PHP

What is PHP?

- "PHP: Hypertext Preprocessor“
- Open-source, server-side scripting language
- Used to generate dynamic web-pages
- PHP scripts reside between reserved PHP tags
  - This allows the programmer to embed PHP scripts within HTML pages

What is PHP (cont'd)

- Interpreted language, scripts are parsed at run-time rather than compiled beforehand
- Executed on the server-side
- Source-code not visible by client
  - 'View Source' in browsers does not display the PHP code
- Various built-in functions allow for fast development
- Compatible with many popular databases

What Can PHP Do?

- PHP can generate dynamic page content
- 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 add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

What does PHP code look like?

- Structurally similar to C/C++
- Supports procedural and object-oriented paradigm (to some degree)
- All PHP statements end with a semi-colon
- Each PHP script must be enclosed in the reserved PHP tag

```
<?php
      …
?>
```

- A PHP script can be placed anywhere in the document.
- A PHP script starts with <?php and ends with ?>:

```
<?php
        // PHP code goes here
?>
```

- The default file extension for PHP files is ".php".
- A PHP file normally contains HTML tags, and some PHP scripting code.

```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Comments in PHP

- Standard C, C++, and shell comment symbols

```
// C++ and Java-style comment

# Shell-style comments

/* C-style comments
     These can span multiple lines */
```

String Handling

- String literals (constants) enclosed in double quotes “ ” or single quotes ‘ ’
- Within “”, variables are replaced by their value: – called *variable interpolation*. “My name is $name, I think”
- Within single quoted strings, interpolation doesn’t occur
- Strings are concatenated (joined end to end) with the dot operator “key”.”board” == “keyboard”
- Standard functions exist: strlen(), substr() etc
- Values of other types can be easily converted to and from strings – numbers implicitly converted to strings in a string context.
- Regular expressions be used for complex pattern matching.

Variables in PHP

- PHP variables must begin with a “$” sign
- Case-sensitive ($Foo != $foo != $fOo)
- Global and locally-scoped variables
  - Global variables can be used anywhere
  - Local variables restricted to a function or class
- Certain variable names reserved by PHP
  - Form variables ($_POST, $_GET)
  - Server variables ($_SERVER)
  - Etc.

Variable usage

```
<?php
$foo = 25;      // Numerical variable
$bar = “Hello”;// String variable

$foo = ($foo * 7);  // Multiplies foo by 7
$bar = ($bar * 7);  // Invalid expression
?>
```

Echo

- The PHP command ‘**echo**’ is used to output the parameters passed to it
  - The typical usage for this is to send data to the client’s web-browser
- Syntax
  - void **echo** (string arg***1*** [, string arg***n***...])
  - In practice, arguments are not passed in parentheses since **echo** is a language construct rather than an actual function

Echo example

```
<?php
$foo = 25;                // Numerical variable
$bar = "Hello";           // String variable

echo $bar;                // Outputs Hello
echo $foo,$bar;           // Outputs 25Hello
echo "5x5=",$foo;         // Outputs 5x5=25
echo "5x5=$foo";          // Outputs 5x5=25
echo '5x5=$foo';          // Outputs 5x5=$foo
?>
```

- **Notice** how echo '5x5=$foo' outputs $foo rather than replacing it with 25
- Strings in single quotes (' ') are not interpreted or evaluated by PHP
- This is true for both variables and character escape-sequences (such as "\n" or "\\")

PHP print() Function

- The print() function outputs one or more strings.
- The print() function is not actually a function, so you are not required to use parentheses with it.
- The print() function is slightly slower than echo().

-
  ```
  <?php
  print "Hello world!";
  ?>
  ```

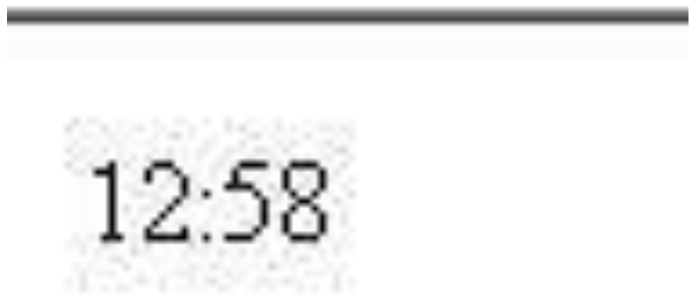
-
  ```
  <?php
  $str = "Hello world!";
  print $str;
  ?>
  ```

PHP details

- Procedural language
  - Compare with Javascript which is event-driven
- C-like syntax - { } ;
- Extensive Function Library
- Good Web-server integration
  - Script embedded in HTML
  - Easy access to form data and output of HTML pages
- Not fully object-oriented
  - Java is fully object oriented – all functions have to be in a class
  - In PHP, classes are additional but quite simple to use

PHP and HTML

- **HTML-embedded**
  - PHP scripts are essentially HTML pages with the occasional section of PHP script.
  - PHP script is enclosed in the tag pair:
    - <?php print date(“H:I”) ?>



```
12:58
```

Variables

Creation of variables

- Variable names always start with $sign. Rest of the variable name can contain alphanumeric, underscore and hyphen.

eg: $name = “abc”;

    $age = 36;

- PHP is case sensitive.

eg: <? Php

```
        $Name = “abc”;
        echo $name;    ?>
```

Escape sequences in PHP

- \n - Insert a new line character.
- \r - Carriage return
- \t - Horizontal tab
- \\ - Backslash
- \$ - Dollar
- \” - Double quote

Changing Data Type

- ‘12,361’ can be used in arithmetic expressions.
- Conversion of string to a number stops when any characters are encountered.
- Type casting process involves dynamically changing the type of data item.
- If we add a double to an integer, the result will be a double even if we are storing it in the original integer variable.
- $int=$int+$float;
- $res=$float+(float)$int;

- is_array(var) – returns TRUE if the variable is an array.
- is_double(var) – returns TRUE if the variable is a double.
- is_float(var) – returns TRUE if the variable is a floating point number.
- is_int(var) - returns TRUE if the variable is an integer.
- is_string(var) - returns TRUE if the variable is a string.
- is_object(var) - returns TRUE if the variable is an object.

Arithmetic Operations

```
<?php
        $a=15;
        $b=30;
        $total=$a+$b;
        Print $total;
        Print "<p><h1>$total</h1>";
        // total is 45
?>
```

- $a - $b   // subtraction
- $a * $b   // multiplication
- $a / $b   // division
- $a += 5 // $a = $a+5 Also works for *= and /=

Concatenation

- Use a period to join strings into one.

```
<?php
$string1=“Hello”;
$string2=“PHP”;
$string3=$string1 . “ ” . $string2;
Print $string3;
?>
```

```
Hello PHP
```

PHP Control Structures

- Control Structures: Are the structures within a language that allow us to control the flow of execution through a program or script.
- Grouped into conditional (branching) structures (e.g. if/else) and repetition structures (e.g. while loops).

PHP Conditional Statements

In PHP we have the following conditional statements:

- **if statement** - executes some code only if a specified condition is true
- **if...else statement** - executes some code if a condition is true and another code if the condition is false
- **if...else if....else statement** - selects one of several blocks of code to be executed
- **switch statement** - selects one of many blocks of code to be executed

The if Statement

- The if statement is used to execute some code **only if a specified condition is true**.

Syntax

```
if (condition)
{
code to be executed if condition is true;
}
```

Example

```
<?php
$t=date("H");
if ($t<"20")
{
echo "Have a good day!";
}
?>
```

- The example will output "Have a good day!" if the current time is less than 20

The if...else Statement

Syntax

```
if (condition)
{
code to be executed if condition is true;
}
else
{
code to be executed if condition is false;
}
```

```
< ?php
$t=date("H");
if ($t<"20")
{
echo "Have a good day!";
}
else
{
echo "Have a good night!";
}
?>
```

- The example will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise.

The if...else if....else Statement

Syntax

```
if (condition)
    {
    code to be executed if condition is true;
    }
    elseif (condition)
    {
    code to be executed if condition is true;
    }
    else
    {
    code to be executed if condition is false;
    }
```

```
< ?php
$t=date("H");
if ($t<"10")
{
echo "Have a good morning!";
}
else if ($t<"20")
{
echo "Have a good day!";
}
else
{
echo "Have a good night!";
}
?>
```

- The example will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!“.

- Example if/else if/else statement:
  - if ($foo == 0) {
  -         echo 'The variable foo is equal to 0';
  - }
  - else if (($foo > 0) && ($foo <= 5)) {
  -         echo 'The variable foo is between 1 and 5';
  - }
  - else {
  -         echo 'The variable foo is equal to '.$foo;
  - }

The PHP switch Statement

```
switch (n)
  {
  case label1:
    code to be executed if n=label1;
    break;
  case label2:
    code to be executed if n=label2;
    break;
  case label3:
    code to be executed if n=label3;
    break;
  ...
  default:
    code to be executed if n is different from all labels;
  }
```

```
<?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, or
    green!";
    }
    ?>
```

While Loops

- While (condition)
  {
      Statements;
  }

```
<?php
$count=0;
While($count<3)
{
        Print "hello PHP. ";
        $count += 1;
        // $count = $count + 1;
        // or
        // $count++;
?>
```

```
hello PHP. hello PHP. hello PHP.
```

The PHP do...while Loop

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do
   {
   code to be executed;
   }
 while (condition is true);
```

-
  ```
  <?php
  $x=1;
  do
    {
    echo "The number is: $x <br>";
    $x++;
    }
  while ($x<=5)
  ?>
  ```

The PHP for Loop

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

Syntax

for (*init counter; test counter; increment counter*)
{
code to be executed;
}

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

```
<?php
  for ($x=0; $x<=10; $x++)
    {
    echo "The number is: $x <br>";
    }
  ?>
```

Data Types in PHP

- PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be categorized further in 3 types:

1. Scalar Types
2. Compound Types
3. Special Types

- Scalar(It holds only single value)

1. Integer
2. Float/double
3. String
4. Boolean

- **Integer Data type**
- Integer means **numeric** data types. A whole number with **no fractional** component. Integer may be **less** than **greater** than or equal to **zero**.
- The size of an integer is platform-dependent, although a maximum value of about two billion is the usual value (that's 32 bits signed).
- **Float/double Data type**
- It is also called **numeric** data types.A number with a **fractional** component.

- **String Data type**
- **Non numeric** data type **String** can hold **letters**, **numbers** and **special characters**.
- **String** value must be enclosed either in **single quotes** or **double quotes**.
- **Boolean Data type**
- **Boolean** are the simplest data type. Like a switch that has only two states ON means true(1) and OFF means false(0).

- **Compound(Multiple values in single variable)**

1. Array
2. Object

- <?php
- class Demo
- {
- public function show()
- {
- echo "This is show method<br/>";
- }
- }
- $obj= new Demo();
- //$obj->show();
- //$obj->show();
- var_dump($obj);
- ?>

- Special Data types

1. Null
2. Resource

- **Null Data type**
- The special Data type "NULL" represents a variable with no value.
- **Resource Data Type**
- The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. A common example of using the resource data type is a database call.
- <?php
- $con = mysqli_connect("localhost","root","","users");
- ?>

Date Display

```
2009/4/1
```

$datedisplay=date(“yyyy/m/d”);
Print $datedisplay;
# If the date is April 1st, 2009
# It would display as 2009/4/1

```
Wednesday, April 1, 2009
```

$datedisplay=date(“l, F m, Y”);
Print $datedisplay;
# If the date is April 1st, 2009
# Wednesday, April 1, 2009

Month, Day & Date Format Symbols

M	Jan
F	January
m	01
n	1

Day of Month	d	01
Day of Month	J	1
Day of Week	l	Monday
Day of Week	D	Mon

PHP Arrays

- PHP array is an ordered map (contains value on the basis of key). It is used to hold multiple values of similar type in a single variable.
- There are 3 types of array in PHP.
  - Indexed Array
  - Associative Array
  - Multidimensional Array

PHP Indexed Array

- PHP index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.
- There are two ways to define indexed array:
- $season=**array**("summer","winter","spring","autumn");
- <?php
- $season=**array**("summer","winter","spring","autumn");
- echo "Season are: $season[0], $season[1], $season[2] a nd $season[3]";
- ?>
-

-
  ```
  $season[0]="summer";
   $season[1]="winter";
   $season[2]="spring";
   $season[3]="autumn";
  ```

-
  ```
  <?php
        $season[0]="summer";
        $season[1]="winter";
        $season[2]="spring";
        $season[3]="autumn";
        echo "Season are: $season[0], $season[1], $
       season[2] and $season[3]";
  ?>
  ```

PHP Associative Array

- We can associate name with each array elements in PHP using => symbol.
- There are two ways to define associative array:
- <?php

```
$salary=array("Sonoo"=>"350000","John"=>"4
50000","Kartik"=>"200000");
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
?>
```

- <?php

```
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
echo "Sonoo salary: ".$salary["Sonoo"]."<br/>"
echo "John salary: ".$salary["John"]."<br/>";
echo "Kartik salary: ".$salary["Kartik"]."<br/>";
?>
```

PHP For Each Loop

- The foreach loop - Loops through a block of code for each element in an array.
- The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.
- Syntax
- foreach ($*array* as $*value*) {
    code to be executed;
  }
- For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

-
  ```
  <?php
  $colors
  = array("red", "green", "blue", "yellow");
  foreach ($colors as $value) {
    echo "$value <br>";
  }
  ?>
  ```
- Output

-
  ```
  red
  green
  blue
  yellow
  ```

-
  ```
  <?php
  $age
  = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"
  43");
  foreach($age as $x => $val) {
    echo "$x = $val<br>";
  }
  ?>
  ```
- Output

-
  ```
  Peter = 35
  Ben = 37
  Joe = 43
  ```

- To loop through and print all the values of an associative array, you could use a foreach loop, like this:

```
<html>
<body>
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
foreach($age as $x=>$x_value)
{
echo "Key=" . $x . ", Value=" . $x_value;
echo "<br>";
}
?>
</body>
</html>
```

- Op : Key=Peter, Value=35

```
Key=Ben, Value=37          Key=Joe, Value=43
```

PHP Multidimensional Array

- PHP multidimensional array is also known as array of arrays. It allows you to store tabular data in an array. PHP multidimensional array can be represented in the form of matrix which is represented by row * column.
- $emp = **array**

```
(
 array(1,"sonoo",400000),
 array(2,"john",500000),
 array(3,"rahul",300000)
 );
```

```
<?php
$emp = array
  (
  array(1,"sonoo",400000),
  array(2,"john",500000),
  array(3,"rahul",300000)
  );
for ($row = 0; $row < 3; $row++) {
  for ($col = 0; $col < 3; $col++) {
    echo $emp[$row][$col]."  ";
  }
  echo "<br/>";
}
?>
```

Output

```
1 sonoo 400000
2 john 500000
3 rahul 300000
```

Array Functions

- **1) PHP array() function**
- PHP array() function creates and returns an array. It allows you to create indexed, associative and multidimensional arrays.
- **2) PHP array_change_key_case() function**
- PHP array_change_key_case() function changes the case of all key of an array.
- It changes case of key only.

- <?php

$salary=**array**("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");

print_r(array_change_key_case($salary,CASE_UPPER));

?>

- Array ([SONOO] => 550000 [VIMAL] => 250000 [RATAN] => 200000)

- <?php

$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");

print_r(array_change_key_case($salary,CASE_LOWER));

?>

- Array ([sonoo] => 550000 [vimal] => 250000 [ratan] => 200000)

- **3) PHP array_chunk() function**
- PHP array_chunk() function splits array into chunks. By using array_chunk() method, you can divide array into many parts.
- <?php

$salary=**array**("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");

print_r(array_chunk($salary,2));

?>

- Array (

[0] => Array ([0] => 550000 [1] => 250000)

[1] => Array ([0] => 200000)

)

- **4) PHP count() function**
- PHP count() function counts all elements in an array.
-
```
<?php
$season=array("summer","winter","spring","autumn");
echo count($season);
?>
```

- **5) PHP sort() function**
- PHP sort() function sorts all the elements in an array.
- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key
-
```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

- **6) PHP array_reverse() function**
- PHP array_reverse() function returns an array containing elements in reversed order.
- <?php
- $season=**array**("summer","winter","spring","autumn");
- $reverseseason=array_reverse($season);
- **foreach**($reverseseason **as** $s)
- {
-   echo "$s<br />";
- }
- ?>
- autumn
- spring
- winter
- summer

- **7) PHP array_search() function**
- PHP array_search() function searches the specified value in an array. It returns key if search is successful.
- <?php
- $season=array("summer","winter","spring","autumn");
- $key=array_search("spring",$season);
- echo $key;
- ?>
- Output:
- 2

- **8) PHP array_intersect() function**
- PHP array_intersect() function returns the intersection of two array. In other words, it returns the matching elements of two array.
- <?php
- $name1=array("sonoo","john","vivek","smith");
- $name2=array("umesh","sonoo","kartik","smith");
- $name3=array_intersect($name1,$name2);
- foreach($name3 as $n)
- {
-   echo "$n<br />";
- }
- ?>
- Output:

```
sonoo
smith
```

- array_pop(array) – removes last element from the array and returns it.
- Array_keys(array) – returns an array containing all of the keys from the associative array which is given as the parameter.
  - <?php
  - $a=array("Volvo"=>"XC90","BMW"=>"X5","Toyota"=>"Highlander");
  - print_r(array_keys($a));
  - ?>
  - Output
  - Array ([0] => Volvo [1] => BMW [2] => Toyota)
- array_push(array, var[, varn]) – adds one or more elements into the end of the array.
- array_shift(array) – returns the first element from the array, remove it from the array and so shortens the array by one.
- array_slice(array, offset[, length]) – returns a subarray at the position indicated by the offset parameter. If no length is given, all elements to the end of the array are returned.

- array_unshift(array, var1[, varn]) – function inserts new elements to an array. The new array values will be inserted in the beginning of the array.
  - <?php
  - $a=array("a"=>"red","b"=>"green");
  - array_unshift($a,"blue");
  - print_r($a);
  - ?>
  - Output: Array ([0] => blue [a] => red [b] => green)
- Each(array) – returns the next key:value pair from an array.
- in_array($var, array) – returns TRUE if the variable, $var is present in the array.
- array_merge(arr1, arr2[, arrn]) – merge all of the arrays. If common keys, the values from later arrays will override those from earlier ones.

- list(var1, var2[, varn]) - used to assign values to a list of variables in one operation.

  -
    ```
    <?php
    $info = array('coffee', 'brown', 'caffeine');
    // Listing all the variables
    list($drink, $color, $power) = $info;
    echo "$drink is $color and $power makes it special.\n";
    // Listing some of them
    list($drink, , $power) = $info;
    echo "$drink has $power.\n";
    // Or let's skip to only the third one
    list( , , $power) = $info;
    echo "I need $power!\n";
    // list() doesn't work with strings
    list($bar) = "abcde";
    var_dump($bar); // NULL
    ?>
    ```
- Sizeof(var) - Return the number of elements in an array.

- The **current()** function returns the value of the current element in an array.
- Every array has an internal pointer to its "current" element, which is initialized to the first element inserted into the array.
- <?php
- $people = array("Peter", "Joe", "Glenn", "Cleveland");
- echo current($people) . "<br>";
- ?>
- Output
- Peter

- next() - moves the internal pointer to, and outputs, the next element in the array
- current() - returns the value of the current element in an array
- end() - moves the internal pointer to, and outputs, the last element in the array

-
  ```
  <?php
  $people = array("Peter", "Joe", "Glenn", "Cleveland");
  echo current($people) . "<br>";
  echo next($people) . "<br>";
  echo prev($people);
  ?>
  ```
- Output

-
  ```
  Peter
  Joe
  Peter
  ```

Form handling

- We can create and use forms in PHP. To get form data, we need to use PHP superglobals $_GET and $_POST.
- Example
- When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.
- To display the submitted data you could simply echo all the variables.

GET vs. POST

- Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
- Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- $_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.

File: form1.html

```
<form action="welcome.php" method="get">
Name: <input type="text" name="name"/>
<input type="submit" value="visit"/>
</form>
```

File: welcome.php

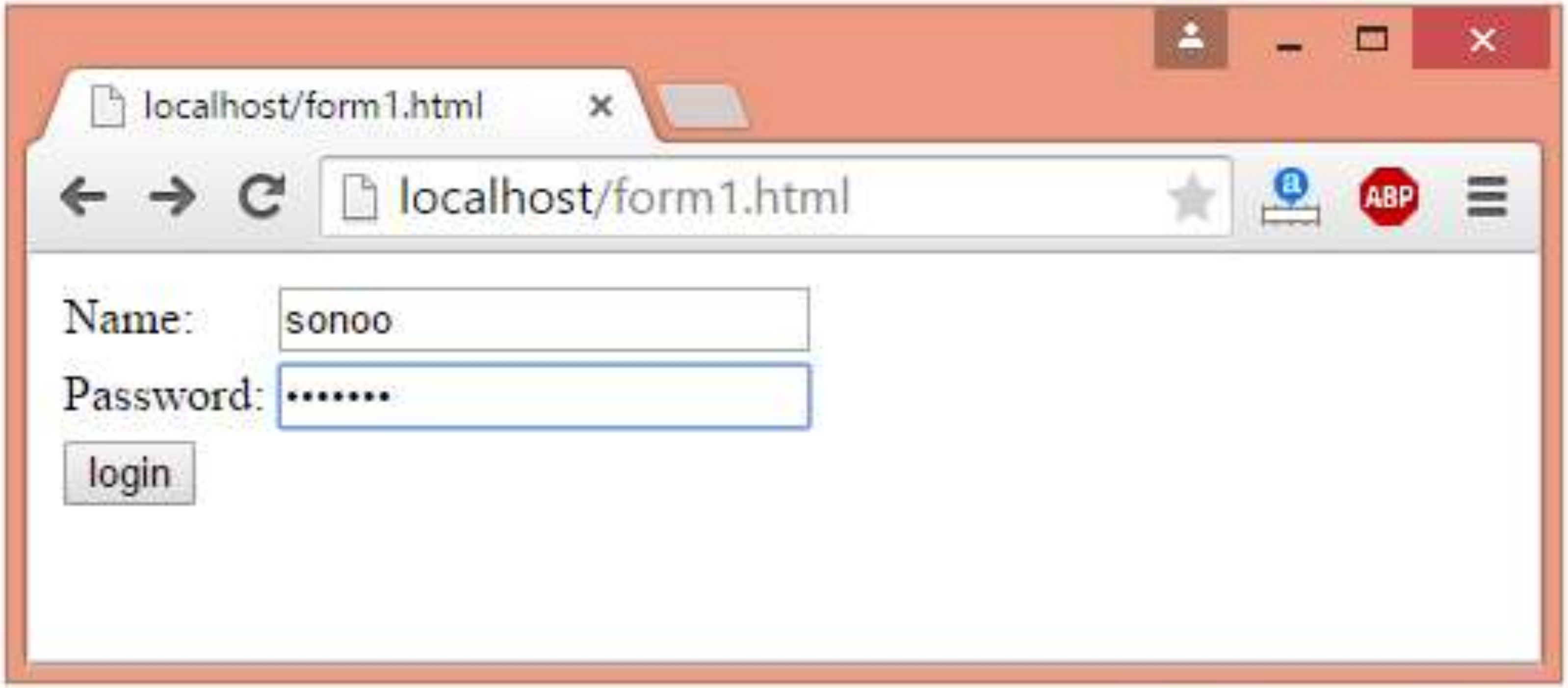
```
<?php
$name=$_GET["name"];//receiving name field value in
 $name variable
echo "Welcome, $name";
?>
```

- *File: form1.html*

```
<form action="login.php" method="post">
<table>
<tr><td>Name:</td><td> <input type="text" name="name"/></td></tr>
<tr><td>Password:</td><td> <input type="password" name="password"/>
</td></tr>
<tr><td colspan="2"><input type="submit" value="login"/>  </td></tr>
</table>
</form>
```

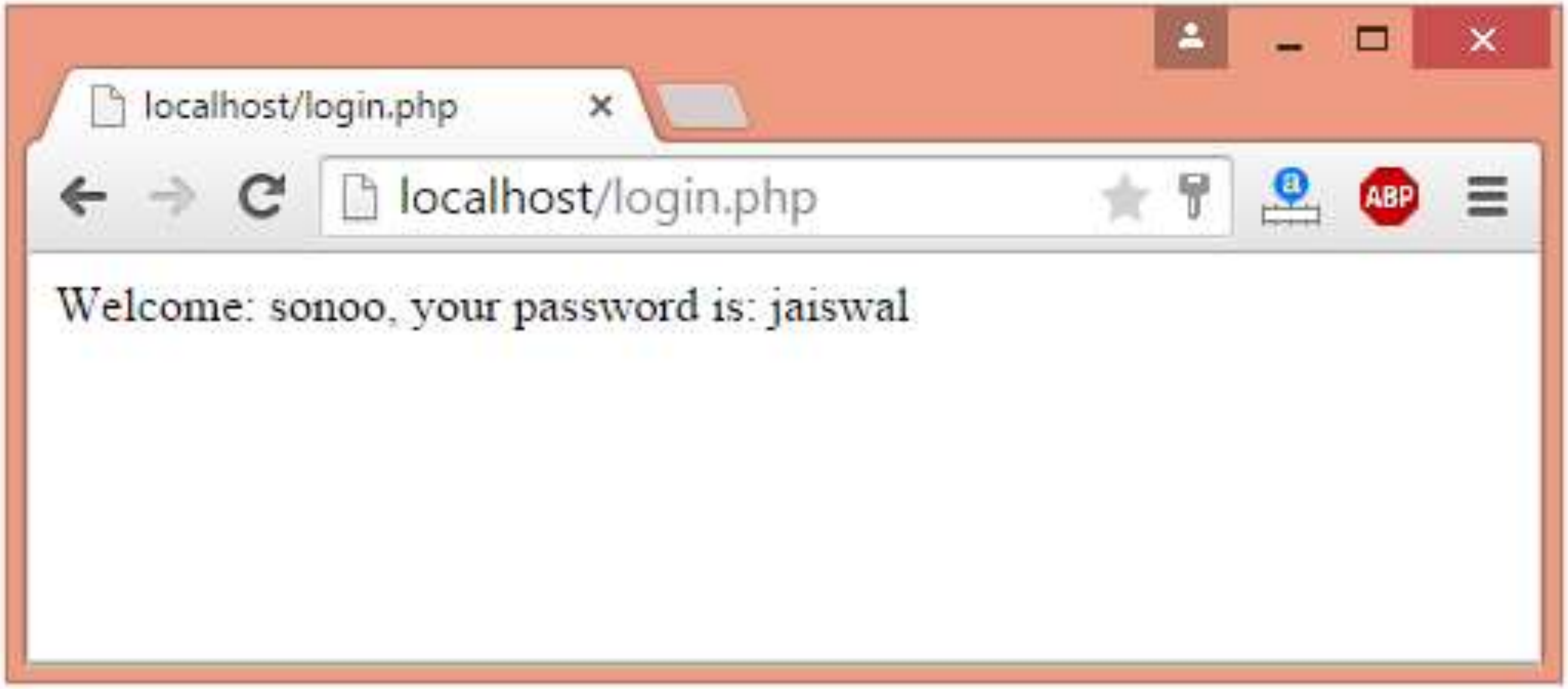
- *File: login.php*

```
<?php
$name=$_POST["name"];//receiving name field value in $name variable
$password=$_POST["password"];//receiving password field value in $pas
sword variable
echo "Welcome: $name, your password is: $password";
?>
```



Name: sonoo

Password: •••••••



Welcome: sonoo, your password is: jaiswal

Include Files

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with **the include or require** statement.

Include(“opendb.php”);

require (“closedb.php”);

- **The include and require statements are identical, except upon failure:**
  - require will produce a fatal error (E_COMPILE_ERROR) and stop the script
  - include will only produce a warning (E_WARNING) and the script will continue

This inserts files; the code in files will be inserted into current code. This will provide useful and protective means once you connect to a database, as well as for other repeated functions.

User-Defined functions

- A user-defined function declaration starts with the word function:
- function *functionName*() {
      code to be executed;
  }
- Function names are NOT case-sensitive.
- <?php

```
function writeMsg() {
    echo "Hello world!";
}
writeMsg(); // call the function
?>
```

-
  ```
  <?php
  function familyName($fname, $year) {
      echo "$fname Refsnes. Born in $year
  <br>";
  }

  familyName("Hege", "1975");
  familyName("Stale", "1978");
  familyName("Kai Jim", "1983");
  ?>
  ```

```php
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

-
  ```
  <?php
  function sum($x, $y) {
      $z = $x + $y;
      return $z;
  }

  echo "5 + 10 = " . sum(5, 10) . "<br>";
  echo "7 + 13 = " . sum(7, 13) . "<br>";
  echo "2 + 4 = " . sum(2, 4);
  ?>
  ```

call by value

- In case of PHP call by value, actual value is not modified if it is modified inside the function.

```php
<?php
function adder($str2)
{
    $str2 .= 'Call By Value';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

call by reference

- In case of PHP call by reference, actual value is modified if it is modified inside the function.
- In such case, you need to use & (ampersand) symbol with formal arguments. The & represents reference of the variable.

```
<?php
function adder(&$str2)
{
    $str2 .= 'Call By Reference';
}
$str = 'This is ';
adder($str);
echo $str;
?>
```

USER-DEFINED FUNCTIONS

PHP Variable Scopes

- The scope of a variable is the part of the script where the variable can be referenced/used.
- PHP has four different variable scopes:
  - local
  - global
  - static
  - Parameter

Local Scope

- A variable declared **within** a PHP function is local and can only be accessed within that function:

Example

```
< ?php
    $x=5; // global scope
    function myTest()
    {
    echo $x; // local scope
    }
    myTest();
    ?>
```

- The script above will not produce any output because the echo statement refers to the local scope variable $x, which has not been assigned a value within this scope.
- You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.
- Local variables are deleted as soon as the function is completed.

Global Scope

- A variable that is defined outside of any function, has a global scope.
- Global variables can be accessed from any part of the script, EXCEPT from within a function.
- To access a global variable from within a function, use the **global** keyword:

Example

-
  ```
  < ?php
  $x=5; // global scope
  $y=10; // global scope
  function myTest()
  {
  global $x,$y;
  $y=$x+$y;
  }
  myTest();
  echo $y; // outputs 15
  ?>
  ```

- PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.
- The example above can be rewritten like this:

-
  ```
  < ?php
  $x=5;
  $y=10;
  function myTest()
  {
  $GLOBALS['y']=$GLOBALS['x']+$GLOBALS['y'];
  }
  myTest();
  echo $y;
  ?>
  ```

Static Scope

- When a function is completed, all of its variables are normally deleted. However, sometimes you want a local variable to not be deleted.
- To do this, use the **static** keyword when you first declare the variable:

Example

-
  ```
  < ?php
  function myTest()
  {
  static $x=0;   echo $x;
  $x++;
  }
  myTest(); myTest(); myTest();
  ?>
  ```
- Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Parameter Scope

- A parameter is a local variable whose value is passed to the function by the calling code.
- Parameters are declared in a parameter list as part of the function declaration:

Example

```
< ?php
   function myTest($x)
   {
   echo $x;
   }
   myTest(5);
   ?>
```

- Parameters are also called arguments.

PHP Session Management

- A session is a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the users computer.
- When you work with an application, you open it, do some changes, and then you close it. This is much like a Session.
- The computer knows who you are. It knows when you start the application and when you end.
- But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.
- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.
- Session variables hold information about one single user, and are available to all pages in one application.

- A session is started with the session_start() function.
- Session variables are set with the PHP global variable: $_SESSION.
- "demo_session1.php"
-
```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>
```

- The session_start() function must be the very first thing in your document. Before any HTML tags.
- Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).
- Also notice that all session variable values are stored in the global $_SESSION variable.
-
```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
</body>
</html>
```

-
  ```
  <?php
  session_start();
  ?>
  <!DOCTYPE html>
  <html>
  <body>
  <?php
  print_r($_SESSION);
  ?>
  </body>
  </html>
  ```
- Output

-
  ```
  Array ( [favcolor] => green [favanimal] => cat )
  ```

-
  ```
  <?php
  session_start();
  ?>
  <!DOCTYPE html>
  <html>
  <body>
  <?php
  // to change a session variable, just overwrite
  it
  $_SESSION["favcolor"] = "yellow";
  print_r($_SESSION);
  ?>
  </body>
  </html>
  ```

- To remove all global session variables and destroy the session, use session_unset() and session_destroy().

-
  ```
  <?php
  session_start();
  ?>
  <!DOCTYPE html>
  <html>
  <body>
  <?php
  // remove all session variables
  session_unset();
  // destroy the session
  session_destroy();
  ?>
  </body>
  </html>
  ```

Cookie management

- A cookie is a small file with the maximum size of 4KB that the web server stores on the client computer.
- They are typically used to keeping track of information such as a username that the site can retrieve to personalize the page when the user visits the website next time.
- A cookie can only be read from the domain that it has been issued from.
- Cookies are usually set in an HTTP header but JavaScript can also set a cookie directly on a browser.
- There are three steps involved in identifying returning users −
  - Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
  - Browser stores this information on local machine for future use.
  - When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

- <?php

setcookie(cookie_name, cookie_value, [expiry_time], [cookie_path], [domain], [secure]);

- ?>
- PHP provided **setcookie()** function to set a cookie. This function requires upto six arguments and should be called before <html> tag.
- **Name** − This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** − This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** − This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** − This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** − This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security** − This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

```
<!DOCTYPE html>
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($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];
}
?>
</body>
</html>
```

```
<?php
   setcookie("name", "John Watkin", time()+3600, "/","", 0);
   setcookie("age", "36", time()+3600, "/", "",  0);
?>
<html>

   <head>
      <title>Setting Cookies with PHP</title>
   </head>

   <body>
      <?php echo "Set Cookies"?>
   </body>

</html>
```

```
<html>
   <head>
      <title>Accessing Cookies with PHP</title>
   </head>
   <body>
      <?php
         echo $_COOKIE["name"]. "<br />";

         /* is equivalent to */
         echo $HTTP_COOKIE_VARS["name"]. "<br />";

         echo $_COOKIE["age"] . "<br />";

         /* is equivalent to */
         echo $HTTP_COOKIE_VARS["age"] . "<br />";
      ?>
   </body>
</html>
```

```
<?php
   setcookie( "name", "", time()- 60, "/","", 0);
   setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>

   <head>
      <title>Deleting Cookies with PHP</title>
   </head>

   <body>
      <?php echo "Deleted Cookies" ?>
   </body>

</html>
```

Built-In-Functions

echo and print Statements

- They are both used to output data to the screen.
- The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions.
- echo can take multiple parameters (although such usage is rare) while print can take one argument.
- echo is marginally faster than print.
- The echo statement can be used with or without parentheses: echo or echo().

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>
```

- The print statement can be used with or without parentheses: print or print().

-
  ```
  <?php
  $txt1 = "Learn PHP";
  $txt2 = "W3Schools.com";
  $x = 5;
  $y = 4;

  print "<h2>" . $txt1 . "</h2>";
  print "Study PHP at " . $txt2 . "<br>";
  print $x + $y;
  ?>
  ```

printf() and sprint()

- The printf() function outputs a formatted string.
- The arg1, arg2, … argn parameters will be inserted at percent (%) signs in the main string. This function works "step-by-step". At the first % sign, arg1 is inserted, at the second % sign, arg2 is inserted, etc.
- printf(*format string,arg1,arg2,…… argn*)

Parameter	Description
format	Required. Specifies the string and how to format the variables in it.Possible format values: •%% - Returns a percent sign •%b - Binary number •%c - The character according to the ASCII value •%d - Signed decimal number (negative, zero or positive) •%e - Scientific notation using a lowercase (e.g. 1.2e+2) •%E - Scientific notation using a uppercase (e.g. 1.2E+2) •%u - Unsigned decimal number (equal to or greather than zero) •%f - Floating-point number (local settings aware) •%F - Floating-point number (not local settings aware) •%g - shorter of %e and %f •%G - shorter of %E and %f •%o - Octal number •%s - String •%x - Hexadecimal number (lowercase letters) •%X - Hexadecimal number (uppercase letters) Additional format values. These are placed between the % and the letter (example %.2f): •+ (Forces both + and - in front of numbers. By default, only negative numbers are marked) •' (Specifies what to use as padding. Default is space. Must be used together with the width specifier. Example: %'x20s (this uses "x" as padding) •- (Left-justifies the variable value) •[0-9] (Specifies the minimum width held of to the variable value) •.[0-9] (Specifies the number of decimal digits or maximum string length).
arg1	Required. The argument to be inserted at the first %-sign in the format string
arg2	Optional. The argument to be inserted at the second %-sign in the format string
	Optional. The argument to be inserted at the third, fourth, etc. %-sign in the format strin

String functions –

- strlen(string) – returns the number of characters in the string.
- Trim(string [,character set]) – removes white spaces and related characters from the start and end of the string. We can supply a list of characters to remove as the second parameter.
- By default t the following characters are removed: space, tab, new line, carriage return, vertical tab, null

- Substr(string, start [,length]) - The substr() function returns a part of a string.

Parameter	Description
string	Required. Specifies the string to return a part of
start	•Required. Specifies where to start in the stringA positive number - Start at a specified position in the string •A negative number - Start at a specified position from the end of the string •0 - Start at the first character in string
length	•Optional. Specifies the length of the returned string. Default is to the end of the string.A positive number - The length to be returned from the start parameter •Negative number - The length to be returned from the end of the string

- ucfirst(string) – returns a copy of the string with the first character converted to upper case.
- ucwords(string) – returns a copy of the string with the first character of each word converted to uppercase. Words are delimited by whitespace.
- strtolower(string)
- strtoupper(string)
- strstr(string, search_term) – if the search term is found the substring from its beginning to the end of the string is returned.

- strcasecmp(string, string2) – lexigraphical comparison.
  - 0 - if the two strings are equal
  - <0 - if string1 is less than string2
  - >0 - if string1 is greater than string2
- strtok(*string,split*) - The strtok() function splits a string into smaller strings (tokens).

  -
    ```
    <?php
    $string = "Hello world. Beautiful day today.";
     $token = strtok($string, " ");
    while ($token !== false)
       {
       echo "$token<br>";
       $token = strtok(" ");
       }
    ?>
    ```

- The str_replace(*find,replace,string,count*) function replaces some characters with some other characters in a string.
- This function works by the following rules:
  - If the string to be searched is an array, it returns an array
  - If the string to be searched is an array, find and replace is performed with every array element
  - If both find and replace are arrays, and replace has fewer elements than find, an empty string will be used as replace
  - If find is an array and replace is a string, the replace string will be used for every find value
  - A variable that counts the number of replacements

- explode(*separator*,*string*,*limit*) - The explode() function breaks a string into an array.

Parameter	Description
separator	Required. Specifies where to break the string
string	Required. The string to split
limit	Optional. Specifies the number of array elements to return. Possible values: •Greater than 0 - Returns an array with a maximum of *limit* element(s) •Less than 0 - Returns an array except for the last -*limit* elements() •0 - Returns an array with one element

- implode(*separator*,*array*) - The implode() function returns a string from the elements of an array. Returns a string from elements of an array.
- Join – an alias for implode.
- strip_tags(*string*,*allow*) - The strip_tags() function strips a string from HTML, XML, and PHP tags.
  - string Required. Specifies the string to check
  - allow Optional. Specifies allowable tags. These tags will not be removed

- Math Function
    - Date and Time
- The date/time functions allow you to get the date and time from the server where your PHP script runs.
- date(format [,timestamp]) – formats a time stamp as a date. If no timestamp is supplied, the current time is used.
- Echo (date(“l dS F, Y”));

Character	Meaning	Character	Meaning
a	Display am or pm	A	Display AM or PM
B	Swatch Internet time (invented by the Swiss watch-makers Swatch)	d	Day of the month as a pair of digits
D	Day of the week as three letters	F	Month as text
g	Hour in 12 hour format with no leading 0	G	Hour in 24 hour format with no leading 0
h	Hour in 12 hour format *with* leading 0	H	Hour in 24 hour format with leading 0
i	Minutes	j	Day of the month as integer
l	Day of the week in long text format	m	Month as integer
M	Month as three letters	n	Month as integer without leading 0
r	Date formatted according to RFC 822	s	Seconds
S	Ordinal suffix (such as "th" or "nd")	t	Number of days in the month
T	Timezone of the machine	U	Seconds since midnight on January 1st, 1970
w	Day of the week as integer	Y	Year in four digit format
y	Year in two digit format	z	Day of the year as integer

Table 12.4 Time and date formatting characters

- Getdate([timestamp]) – returns an associative array containing the elements of the date. Keys are:
    - Seconds, minutes, hours, mday(day of month), wday(day of week), mon(numeric month), year(numeric year), yday(day of the year as an integer), weekday(as text, using the full name of the day), month(as text using the full name).
- localtime([timestamp[, associative]]) – returns an array containing the elements of the timestamp. If associative value is TRUE then will return associative array. Keys are:
    - Tm_sec, min, hour, mday, mon, year, wday, yday

- mktime(hour, min, sec, month, day, year) – returns timestamp in seconds for the given date and time.

PHP Regular Expressions

- Regular expressions are powerful pattern matching algorithm that can be performed in a single expression.
- Regular expressions use arithmetic operators such as (+,-,^) to create complex expressions.
- Regular expressions help you accomplish tasks such as validating email addresses, IP address etc.
- Regular expressions simplify identifying patterns in string data by calling a single function. This saves us coding time.
- When validating user input such as email address, domain names, telephone numbers, IP addresses,
- Highlighting keywords in search results
- When creating a custom HTML template. Regular expressions can be used to identify the template tags and replace them with actual data.

preg_match(pattern, string[, matches])

- preg_match – this function is used to perform a pattern match on a string. It returns true if a match is found and false if a match is not found.
- Pattern - The pattern to search for, as a string.
- String – input string
- Matches - If matches is provided, then it is filled with the results of search. $matches[0] will contain the text that matched the full pattern, $matches[1] will have the text that matched the first captured parenthesized subpattern, and so on.

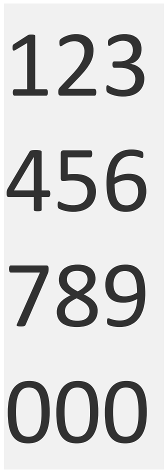
- <?php
  preg_match('/(foo)(bar)(baz)/', 'foobarbaz', $
  matches);
  print_r($matches);
  ?>

```
Array ( [0] => Array ( [0] => foobarbaz [1] => 0 )
[1] => Array ( [0] => foo [1] => 0 )
 [2] => Array ( [0] => bar [1] => 3 )
 [3] => Array ( [0] => baz [1] => 6 ) )
```

- preg_match_all(pattern, string)
- Works like preg_match() but matches all occurrences of the pattern in the string.
  - Preg_replace(pattern, replacement, string)
- If the regex pattern is found in the string, it is replaced by the string supplied as the second parameter.

preg_split(pattern, string)

- preg_split – this function is used to perform a pattern match on a string and then split the results into a numeric array
- <?php
-    $ip = "123.456.789.000"; // some IP address
-    $iparr = preg_split ("/\./", $ip);
-    print "$iparr[0] <br />";
-    print "$iparr[1] <br />" ;
-    print "$iparr[2] <br />" ;
-    print "$iparr[3] <br />" ;
- ?>



```
123
456
789
000
```

Error Handling

- Error handling is the process of catching errors raised by your program and then taking appropriate action. If you would handle errors properly then it may lead to many unforeseen consequences.
- We will show different error handling methods:
  - Simple "die()" statements
  - Custom errors and error triggers
  - Error reporting
- **Die()**
- The die function combines the echo and exit function in one. It is very useful when we want to output a message and stop the script execution when an error occurs..
- die($message)
- Example

```
if (isset($_POST['submit']))
{
    <?php $name = $_POST['name'];
    $phone = $_POST['phone'];
    if (is_numeric($name))//checking if the value is numeric
    {
       die("Name must not be numeric!");//error message if the value
       is numeric
    }
    else
    {
         echo "<br />Name entered is:".$_POST['name'];
    }
    if (!is_numeric($phone))//checking if the value is not numeric
    {
          die("<br />Phone no must be numeric!");//error message if
               the value is not numeric
    }
    else
    {
       echo "<br />Phone No is:".$_POST['phone'];
    }
}
?>
```

- **Custom Error handling**: Creating a custom error handler in PHP is quite simple. Create a function that can be called when a error has been occurred in PHP.
- error_function($error_level, $error_message, $error_file, $error_line, $error_context)
  - $error_level: It is required parameter and it must be an integer. There are predefined error levels.
  - $error_message: It is required parameter and it is the message which user want to print.
  - $error_file: It is optional parameter and used to specify the file in which error has been occurred.
  - $error_line: It is optional parameter and used to specify the line number in which error has been occurred.
  - $error_context: It is optional parameter and used to specify an array containing every variable and their value when error has been occurred.

- error_level: These are the possible error level which are listed below:
  - E_ERROR :fatal runtime error execution of script has been halted
  - E_WARNING :non fatal runtime error execution of script has been halted
  - E_PARSE :compile time error it is generated by the parser
  - E_NOTICE :The script found something that might be an error
  - E_CORE_ERROR :Fatal errors that occurred during initial startup of script
  - E_CORE_WARNING :Non fatal errors that occurred during initial startup of script
  - E_ALL :All errors and warning

- Can control which errors are displayed in script output with a built in function called error_reporting().
- This function accepts one or more of the named constants and tells the script to report only errors that match that type.
- int error_reporting ([int $level])
- <?php
- //only display fatal errors
- Error_reporting(E_ERROR);
- Echo 45/0;
- ?>

PHP Exception Handling

- Exceptions are used to change the normal flow of a script if a specified error occurs.
- Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.
- PHP provides following specialized keywords for this purpose.
  - **try:** It represent block of code in which exception can arise.
  - **catch:** It represent block of code that will be executed when a particular exception has been thrown.
  - **throw:** It is used to throw an exception. It is also used to list the exceptions that a function throws, but doesn’t handle itself.
  - **finally:** It is used in place of catch block or after catch block basically it is put for cleanup activity in PHP code.

```
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}
//trigger exception in a "try" block
try {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the number is 1 or below';
}
//catch exception
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}?>
```

- The checkNum() function is created. It checks if a number is greater than 1. If it is, an exception is thrown
- The checkNum() function is called in a "try" block
- The exception within the checkNum() function is thrown
- The "catch" block retrieves the exception and creates an object ($e) containing the exception information
- The error message from the exception is echoed by calling $e->getMessage() from the exception object

PHP $_REQUEST

- PHP $_REQUEST is a PHP super global variable which is used to collect data after submitting an HTML form.