UNIT-2

➢ Arrays in PHP

Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable thereby saving us the effort of creating a different variable for every data. The arrays are helpful to create a list of elements of similar types, which can be accessed using their index or key. An array is created using an **array()** function in PHP.

There are basically three types of arrays in PHP:

- **Indexed or Numeric Arrays:** An array with a numeric index where values are stored linearly.
- **Associative Arrays:** An array with a string index where instead of linear storage, each value can be assigned a specific key.
- **Multidimensional Arrays:** An array which contains single or multiple array within it and can be accessed via multiple indices.

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:

1st way:

```
$season=array("summer","winter","spring","autumn");
```

2nd way:

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

array1.php

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

```
?>
```

Output:

```
Season are: summer, winter, spring and autumn
```

PHP Associative Array

These types of arrays are similar to the indexed arrays but instead of linear storage, every value can be assigned with a user-defined key of string type.

We can associate name with each array elements in PHP using => symbol.

There are two ways to define associative array:

1st way:

```
$salary=array("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");
```

2nd way:

```
$salary["Sonoo"]="350000";
$salary["John"]="450000";
$salary["Kartik"]="200000";
```

Example

File: arrayassociative1.php

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

Output:

```
Sonoo salary: 350000
John salary: 450000
Kartik salary: 200000
```

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 Multidimensional Array Example

Let's see a simple example of PHP multidimensional array to display following tabular data. In this example, we are displaying 3 rows and 3 columns.

Id	Name	Salary
1	sonoo	400000
2	john	500000
3	rahul	300000

multiarray.php

```
<?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
```

➢ PHP Array Functions

PHP provides various array functions to access and manipulate the elements of array. The important PHP array functions are given below.

1) PHP array() function

PHP array() function creates and returns an array. It allows you to create indexed, associative and multidimensional arrays.

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

Output:

```
Season are: summer, winter, spring and autumn
```

2) PHP array_change_key_case() function

PHP array_change_key_case() function changes the case of all key of an array.

Note: It changes case of key only.

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_change_key_case($salary,CASE_UPPER));
?>
```

Output:

```
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.

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"200000");
print_r(array_chunk($salary,2));
?>
```

Output:

```
Array (
[0] => Array ( [0] => 550000 [1] => 250000 )
[1] => Array ( [0] => 200000 )
)
```

4) PHP count() function

PHP count() function counts all elements in an array.

Example

```
<?php
$season=array("summer","winter","spring","autumn");
echo count($season);
?>
```

Output:

```
4
```

5) PHP sort() function

PHP sort() function sorts all the elements in an array.

Example

```
<?php
$season=array("summer","winter","spring","autumn");
```

```
sort($season);

foreach( $season as $s )

{

 echo "$s<br />";

}

?>
```

Output:

```
autumn

spring

summer

winter
```

6) PHP array_reverse() function

PHP array_reverse() function returns an array containing elements in reversed order.

Example

```
<?php

$season=array("summer","winter","spring","autumn");

$reverseseason=array_reverse($season);

foreach( $reverseseason as $s )

{

 echo "$s<br />";

}

?>
```

Output:

```
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.

Example

```
<?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.

Example

```
<?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
```

➢ PHP OOP - Classes and Objects

A class is a template for objects, and an object is an instance of class.

Let's assume we have a class named Fruit. A Fruit can have properties like name, color, weight, etc. We can define variables like $name, $color, and $weight to hold the values of these properties.

When the individual objects (apple, banana, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Define a Class

A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods go inside the braces:

Syntax

```
<?php
class Fruit {
  // code goes here...
}
?>
```

Below we declare a class named Fruit consisting of two properties ($name and $color) and two methods set_name() and get_name() for setting and getting the $name property:

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Define Objects

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Objects of a class are created using the new keyword.

In the example below, $apple and $banana are instances of the class Fruit:

Example

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

Apple

Banana

In the example below, we add two more methods to class Fruit, for setting and getting the $color property:

Example

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

```
Name:Apple

Color: Red
```

➢ PHP String Functions

We have to learn about the in-built functions for string processing in PHP.

In general practice, using the right string function will save you a lot of time as they are pre-defined in PHP libraries and all you have to do is call them to use them.

Below we have a list of some commonly used string functions in PHP:

strlen($str)

This function returns the length of the string or the number of characters in the string including whitespaces.

```
<?php
$str = "Welcome to Studytonight";
echo "Length of the string is: ". strlen($str);
?>
```

Length of the string is: 23

str_word_count($str)

This function returns the number of words in the string. This function comes in handly in form field validation for some simple validations.

```
<?php
$str = "Welcome to Studytonight";
echo "Number of words in the string are: ". str_word_count($str);
?>
```

Number of words in the string are: 3

strrev($str)

This function is used to reverse a string.

Let's take an example and see,

```
<?php
$str = "Welcome to Studytonight";
echo "Reverse: ". strrev($str);
?>
```

Copy

Reverse: thginotydutS ot emocleW

strpos($str, $text)

This function is used to find the position of any text/word in a given string. Just like an array, string also assign index value to the characters stored in it, starting from zero.

```
<?php
$str = "Welcome to Studytonight";
```

```
echo "Position of 'Studytonight' in string: ". strpos($str, 'Studytonight');
?>
```

Position of 'Studytonight' in string: 11

str_replace($replacethis, $replacewith, $str)

This function is used to replace a part of the string with some text. While using this function, the first argument is the part of string that you want to replace, second argument is the new text you want to include, and the last argument is the string variable itself.

Let's take an example,

```
<?php
$str = str_replace("Studytonight", "Studytonight.com", "Welcome to Studytonight");
echo $str;
?>
```

Welcome to Studytonight.com

ucwords($str)

This function is used for formatting the string. This function converts first letter/character of every word in the string to uppercase.

Let's take an example and see,

```
<?php
$str = "welcome to studytonight";
echo ucwords($str);
?>
```

Welcome To Studytonight

strtoupper($str)

To convert every letter/character of every word of the string to uppercase, one can use strtoupper() method.

```
<?php
$str = "welcome to studytonight";
echo strtoupper($str);

?>
```

WELCOME TO STUDYTONIGHT

strtolower($str)

This function is used to convert every letter/character of a string to lowercase.

```
<?php
$str = "WELCOME TO STUDYTONIGHT";
echo strtolower($str);
?>
```

welcome to studytonight

str_repeat($str, $counter)

This function is used to repeat a string a given number of times. The first argument is the string and the second argument is the number of times the string should be repeated.

```
<?php
$str = "Studytonight";
echo str_repeat($str, 4);
?>
```

StudytonightStudytonightStudytonightStudytonight

strcmp($str1, $str2)

This function is used to compare two strings. The comparison is done alphabetically. If the first string is greater than second string, the result will be greater than 0, if the first string is equal to the second string, the result will be equal to 0 and if the second string is greater than the first string, then the result will be less than 0.

```
<?php
$str1 = "Studytonight";
$str2 = "Studytonight.com";
// comparing str1 and str2
echo strcmp($str1, $str2);
// comparing str2 and str1
echo strcmp($str2, $str1);
// comparing str1 with str1
echo strcmp($str1, $str1);
?>
```

```
-4
4
0
```

substr($str, $start, $length)

This function is used to take out a part of the string(substring), starting from a particular position, of a particular length.

The first argument is the string itself, second argument is the starting index of the substring to be exracted and the third argument is the length of the substring to be extracted.

```
<?php
$str = "Welcome to Studytonight";
echo substr($str, 11, 12);
?>
```

```
Studytonight
```

trim($str, charlist)

This function is used to remove extra whitespaces from beginning and the end of a string. The second argument charlist is optional. We can provide a list of character, just like a string, as the second argument, to trim/remove those characters from the main string.

```
<?php
$str1 = "   Hello World   ";
echo trim($str1) . "<br/>";
$str2 = "Hello Hello";
echo trim($str2,"Heo");
?>
```

```
Hello World
llo Hell
```

As you can see in the output, additional spaces from the beginning and end are removed and in the second case, the characters specified are removed from the beginning and the end of the string.

explode(separator, $str, $limit)

This function is used to break a string, create an array of the broken parts of the string and return the array. The first argument, **separator** defines where to break the string from. It can be a space, hiphen(-) or any other character.

The second argument of this function is the string itself and the third argument is the **limit**, which specifies the number of array elements to return. the third argument is optional.

Let's have an example,

```
<?php


$str = "Its a beautiful day";
   print_r(explode(" ", $str));
?>
```

```
Array (
   [0] => Its
   [1] => a
   [2] => beautiful
   [3] => day
)
```

In the example above, we have provided **space** as separator to break the string and return an array.

If we provide the third argument **limit** as well, we can limit the number of array elements returned. For example, if we provide 2 as the third argument, then we will only get 2 elements in the array, the first two.

implode(separator, $arr)

This function is used to form a string using the array elements from the array provided and join them using the **separator**.

Let's take an example,

```
<?php
$arr = array("Its", "a", "beautiful", "day");
// <br> is used to jump to next line
echo implode(" ", $arr) . "<br>";
echo implode("-", $arr) . "<br>";
echo implode("/", $arr) . "<br>";
?>
```

```
Its a beautiful day
Its-a-beautiful-day
```

```
Its/a/beautiful/day
```

➢ PHP Date and Time

- The date & time using the date() function in PHP, we will also see the various formatting options available with these functions & understand their implementation through the examples.

Date and time are some of the most frequently used operations in PHP while executing SQL queries or designing a website etc. PHP serves us with predefined functions for these tasks. Some of the predefined functions in PHP for date and time are discussed below.

PHP date() Function: The PHP date() function converts timestamp to a more readable date and time format.

The computer stores dates and times in a format called UNIX Timestamp, which measures time as a number of seconds since the beginning of the Unix epoch (midnight Greenwich Mean Time on January 1, 1970, i.e. January 1, 1970, 00:00:00 GMT). Since this is an impractical format for humans to read, PHP converts timestamp to a format that is readable and more understandable to humans.

Syntax:

date(format, timestamp)

Explanation:

- The format parameter in the date() function specifies the format of returned date and time.
- The timestamp is an optional parameter, if it is not included then the current date and time will be used.

Example: The below program explains the usage of the date() function in PHP.

```
<?php
 echo "Today's date is :";
 $today = date("d/m/Y");
 echo $today;
?>
```

Output:

```
Today's date is :05/12/2017
```

Formatting options available in date() function:

The format parameter of the date() function is a string that can contain multiple characters allowing to generate the dates in various formats. Date-related formatting characters that are commonly used in the format string:

- d: Represents day of the month; two digits with leading zeros (01 or 31).
- D: Represents day of the week in the text as an abbreviation (Mon to Sun).
- m: Represents month in numbers with leading zeros (01 or 12).
- M: Represents month in text, abbreviated (Jan to Dec).
- y: Represents year in two digits (08 or 14).
- Y: Represents year in four digits (2008 or 2014).

The parts of the date can be separated by inserting other characters, like hyphens (-), dots (.), slashes (/), or spaces to add additional visual formatting.

Example: The below example explains the usage of the date() function in PHP.

```
<?php
 echo "Today's date in various formats:" . "\n";
 echo date("d/m/Y") . "\n";
 echo date("d-m-Y") . "\n";
 echo date("d.m.Y") . "\n";
 echo date("d.M.Y/D");
?>
```

Output:

```
Today's date in various formats:
05/12/2017
05-12-2017
05.12.2017
05.Dec.2017/Tue
```

The following characters can be used along with the date() function to format the time string:

- h: Represents hour in 12-hour format with leading zeros (01 to 12).
- H: Represents hour in 24-hour format with leading zeros (00 to 23).
- i: Represents minutes with leading zeros (00 to 59).
- s: Represents seconds with leading zeros (00 to 59).
- a: Represents lowercase antemeridian and post meridian (am or pm).
- A: Represents uppercase antemeridian and post meridian (AM or PM).

Example: The below example explains the usage of the date() function in PHP.

```
<?php
 echo date("h:i:s") . "\n";
 echo date("M,d,Y h:i:s A") . "\n";
 echo date("h:i a");
?>
```

Output:

```
03:04:17
Dec,05,2017 03:04:17 PM
03:04 pm
```