢
➢      function current() {
➢          return "<td style='width: 150px; border: 1px solid black;'>" . parent::current(). "</td>";
➢      }
➢
➢      function beginChildren() {
➢          echo "<tr>";
➢      }
➢
➢      function endChildren() {
➢          echo "</tr>" . "\n";
➢      }
➢  }
```

```
➢
➢ $servername = "localhost";
➢ $username = "root";
➢ $password = "";
➢ $dbname = "test";
➢
➢ try {
➢     $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
➢     $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
➢     $stmt = $conn->prepare("SELECT id, firstname, lastname FROM student");
➢     $stmt->execute();
➢
➢     // set the resulting array to associative
➢     $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
➢
➢     foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
➢         echo $v;
➢     }
➢ }
➢ catch(PDOException $e) {
➢     echo "Error: " . $e->getMessage();
➢ }
➢ $conn = null;
➢ echo "</table>";
➢ ?>
```

PHP MySQL Use The WHERE Clause

```
➢ $stmt=$conn->prepare("SELECT id,firstname,lastname FROM student where lastname='Doe'");
```

PHP MySQL Use The ORDER BY Clause

```
➢ $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests
  ORDER BY lastname");
```

Delete Data From a MySQL Table Using MySQLi and PDO

```
➢ <?php
➢       $servername="localhost";
➢       $username="root";
➢       $password="";
➢       $dbname="test";
➢       try
```

```
    {
        $conn=new
PDO("mysql:host=$servername;dbname=$dbname",$username,$password);
        $conn->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
        $sql="DELETE FROM student where ID=9";
        $conn->exec($sql);
        print("Record deleted successfully");
    }
    catch(PDOEXCEPTION $E)
    {
        PRINT("ERROR: ".$E->getMessage());
    }
    $conn=Null;
?>
```

Update Data In a MySQL Table Using MySQLi and PDO

```
$sql="UPDATE STUDENT SET FIRSTNAME='JEEVAN SAI' WHERE ID=1";
$conn->exec($sql);
```

Limit Data Selections From a MySQL Database

```
$stmt=$conn->prepare("SELECT * FROM student LIMIT 4");
$stmt->execute();//This prints the table upto 4 th id
```

- **OR**

```
$stmt=$conn->prepare("SELECT * FROM student LIMIT 4,8");
$stmt->execute();//This prints the table from 4 to 8 th id
```