UNIT – III

WORKING WITH PHP ARRAYS & FUNCTIONS

PHP Arrays

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

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

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

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] . ".";
?>
```

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);
?>
```

PHP 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:

Example

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

Define Starting Index

To define starting index we need to specify it in the array function itself. By default starting index of an array is 0.

```
$numbers = array(2=>4, 6, 2, 22, 11);
```

Above example shows that starting index of an array $numbers is 2.

PHP - 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.

PHP - 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>";
?>
```

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):

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

PHP - Sort Functions For Arrays

- `sort()` - sort arrays in ascending order

  ```
  <?php
  $numbers = array(4, 6, 2, 22, 11);
  sort($numbers);
  ?>
  ```

- `rsort()` - sort arrays in descending order

  ```
  <?php
  $numbers = array(4, 6, 2, 22, 11);
  rsort($numbers);
  ?>
  ```

- `asort()` - sort associative arrays in ascending order, according to the value

  ```
  <?php
  $age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
  ```

```
asort($age);
?>
```

- `ksort()` - sort associative arrays in ascending order, according to the key

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
?>
```

- `arsort()` - sort associative arrays in descending order, according to the value

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

- `krsort()` - sort associative arrays in descending order, according to the key

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
?>
```

Functions

PHP User Defined Functions

Besides the built-in PHP functions, we can create our own functions.

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads.

A function will be executed by a call to the function.

Create a User Defined Function in PHP

A user-defined function declaration starts with the word `function`:

Syntax

```
function functionName() {
    code to be executed;
}
```

A function name can start with a letter or underscore (not a number) and Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // call the function
?>
```

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<?php
function familyName($fname) {
    echo "$fname <br>";
}

familyName("Shah");
familyName("Patel");
?>
```

The following example has a function with two arguments ($fname and $year):

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

familyName("Shah", "1975");
familyName("Patel", "1978");
?>
```

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<?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 Functions - Returning values

To let a function return a value, use the `return` statement:

Example

```
<?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);
?>
```

Passing with Value

Passing by value means passing the value directly to a function. The called function uses the value in a local variable; any changes to it **do not** affect the source variable.

```
<?php
//Call by value program
function abc($x)
{
$x=$x+10;
return($x);
}
$a=20;
echo abc($a)."<br>"; // a is 30 here
echo ($a); // a is 20 here
?>
```

Passing with Reference

The following things can be passed by reference:

- Variables, i.e. *foo($a)*

```
<?php
function foo(&$var)
{
    $var++;
}

$a=5;
foo($a);
// $a is 6 here
?>
```

- References returned from functions, i.e.:

```
<?php
function foo(&$var)
{
    $var++;
}
function &bar()
{
    $a = 5;
```

```
    return $a;
}
foo(bar());
?>
```

PHP variables are assigned by value, passed to functions by value, and when containing/representing objects are passed by reference. You can force variables to pass by reference using an &.

```
<?php
function changeValue(&$var)
{
   $var++;
}
$result=5;
changeValue($result);

echo $result; // $result is 6 here
?>
```

String Functions

Get The Length of a String

The PHP `strlen()` function returns the length of a string.

The example below returns the length of the string "Hello world!":

Example

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

Count The Number of Words in a String

The PHP `str_word_count()` function counts the number of words in a string:

Example

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

Reverse a String

The PHP `strrev()` function reverses a string:

Example

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

Search For a Specific Text Within a String

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!":

Example

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

Replace Text Within a String

The PHP `str_replace()` function replaces some characters with some other characters in a string.

The example below replaces the text "world" with "Dolly":

Example

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

Repeat a String

The PHP `str_repeat()` function repeats a string specified number of times.

Example

```php
<?php
echo str_repeat("world", 10); // Repeats world 10 no, of times
?>
```

Split a String

The PHP `str_split() and strtok()` function splits a string into an array.

Example

```
<?php
print_r(str_split("Hello"));
?>
```

Output: Array ([0] => H [1] => e [2] => l [3] => l [4] => o)

Splits a string “Hello” in array of length 1 because here length is not specified. Following example shows splitting a string “Hello” into array of length 3.

```
<?php
print_r(str_split("Hello",3));
?>
```

Output: Array ([0] => Hel [1] => lo)

```
<?php
$string = "Hello world. Beautiful day today.";
$token = strtok($string, " ");

while ($token !== false)
{
echo "$token<br>";
$token = strtok(" ");
}
?>
```

Output:

Hello
world.
Beautiful
day
today.

Shuffle a String

The PHP `str_shuffle()` function Randomly shuffle all characters of a string.

Example

```
<?php
echo str_shuffle("Hello World");
?>
```

Output: rHWl leodol

Output of shuffle is different all the time you will refresh the page.

Comparing Two Strings

The PHP `strcmp() and strcasecmp()` function Randomly shuffle all characters of a string.

strcasecmp()

Compare two strings (case-insensitive):

```
<?php
echo strcasecmp("Hello world!","HELLO WORLD!");
?>
```

```
Output: 0 //means both strings are equal
```

strcmp()

Compare two strings (case-sensitive):

```
<?php
echo strcmp("Hello world!","HELLO WORLD!");
?>
Output: -1 //means both strings are not equal
```

Find Position of String

Returns the position of the first occurrence of a string inside another string (case-sensitive)

```
<?php
echo strpos("Original String","String");
?>
```

```
Output: 8
```

Changing case of String

strtoupper()

Convert all characters to uppercase:

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

strtolower()

Convert all characters to lowercase:

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

Substring

substr() Returns a part of the string

```
<?php
echo substr("Hello world",6); //return a string starting from index 6
?>
```

```
Output: world
```

Removing unwanted characters or whitespaces

trim() Removes whitespace (blank, tab, newline) or other characters from both sides of a string.

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

Date Time Functions

Date() Function

The PHP `date()` function formats a timestamp to a more readable date and time.

Syntax

`date(`*`format,timestamp`*`)`

format : Required. Specifies the format of the timestamp

timestamp: Optional. Specifies a timestamp. Default is the current date and time

The required *format* parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)
- l (lowercase 'L') - Represents the day of the week

Here are some characters that are commonly used for times:

- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting. The example below formats today's date in three different ways:

```php
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
echo "The time is " . date("h:i:sa");
?>
```

Create a Date With PHP mktime()

The `mktime()` function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

Syntax

```
mktime(hour,minute,second,month,day,year)
```

The example below creates a date and time from a number of parameters in the `mktime()` function:

Example

```
<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is " . date("Y-m-d h:i:sa", $d); //o/p: Created date is 2014-08-12 11:14:54am
?>
```

getdate() Function

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

```
<?php
print_r(getdate());
?>
```

output: Array ([seconds] => 51 [minutes] => 29 [hours] => 8 [mday] => 9 [wday] => 0 [mon] => 9 [year] => 2018 [yday] => 251 [weekday] => Sunday [month] => September [0] => 1536496191)

checkdate() Function

Syntax

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

month: Required. Specifies the month as a number between 1 and 12

day: Required. Specifies the day as a number between 1 and 31

year: Required. Specifies the year as a number between 1 and 32767

Return Value: TRUE if the date is valid. FALSE otherwise

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

time() Function

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

```
<?php
$t=time();
echo($t . "<br>"); // 1536496945
echo(date("Y-m-d",$t)); // 2018-09-09
?>
```

Common Mathematical Functions

abs(): 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

base_convert(): The base_convert() function converts a number from one number base to another.

Syntax: base_convert(*number,frombase,tobase*);

```
<?php
$hex = "E196";
echo base_convert($hex,16,8);
?>
```

ceil():Round numbers up to the nearest integer.

```
<?php
echo(ceil(0.60) . "<br>"); //1
echo(ceil(0.40) . "<br>"); //1
echo(ceil(5) . "<br>"); //5
echo(ceil(5.1) . "<br>"); //6
echo(ceil(-5.1) . "<br>"); //-5
echo(ceil(-5.9)); //-5
?>
```

floor():Round numbers down to the nearest integer.

```
<?php
echo(floor(0.60) . "<br>"); //0
echo(floor(0.40) . "<br>"); //0
echo(floor(5) . "<br>"); //5
echo(floor(5.1) . "<br>"); //5
echo(floor(-5.1) . "<br>"); //-6
echo(floor(-5.9)); //-6
?>
```

exp(): Return 'e' raised to the power of different numbers.

```
<?php
echo(exp(0) . "<br>"); //1
echo(exp(1) . "<br>"); // 2.718281828459
?>
```

fmod(): Return the remainder of x/y.

```
<?php
$x = 7;
$y = 2;
$result = fmod($x,$y);
echo $result; //1
?>
```

log():Return the natural logarithm of different numbers

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

log10():Return the base-10 logarithm of different numbers

```
<?php
echo(log10(2) . "<br>");
echo(log10(1) . "<br>");
echo log10(0);
?>
```

max(): Find highest value with the max() function

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

min(): Find lowest value with the min() function

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

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

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

pi():Returns the value of pi.

```
<?php
echo(pi());
?>
```

rand(): Returns the random value.

```
<?php
echo(rand() . "<br>"); //returns the any random value
echo(rand(10,100)); //returns the random value from 10 to 100
?>
```

round(): Rounds the numbers.

```
<?php
echo(round(0.60) . "<br>");
echo(round(0.49) . "<br>");
echo(round(-4.40) . "<br>");
echo(round(-4.60));
?>
```

sqrt(): Returns the squareroot of different numbers.

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