Shree Adarsh BCA College-Botad

BCA SEM-5

Web Application Development Using PHP

Unit -2 Basic Of PHP

- Conditional Statement.
- Looping Statement.
- Array- Types of Array(Numeric, Associative, Multidimensional).
- PHP Server variables.
- Built-in-functions:

- **String**(print(),echo(),chr(),trim(),ltrim(),rtrim(),soundex(),str_word_count(),strcmp(),stristr(),strstr(),strlen(),strpos(),strrev(),substr(),strtoupper(),strtolower(),ucfirst(),ucword(), sucbstr_replace())
- **Mathametical**(abs(),sqrt(),log(),floor(),ceil(),pow(),max(),min())
- **Date/Time**(Date(),time(),getdate(),gettimeofday(),localtime(),checkdate())

Prepared By:

Maitree Shah

❖ Conditional Statements:

- Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.
- In PHP we have the following conditional statements:
  1) **if statement** - executes some code if one condition is true
  2) **if...else statement** - executes some code if a condition is true and
     Another code if that condition is false
  3) **if...elseif....else statement** - executes different codes for more than two conditions
  4) **switch statement** - selects one of many blocks of code to be executed

1) The if Statement

- The if statement executes some code if one condition is true.

Syntax

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

- The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

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

2) The if...else Statement

- The if....else statement executes some code if a condition is true and another code if that condition is false.

Syntax

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

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

Example

```
<?php
$t = 30;
if ($t <20)
{
    echo "Have a good day!";
}
Else
{
    echo "Have a good night!";
}
?>
```

3) The if...elseif....else Statement

- The if....elseif...else statement executes different codes for more than two conditions.

Syntax

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

- The example below 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

```
<?php
$t = 20;
if ($t < 10)
{
    echo "Have a good morning!";
}
elseif ($t <15)
{
    echo "Have a good day!";
}
else
{
    echo "Have a good night!";
}
?>
```

4) Switch Statement

- The switch statement is used to perform different actions based on different conditions.
- Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
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;
}
```

- This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once.
- The value of the expression is then compared with the values for each case in the structure.
- If there is a match, the block of code associated with that case is executed.
- Use break to prevent the code from running into the next case automatically.
- The default statement is used if no match is found.

Example

```
<?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!";
}
?>
```

Your favorite color is red!

❖ Looping Statement:

- Often when you write code, you want the same block of code to run over and over again in a row.
- Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.
- In PHP, we have the following looping statements:
  1) **while**- loops through a block of code as long as the specified condition istrue.
  2) do...while- loops through a block of code once, and then repeats theloop as long as the specified condition is true
  3) for - loops through a block of code a specified number of times
  4) foreach - loops through a block of code for each element in an array

1) The while Loop

- The while loop executes a block of code as long as the specified condition is true.

Syntax

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

- The example below first sets a variable $x to 1 ($x = 1).
- Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

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

Output

```
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
```

2) 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);
```

- The example below first sets a variable $x to 1 ($x = 1).
- Then, the do while loop will write some output, and then increment the variable $x with 1.
- Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

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

Output

```
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
```

- Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the

do while loop would execute its statements at least once, even if the condition is false the first time.

- The example below sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

Example

```
<?php
$x = 6;
do {
echo "The number is: $x <br>";
$x++;
} while ($x <= 5);
?>
```

Output

The number is: 6

3) for Loop

- PHP for loops execute a block of code a specified number of times.
- 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

The example below displays the numbers from 0 to 5:

Example

```
<?php
for ($x = 0; $x <= 5; $x++) {
    echo "The number is: $x <br>";
```

```
}
?>
```

Output

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
```

4) foreach Loop

- 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 $value and the array pointer is moved by one, until it reaches the last array element.
- The following example demonstrates a loop that will output the values of the given array ($colors):

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

Output

```
red
green
blue
yellow
```

✓ Break and Continue Statements

- Sometimes you may want to let the loops start without any condition, and allow the statements inside the brackets to decide when to exit the loop.
- There are two special statements that can be used inside loops: **Break** and **Continue.**
- The **Break** statement terminates the current While or For loop and continues executing the code that follows after the loop (if any).

Example:

```
<?php
echo "<p><b> Break statement:</b></p>";
for ($i=0; $i<=10; $i++)
{
    if ($i==3)
    {
        break;
    }
    echo "The number is ".$i; echo "<br />";
}
```

- The **Continue** statement terminates execution of the block of statements in a While or For loop and continues execution of the loop with the next iteration:

☐ Example:

```
<?php
echo "<p><b>Example of using the Continue
statement:</b><p>";
for ($i=0; $i<=10; $i++)
{
```

```
    if (i==3)
    {continue;}
    echo "The number is ".$i; echo "<br />";
}
  ?>
```

❖ Array:

- An array stores multiple values in one single variable:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] .
"."";
?>
```

I like Volvo, BMW and Toyota.

What is an Array?

- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
- The solution is to create an array!
- An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array

- In PHP, the array() function is used to create an array:

```
array();
```

Types of arrays:

- In PHP, there are three types of arrays:
  1) **Indexed arrays** - Arrays with a numeric index
  2) **Associative arrays** - Arrays with named keys
  3) **Multidimensional arrays** - Arrays containing one or more arrays

1) Indexed Array

- There are two ways to create indexed arrays:
- The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

- or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

- The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] .
".";
?>
```

I like Volvo, BMW and Toyota.

- Get The Length of an Array - The count() Function
- The count() function is used to return the length (the number of elements) of an array:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
```

```
echo count($cars);
?>
```
3

Loop Through an Indexed Array

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

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);
for($x = 0; $x < $arrlength; $x++) {
echo $cars[$x];
echo "<br>";
}
?>
```
Volvo
BMW
Toyota

2) Associative Arrays

- Associative arrays are arrays that use named keys that you assign to them.
- There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37",
"Joe"=>"43");
```
or:
```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```
- The named keys can then be used in a script:

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```
Peter is 35 years old.

Loop Through an Associative Array

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

Example

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

```
Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43
```

3) Multidimensional Arrays

- However, sometimes you want to store values with more than one key. This can be stored in multidimensional arrays.
- A multidimensional array is an array containing one or more arrays. PHP understands multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.
- **The dimension of an array indicates the number of indices you need to select an element.**
- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

Two-dimensional Arrays

- A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).
- First, take a look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

- We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array
(
array("Volvo",22,18),
array("BMW",15,13),
array("Saab",5,2),
array("Land Rover",17,15)
);
```

- Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.
- To get access to the elements of the $cars array we must point to the two indices (row and column):

Example

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold:
".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold:
".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold:
".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold:
".$cars[3][2].".<br>";
?>
```

Output

Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.

- We can also put a for loop inside another for loop to get the elements of the $cars array (we still have to point to the two indices):

Example

```
<?php
for ($row = 0; $row <4; $row++)
{
        echo "<p><b>Row number $row</b></p>";
        echo "<ul>";
        for ($col = 0; $col <3; $col++)
         {
              echo "<li>".$cars[$row][$col]."</li>";
        }
        echo "</ul>";
}
?>
```

Output

Row number 0
- Volvo
- 22
- 18

Row number 1
- BMW
- 15
- 13

Row number 2
- Saab
- 5
- 2

Row number 3
- Land Rover
- 17

- 15

❖PHP Global Variables - Superglobals:

- Several predefined variables in PHP 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.
- The PHP superglobal variables are:

1. $GLOBALS
2. $_SERVER
3. $_REQUEST
4. $_POST
5. $_GET
6. $_FILES
7. $_ENV
8. $_COOKIE
9. $_SESSION

1) $GLOBALS:

- $GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).
- PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.
- The example below shows how to use the super global variable $GLOBALS:

Example

```
<?php
$x = 75;
$y = 25;
function addition() {
$GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
```

```
echo $z;
?>
100
```

2)$_SERVER:

- $_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.
- The example below shows how to use some of the elements in $_SERVER:

Example

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

Output

```
php/simple.php
localhost
localhost
http://localhost/php/
Mozilla/5.0 (Windows NT 6.1; rv:70.0)
Gecko/20100101 Firefox/70.0
/php/simple.php
```

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the page from which the current page was called
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_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['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

3)$_REQUEST:

- PHP $_REQUEST is a PHP super global variable which is used to collect data after submitting an HTML form.
- The example below shows a form with an input field and a submit button.
- When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag.
- In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post"
action="<?phpecho$_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo"Name is empty";
    } else {
        echo $name;
    }
```

```
}
?>

</body>
</html>
```

4)$_POST:

- PHP $_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post".
- $_POST is also widely used to pass variables.
- The example below shows a form with an input field and a submit button.
- When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data.
- If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post"
action="<?phpecho$_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
```

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo"Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

5)$_GET:

- PHP $_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".
- $_GET is also widely used to pass variables.
- The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag.
- In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_GET to collect the value of the input field:

Example

```
<html>
<body>
```

```
<form method="get" action="<?php echo
$_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
<input type="submit" name="submit">
</form>

<?php
if(isset($_GET['submit']))
{
    // collect value of input field
    $name = $_GET['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

❖ Functions –Built in Functions:

A) String:

A string is a sequence of characters, like "Hello world!".

(1) print()

- It is used to display message on screen.
- **Syntax:** print(strings)

Parameter	Description
strings	Required. One or more strings to be sent to the output

- **Example:**

```
<?php
$str = "Who's Ramesh?";
print $str;
?>
```

The output of the code above will be:

Who's Ramesh?

(2) chr()

- Return character code for given ASCII code.
- **Syntax** :chr(ascii)

Parameter	Description
ascii	Required. An ASCII value

- **Example:**

```
<?php
echo chr(52)."<br />";
echo chr(052)."<br />";

?>
```

The output of the code above will be:

4
*

(3) echo()

- It is used to display message on screen.
- **Syntax:** echo(strings)

Parameter	Description
strings	Required. One or more strings to be sent to the output

- **Example:**

```
<?php
$str = "Who's Ramesh?";
```

```
echo $str;
?>
```

The output of the code above will be:

Who's Ramesh?

4) strlen():

- The PHP strlen() function returns the length of a string.
- The example below returns the length of the string "Hello world!":

Syntax

```
strlen(string)
```

Example

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

Output

```
12
```

5)str_word_count():

- The PHP str_word_count() function counts the number of words in a string.

Syntax

```
str_word_count(string)
```

Example

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

Output

```
2
```

6) strrev():

- The PHP strrev() function reverses a string:

Syntax

```
strrev(string)
```

Example

```
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

Output

```
!dlrow olleH
```

7) **strpos():**

- The PHP strpos() function searches for a specific text within a string.
- If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.
- The example below searches for the text "world" in the string "Hello world!":

Syntax

```
strpos(string,search_string)
```

Example

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

Output

```
6
```

Tip: The first character position in a string is 0 (not 1).

8) **str_replace():**

- The PHP str_replace() function replaces some characters with some other characters in a string.
- The example below replaces the text "world" with "Dolly":

Syntax

```
str_replace(string,string,original_string)
```

Example

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs
Hello Dolly!
?>
```

Output

Hello Dolly!

9) **chr():**

- The PHP chr() function Return characters from different ASCII values.
- The ASCII value can be specified in decimal, octal, or hex values. Octal values are defined by a leading 0, while hex values are defined by a leading 0x.

Syntax

```
chr(ascii)
```

Example

```
<?php
echo chr(52) . "<br>"; // Decimal value
echo chr(052) . "<br>"; // Octal value
echo chr(0x52) . "<br>"; // Hex value
?>
```

Output

```
4
*
R
```

10) **ltrim():**

- The PHP ltrim() function Removes whitespace or other predefined characters from the left side of a string:

Syntax

```
ltrim(string,trimed_characters)
```

Example

```
<?php
$str = "Hello World!";
echo $str . "<br>";
echo ltrim($str,"Hello");
?>
```

Output

```
Hello World!
World!
```

11) **rtrim():**

- The PHP rtrim() function Removes whitespace or other predefined characters from the right side of a string:
  - Syntax

    rtrim(string,trimed_characters)

  Example

```
<?php
$str = "Hello World!";
echo $str . "<br>";
echo rtrim($str," World!");
?>
```

  Output

```
Hello World!
Hello
```

12) **trim():**

- The PHP trim() function removes whitespace and other predefined characters from both sides of a string.

  Syntax

    trim(string,trimed_characters)

  Example

```
<?php
$str = "Hello World!";
echo $str . "<br>";
echo trim($str,"Hed!");
?>
```

  Output

```
Hello World!
llo Worl
```

13) **strcmp():**

- The PHP strcmp() function Compare two strings (case-sensitive):

  Syntax

strcmp(string,string)

Example

```
<?php
echo strcmp("Hello world!","Hello world!");
?>
```

Output

0

- If this function returns 0, the two strings are equal.

14) **substr_replace():**

- The substr_replace() function replaces a part of a string with another string.

Note: If the start parameter is a negative number and length is less than or equal to start, length becomes 0.

Note: This function is binary-safe.

Syntax

substr_replace(*string,replacement,start,length*)

Example

Replace "Hello" with "world":

```
<?php
echo substr_replace("Hello","world",0);
?>
```

Output

World

15) **stristr():**

- The stristr() function searches for the first occurrence of a string inside another string.

Note: This function is binary-safe.

Note: This function is case-insensitive. For a case-sensitive search,use strstr()function.

Syntax

stristr(*string,search*)

Example

- Find the first occurrence of "world" inside "Hello world!", and return the rest of the string:

```
<?php
echo stristr("Hello world!","WORLD");
echo stristr("Hello world!","o");
?
```

Output

```
world!
o world!
```

16) **strstr():**

- The strstr() function searches for the first occurrence of a string inside another string.

Note: This function is binary-safe.

Note: This function is case-sensitive. For a case-insensitive search, use stristr() function.

Syntax

strstr(*string,search*)

Example

- Find the first occurrence of "world" inside "Hello world!" and return the rest of the string:

```
<?php
echo strstr("Hello world!","world");
echo strstr("Hello world!","r");
?
```

Output

```
world!
rld!
```

17) **substr():**

- The substr() function returns a part of a string.

Note: If the start parameter is a negative number and length is less than or equal to start, length becomes 0.

Syntax

substr(*string,start,length*)

Example

- Return "world" from the string:

```
<?php
```

```
echo substr("Hello world",6);
?>
```

Output

World

18) **ucwords():**

- The ucwords() function converts the first character of each word in a string to uppercase.

Syntax

ucwords(*string*)

Example

- Convert the first character of each word to uppercase:

```
<?php
echo ucwords("hello world");
?>
```

Output

Hello World

19) **ucfirst():**

- The ucfirst() function converts the first character of a string to uppercase.

Syntax

ucfirst(*string*)

Example

- Convert the first character of "hello" to uppercase:

```
<?php
echo ucfirst("hello world!");
?>
```

Output

Hello world!

20) **strtoupper():**

- The strtoupper() function converts a string to uppercase.

Syntax

strtoupper(*string*)

Example

- Convert the "hello world!" to uppercase:

```
<?php
echo strtoupper("Hello world!");
?>
```

Output

HELLO WORLD!

21) **strtolower():**

- The strtoupper() function converts a string to uppercase.

Syntax

strtoupper(*string*)

Example

```
<?php
echo strtolower("Hello WORLD.");
?>
```

Output

hello world.

B) Mathametical:

1) **abs():**

- The abs() function returns the absolute (positive) value of a number.

Syntax

abs(*number*);

Example

- Return the absolute value of different numbers:

```
<?php
echo(abs(6.7) . "<br>");
echo(abs(-6.7) . "<br>");
echo(abs(-3) . "<br>");
echo(abs(3));
?>
```

Output

6.7

6.7
3
3

2)**ceil():**

- The ceil() function rounds a number up to the nearest integer which is grater than the number.

Syntax

ceil(*number*);

Example

- Round numbers up to the nearest integer:

```
<?php
echo(ceil(0.60) . "<br>");
echo(ceil(0.40) . "<br>");
echo(ceil(5) . "<br>");
echo(ceil(5.1) . "<br>");
echo(ceil(-5.1) . "<br>");
echo(ceil(-5.9));
?>
```

Output

1
1
5
6
-5
-5

3)**floor():**

- The floor() function rounds a number up to the nearest integer which is smaller than the number.

Syntax

floor(*number*);

Example

- Round numbers down to the nearest integer:

```
<?php
echo(floor(0.60) . "<br>");
```

```
echo(floor(0.40) . "<br>");
echo(floor(5) . "<br>");
echo(floor(5.1) . "<br>");
echo(floor(-5.1) . "<br>");
echo(floor(-5.9));
?>
```

Output

```
0
0
5
5
-6
-6
```

4) **sqrt():**

- The sqrt() function returns the square root of a number.

Syntax

```
sqrt(number);
```

Example

- Return the square root of different numbers:

```
<?php
echo(sqrt(0) . "<br>");
echo(sqrt(1) . "<br>");
echo(sqrt(9) . "<br>");
echo(sqrt(0.64) . "<br>");
echo(sqrt(-9));
?>
```

Output

```
0
1
3
0.8
NAN
```

5) log():

- The log() function returns the natural logarithm of a number, or the logarithm of *number* to *base*.

Syntax

log(*number,base*);

Example

- Return the natural logarithm of different numbers:

```
<?php
echo(log(2.7183) . "<br>");
echo(log(2) . "<br>");
echo(log(1) . "<br>");
echo(log(0));
?>
```

Output

```
1.000006684914
0.69314718055995
0
-INF
```

6) pow():

- The pow() function returns x raised to the power of y.

Syntax

pow(*x,y*);

Example

```
<?php
echo(pow(2,4) . "<br>");
echo(pow(-2,4) . "<br>");
echo(pow(-2,-4) . "<br>");
echo(pow(-2,-3.2));
?>
```

Output

```
16
16
0.0625
NAN
```

7) max():

- The max() function returns the highest value in an array, or the highest value of several specified values.

Syntax

```
max(array_values);
or
max(value1,value2,...);
```

Example

- Find highest value with the max() function:

```
<?php
echo(max(2,4,6,8,10) . "<br>");
echo(max(22,14,68,18,15) . "<br>");
echo(max(array(4,6,8,10)) . "<br>");
echo(max(array(44,16,81,12)));
?>
```

Output

```
10
68
10
81
```

8) min():

- The min() function returns the lowest value in an array, or the lowest value of several specified values.

Syntax

```
min(array_values);
or
min(value1,value2,...);
```

Example

- Find lowest value with the min() function:

```
<?php
echo(min(2,4,6,8,10) . "<br>");
echo(min(22,14,68,18,15) . "<br>");
echo(min(array(4,6,8,10)) . "<br>");
echo(min(array(44,16,81,12)));
?>
```

Output

2
14
4
12

C) Date/Time:

- The date/time functions allow you to get the date and time from the server where your PHP script runs. You can then use the date/time functions to format the date and time in several ways.

Note: These functions depend on the local settings of your server.

1)date():

The date() function formats a local time/date.

Syntax

date(format,timestamp)

Parameter	Description
format	Required. Specifies how to return the result: • d - The day of the month (from 01 to 31) • D - A textual representation of a day (three letters) • j - The day of the month without leading zeros (1 to 31) • l (lowercase 'L') - A full textual representation of a day • N - The ISO-8601 numeric representation of a day (1 for Monday through 7 for Sunday) • S - The English ordinal suffix for the day of the month (2 characters st, nd, rd or th. Works well with j) • w - A numeric representation of the day (0 for Sunday through 6 for Saturday) • z - The day of the year (from 0 through 365) • W - The ISO-8601 week number of year (weeks starting on Monday) • F - A full textual representation of a month (January through December) • m - A numeric representation of a month (from 01 to

	12) • M - A short textual representation of a month (three letters) • n - A numeric representation of a month, without leading zeros (1 to 12) • t - The number of days in the given month • L - Whether it's a leap year (1 if it is a leap year, 0 otherwise) • o - The ISO-8601 year number • Y - A four digit representation of a year • y - A two digit representation of a year • a - Lowercase am or pm • A - Uppercase AM or PM • B - Swatch Internet time (000 to 999) • g - 12-hour format of an hour (1 to 12) • G - 24-hour format of an hour (0 to 23) • h - 12-hour format of an hour (01 to 12) • H - 24-hour format of an hour (00 to 23) • i - Minutes with leading zeros (00 to 59) • s - Seconds, with leading zeros (00 to 59) • e - The timezone identifier (Examples: UTC, Atlantic/Azores) • I (capital i) - Whether the date is in daylights savings time (1 if Daylight Savings Time, 0 otherwise) • O - Difference to Greenwich time (GMT) in hours (Example: +0100) • T - Timezone setting of the PHP machine (Examples: EST, MDT) • Z - Timezone offset in seconds. The offset west of UTC is negative, and the offset east of UTC is positive (-43200 to 43200) • c - The ISO-8601 date (e.g. 2004-02-12T15:19:21+00:00) • r - The RFC 2822 formatted date (e.g. Thu, 21 Dec 2000 16:01:07 +0200) • U - The seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)
timestamp	Optional.

Example

```
<?php
echo("Result with date():<br />");
echo(date("l") . "<br />");
echo(date("l dS \of F Y h:i:s A") . "<br />");
?>
```

The output of the code above could be something like this:

```
Result with date ():
Friday
Friday 08th of February 2013 09:36:39 AM
```

2)time():

- The time() function returns the current time in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

Syntax

```
time();
```

Example

- Return the current time as a Unix timestamp, then format it to a date:

```
<?php
$t=time();
echo($t . "<br>");
echo(date("Y-m-d",$t));
?>
```

Output

```
1545624697
2018-12-23
```

3)getdate():

- The getdate() function returns date/time information of a timestamp or the current local date/time.

Syntax

```
getdate(timestamp);
```

Example

- Return date/time information of the current local date/time:

```
<?php
// Print the array from getdate()
```

```
print_r(getdate());
echo "<br><br>";
// Return date/time info of a timestamp; then format
the output
$mydate=getdate(date("U"));
echo "$mydate[weekday], $mydate[month]
$mydate[mday], $mydate[year]";
?>
```

Output

```
Array ( [seconds] => 33 [minutes] => 12 [hours] =>
23 [mday] => 23 [wday] => 0 [mon] => 12 [year]
=> 2018 [yday] => 356 [weekday] => Sunday
[month] => December )
Sunday, December 23, 2018
```

4) gettimeofday():

- The gettimeofday() function returns the current time.

Syntax

gettimeofday(*return_float);*

Example

- Return the current time:

```
<?php
// Return current time; then format the output
$mytime=gettimeofday();
echo "$mytime[sec].$mytime[usec]";
?>
```

Output

```
1545624833.345095
```

5) localtime():

- The localtime() function returns the local time.

Syntax

localtime(*timestamp,is_assoc);*

Example

- Print local time as an indexed and as an associative array:

```
<?php
```

```
print_r(localtime(time(),true));
?>
```

Output

```
Array ( [tm_sec] => 7 [tm_min] => 15 [tm_hour] =>
23 [tm_mday] => 23 [tm_mon] =>11 [tm_year] =>
118 [tm_wday] => 0 [tm_yday] => 356)
```

6) checkdate():

- The checkdate() function is used to validate a Gregorian date.

Syntax

```
checkdate(month,day,year);
```

Example

- Check if several dates are valid Gregorian dates:

```
<?php
var_dump(checkdate(12,31,-400));
echo "<br>";
var_dump(checkdate(2,29,2003));
echo "<br>";
var_dump(checkdate(2,29,2004));
?>
```

Output

```
bool(false)
bool(false)
bool(true)
```

Theory Assignment-2

1. **What is Array? Explain types of Array in php.**
2. **Explain any 15 string built in function in php.**

Practical Assignment-2

1. Write a php script to find maximum number from three numbers using ternary operator.
2. Write a php script to assign ages to the different person using array.
3. Write a php script to sort an array.
4. Write a php script to print the number using break statement.
5. Write a php script to print the number using continue statement.
6. Write php code that define an associative array with 10 values and display values using foreach loop.
7. PHP Program to Count All Array Elements
8. Write a program that define two dimensional two arrays of 3 rows and 3 coloums and display it's addition.
9. Write a php code that define two dimentional arrays of three rows and three coloumns and display it's multiplication.
10. PHP Program to Check if Two Arrays Contain Same Elements.
11. A code of php that display current running script and server name on which current script running.
12. Write a php script to use all Date function.
13. Write a php script to use all Maths function.
14. Write a php script to use all String function.