Unit 3 Working with PHP Arrays and Functions

Arrays

- An array is a **single variable** that can **hold more than one value at once.**
- You can think of an array as a list of values.
- Each value within an array is called an element, and each element is referenced by its **own index**, which is unique to that array.
- To access an elements value — whether you are creating, reading, writing, or deleting the element you use that elements index.

Types of Array

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

1) Indexed arrays (Numeric arrays)

- In numeric array each element having numeric key associated with it that is starting from 0.

Creating Arrays

- You can use **array()** function to create array.
- **Syntax:** $array_name=array (value1, value2 …valueN);
- 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");
```

- The index can be assigned manually:

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

Functions of array:

Length of an Array

- 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);
?>
```

Output:

3

Accessing Array Elements

- The **print_r()** function is used to print information about a variable.

Example:

```
<?php
    $myarray=array("A","B","C");
    print_r($myarray);
?>
```

Output:

Array ([0] => A [1] => B [2] => C)

- You can refer to individual element of an array in PHP script using its key value as shown below:

Example:

```
<?php
    $myarray=array("A","B","C");
    echo $myarray[1];
?>
```

Output:

B

- In Numeric Array you can use for, while or do while loop to iterate through each element in array because in numeric array key values are consecutive.

Changing Elements

- You can also change value of element using index.

Example:

```
<?php
    $myarray=array("Apache", "MySQL", "PHP");
    $myarray[1]="Oracle";
    for($i=0; $i<3; $i++)
    {
        echo $myarray[$i]."<br>";
    }
?>
```

Output:

Apache
Oracle
PHP

Add Elements to Array

- The **array_push()** function inserts one or more elements to the end of an array.
- **Syntax:** array_push(array,value1,value2...)
  - **array**: Required. Specifies an array.
  - **value1**: Required. Specifies the value to add.
  - **value2**: Optional. Specifies the value to add.

Example:

```
<?php
    $myarray=array("Apache","MySQL", "PHP");
    print_r($myarray);

    echo "<br>";
     $myarray[]="Oracle";
    print_r($myarray);

    echo "<br>";
    array_push($myarray,"Java",".Net");
    print_r($myarray);
?>
```

Output:

Array ([0] => Apache [1] => MySQL [2] =>PHP)
Array ([0] => Apache [1] => MySQL [2] => PHP [3] => Oracle)
Array ([0] => Apache [1] => MySQL [2] => PHP [3] => Oracle [4] => Java [5] => .Net)

Removing Elements from Arrays

- **Unset ()** is used to destroy a variable in PHP. It can be used to remove a single variable, multiple variables, or an element from an array.
- **Syntax:** unset (var1, var2....)
  - var1, var2: The variable to be unset.
- You can remove the last element of an array using **array_pop()**. This will also return that element:

Example:

```
<?php
    $a=array("red","green","blue");
    array_pop($a);
    print_r($a);
?>
```

Output:

Array ([0] => red [1] => green)

Searching Element

- The **array_search()** function search an array for a value and returns the key.
- **Syntax:** array_search(value,array,strict)
  - **value:** Required. Specifies the value to search for.
  - **array**: Required. Specifies the array to search in.
  - **strict**: Optional. If this parameter is set to TRUE, then this function will search for identical elements in the array.

Example:

```
<?php
        $a=array("A"=>"5","B"=>5);
        echo array_search(5,$a);
        echo array_search(5,$a,true);
?>
```

Output:

A

B

- The **in_array()** function searches an array for a specific value.
- **Syntax:** in_array(search,array,type)
  - **search:** Required. Specifies the what to search for
  - **array:** Required. Specifies the array to search
  - **type:** Optional. If this parameter is set to TRUE, the in_array() function searches for the specific type in the array.

Example:

```
<?php
        $people=array("Peter", "Joe", "Glenn", 23);
        if (in_array("23",$people))
        {
            echo "Match found<br>";
        }
        else
        {
            echo "Match not found<br>";
        }
        if (in_array("23", $people, TRUE))
        {
            echo "Match found<br>";
        }
        else
        {
            echo "Match not found<br>";
        }
?>
```

Output:

Match found

Match not found

Sorting 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

Example:

```
<?php
        $srtArray=array(2,8,9,5,6,3);
        for ($i=0; $i<count($srtArray); $i++)
        {
            for ($j=0; $j<count($srtArray); $j++)
            {
                if ($srtArray[$j] > $srtArray[$i])
                {
                        $tmp = $srtArray[$i];
                        $srtArray[$i] = $srtArray[$j];
                        $srtArray[$j] = $tmp;
                }
            }
        }
        foreach($srtArray as $item)
        {
           echo $item."<br>\n";
        }
?>
```

Output:

2 3 5 6 8 9

- Sort array using function.

Example:

```
<?php
        $srtArray=array(2,8,9,5,6,3);
        sort($srtArray);
        foreach($srtArray as $item) {
            echo $item."<br>\n";
        }
?>
```

Output:

2 3 5 6 8 9

2) Associative Array

- The associative part means that arrays store element values in association with key values rather than in a strict linear index order.
- If you store an element in an array, in association with a key, all you need to retrieve it later from that array is the key value.
- **Key** may be either **numeric or string**.
- You can use array() function to create associative array.
- **Syntax:** $array_name=array(key1=>value1, key2=>value2,….. keyN => valueN);
- 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";
```

- You can refer to individual element of an array in PHP using its key value.

Example:

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

Output:

Ben:37

- In associative array you **cannot use for, while or do…while loop** to iterate through each element in array because in Associative array key value are not consecutive.
- So you have to use foreach loop.

Example:

```
<?php
    $myarray=array("Name"=>"James", "Age"=>25, "Gender"=>"Male");
    foreach($myarray as $item) {
        echo $item."<br>";
    }
?>
```

Output:

James
25
Male

3) Multidimensional Array

- Earlier, we have described arrays that are a single list of key/value pairs.
- 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.
- First, take a look at the following table:

Volvo	22	18
BMW	15	13
Saab	5	2

```
<?php
$cars = array (
        array("Volvo",22,18),
        array("BMW",15,13),
        array("Saab",5,2),
 );
for ($row = 0; $row <  3; $row++)
{
        echo "<br>";
        for ($col = 0; $col <  3; $col++)
        {
                echo $cars[$row][$col]." ";
        }
}
?>
```

Output:

Volvo 22 18
BMW 15 13
Saab 5 2

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

- A user defined function declaration starts with the word "function".

- **Syntax:**

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

Example:

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

Output:

Hello world!

Function arguments and returning values from function

- 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 separate with comma.

Example:

```
<?php
function addFunction($num1, $num2)
{
    $sum = $num1 + $num2;
    return $sum;
}
$result=addFunction(10, 20);
echo "Addition=".$result;
?>
```

Output:

Addition=30

Default values for arguments

- You can specify default values for arguments.
- If the argument is omitted from the function call the default is used.
- The default value must be a constant expression such as a string, integer or NULL.
- You can have multiple arguments with default values. Default values assign from right to left.

Example:

```
<?php
    function addFunction($num1, $num2=5)
```

```
    {
        $sum = $num1 + $num2;
        return $sum;
    }
    $result=addFunction(10,20);
    echo "Addition=".$result."<br>";
    $result=addFunction(10);
    echo "Addition=".$result;
?>
```

Output:

Addition=30
Addition=15

Call-by-Value (Pass-by-Value)

- The default behavior for user-defined functions in PHP is call-by-value.
- Call 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.

Example:

```
<?php
    function swap($num1, $num2)
    {
        $temp = $num1;
        $num1=$num2;
        $num2=$temp;
    }
    $num1=10;
    $num2=20;
    echo "Before Swap"."<br>";
    echo "num1=".$num1."<br>";
    echo "num2=".$num2."<br>";
    swap($num1,$num2);
    echo "After Swap"."<br>";
    echo "num1=".$num1."<br>";
    echo "num2=".$num2."<br>";
?>
```

Output:

Before Swap
num1=10
num2=20

```
After Swap
num1=10
num2=20
```

Call-by-Reference (Pass-by-Reference)

- Call by reference means passing the address of a variable where the actual value is stored.
- The called function uses the value stored in the passed address; **any changes to it do affect** the source variable.

Example:

```
<?php
    function swap(&$num1, &$num2)
    {
        $temp = $num1;
        $num1=$num2;
        $num2=$temp;
    }
    $num1=10;
    $num2=20;
    echo "Before Swap"."<br>";
    echo "num1=".$num1."<br>";
    echo "num2=".$num2."<br>";
    swap($num1,$num2);
    echo "After Swap"."<br>";
    echo "num1=".$num1."<br>";
    echo "num2=".$num2."<br>";
?>
```

Output:

```
Before Swap
num1=10
num2=20
After Swap
num1=20
num2=10
```

Variables Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.
- PHP has three different variable scopes:
  - local
  - global

- static

Local

- A variable declared **within** a function has a local scope and can only be accessed within that function.

```
<?php
    function myTest()
    {
        $x = 5; // local scope
        echo "Variable x inside function is: $x";
    }
    myTest();
    //echo "Variable x outside function is: $x";  // using x outside the function will generate an error
?>
```

Global

- A variable declared **outside** a function has a global scope and can only be accessed outside a function:

```
<?php
    $x = 5; // global scope
    function myTest()
    {
        echo "Variable x inside function is: $x"; // using x inside this function will generate an error
    }
    //myTest();
    echo "Variable x outside function is: $x";
?>
```

- The global keyword is used to access a global variable from within a function.
- To do this, use the global keyword before the variables (inside the function):

```
global $x;
```

Static

- Normally, when a function is completed/executed, all of its variables are deleted.
- However, sometimes we want a local variable not to be deleted. We need it for a further job.
- To do this, use the **static** keyword when you first declare the variable:

```
<?php
    function myTest()
    {
        static $x = 0;
        echo $x;
        $x++;
    }
    myTest();
```

```
    echo "<br>";
    myTest();
    echo "<br>";
    myTest();
?>
```

Output:

0

1

2

- Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

String Function

chr

- Returns a character from a specified ASCII value.
- **Syntax: chr(ascii)**

Example:

```
<?php
    echo chr(65);
?>
```

Output:

A

ord

- Returns the ASCII value of the first character of a string.
- **Syntax: ord(string)**

Example:

```
<?php
    echo ord("A")."<br />";
    echo ord("And");
?>
```

Output:

65

65

strtolower

- Converts a string to lowercase letters.
- **Syntax: strtolower(string)**

Example:

```
<?php
    echo strtolower("HELLO");
?>
```

Output:

hello

strtoupper

- Converts a string to uppercase letters.
- **Syntax: strtoupper(string)**

Example:

```
<?php
    echo strtoupper ("hello");
?>
```

Output:

HELLO

strlen

- Returns the length of a string.
- **Syntax: strlen(string)**

Example:

```
<?php
    echo strlen("hello hi");
?>
```

Output:

8

ltrim

- Remove whitespace from the left side of a string.
- **Syntax: ltrim(string,charlist)**
  - **charlist**:- Optional. Specifies which characters to remove from the string.
  - If omitted, all of the following characters are removed:"\0" – NULL, "\t" – tab, "\n" – new line, "\r" – carriage return, " " – white space.

Example:

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

Output:

ello

Hello

rtrim

- Remove whitespace from the right side of a string.
- **Syntax: rtrim(string,charlist)**
  - **charlist**:- Optional. Specifies which characters to remove from the string.
  - If omitted, all of the following characters are removed:"\0" – NULL, "\t" – tab, "\n" – new line, "\r" – carriage return, " " – white space.

Example:

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

    $str="Hello   ";
    echo rtrim($str);
?>
```

Output:

Hell

Hello

trim

- Remove whitespace from both sides of a string.
- **Syntax: trim(string,charlist)**
  - **charlist**:- Optional. Specifies which characters to remove from the string.
  - If omitted, all of the following characters are removed:"\0" – NULL, "\t" – tab, "\n" – new line, "\r" – carriage return, " " – white space.

Example:

```
<?php
    $str="oHello";
    echo trim($str,"o")."<br>";

    $str="   Hello   ";
    echo trim($str);
?>
```

Output:

Hell

Hello

substr

- Returns a part of a string.
- **Syntax: substr(string,start,length)**
  - **start:** Required. Specifies where to start in the string.
    - A 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.

Example:

```
<?php
    echo substr("abcdef",-2);
    echo substr("abcdef",-3,1);
    echo substr("abcdef",0,-1);
    echo substr("abcdef",2,-1);
    echo substr("abcdef",4,-4);
?>
```

Output:

```
ef
d
abcde
cde
```

strcmp

- Compares two strings (case-sensitive).
- strcmp function returns: 0 - if the two strings are equal, negative - if string1 is less than string2 and positive - if string1 is greater than string2.
- **Syntax: strcmp(string1,string2)**

Example:

```
<?php
    echo strcmp("hello","hello");
    echo strcmp("hello","hi");
    echo strcmp("hi","hello");
?>
```

Output:

```
0
```

-1

1

strcasecmp

- Compares two strings (case-insensitive)
- strcasecmp function returns: 0 - if the two strings are equal, negative - if string1 is less than string2 and positive - if string1 is greater than string2.
- **Syntax: strcasecmp(string1,string2)**

```
<?php
        echo strcasecmp("hello","Hello");
?>
```

Output:

0

strpos

- Returns the position of the **first** occurrence of a string inside another string (case-sensitive).
- If the string is not found, this function returns FALSE.
- **Syntax:- strpos(string,find,start)**
  - **find:-** Required. Specifies the string to find
  - **start:-** Optional. Specifies where to begin the search

Example:

```
<?php
    echo strpos("hello","e");
    echo strpos("hello","l");
?>
```

Output:

1

2

strrpos

- Returns the position of the **last** occurrence of a string inside another string (case-sensitive).
- If the string is not found, this function returns FALSE.
- **Syntax:- strrpos(string,find,start)**
  - **find:-** Required. Specifies the string to find
  - **start:-** Optional. Specifies where to begin the search

Example:

```
<?php
    echo strrpos("hello","l");
?>
```

Output:

3

strstr

- Finds the first occurrence of a string inside another string (case-sensitive).
- This function returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found.
- **Syntax: strstr(string,search)**
  - **search:** Required. Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number.

Example:

```
<?php
    echo strstr("Hello world!","wor");
    echo strstr("Hello world!","Wor");
?>
```

Output:

world!

stristr

- Finds the first occurrence of a string inside another string (case-insensitive).
- This function returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found. x
- **Syntax: stristr(string,search)**

Example:

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

Output:

world!

str_replace

- Replaces some characters in a string (case-sensitive).
- **Syntax:- str_replace(find,replace,string,count)**
  - **find:** Required. Specifies the value to find
  - **replace:** Required. Specifies the value to replace the value in find
  - **string:** Required. Specifies the string to be searched
  - **count:** Optional. A variable that counts the number of replacements

Example:

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

Output:

Hello Peter!

strrev

- Reverses a string.
- **Syntax: strrev(string)**

Example:

```
<?php
    echo strrev("hello");
?>
```

Output:

olleh

Math Function

abs

- Returns the absolute value of a number.
- **Syntax: abs(x)**

Example:

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

Output:

6.7

3

ceil

- Returns the smallest integer value that is greater than or equal to x.
- **Syntax: ceil(x)**

Example:

```
<?php
    echo ceil(0.60)."<br>";
    echo ceil(5)."<br>";
    echo ceil(-4.9);
 ?>
```

Output:

1

5

-4

floor

- Returns the largest integer value that is less than or equal to x.
- **Syntax: floor(x)**

  Example:

```
<?php
    echo floor(0.60)."<br>";
    echo floor(5)."<br>";
    echo floor (4.9);
?>
```

  Output:

```
0
5
4
```

round

- Rounds a number to the nearest integer.
- **Syntax: round(x,precision)**
  - **precision**: Optional. The number of digits after the decimal point

  Example:

```
<?php
    echo round(5.335,2)."<br>";
    echo round(0.49);
?>
```

  Output:

```
5.34
0
```

fmod

- Returns the remainder (modulo) of the division of the arguments.
- **Syntax: fmod(x,y)**

  Example:

```
<?php
    echo fmod(5,2);
?>
```

  Output:

```
1
```

min

- Returns the number with the lowest value from specified numbers.

- **Syntax: min(x1,x2,x3…)**

  Example:

```
<?php
    echo min(5,2,1,-4);
?>
```

  Output:

  -4

max

- Returns the number with the highest value from specified numbers.
- **Syntax: max(x1,x2,x3…)**

  Example:

```
<?php
    echo max(5,2,1,-4);
?>
```

  Output:

  5

pow

- Returns the value of x to the power of y.
- **Syntax: pow(x,y)**

  Example:

```
<?php
    echo pow(5,2);
?>
```

  Output:

  25

sqrt

- Returns the square root of a number x.
- **Syntax: sqrt(x)**

  Example:

```
<?php
    echo sqrt(25);
?>
```

  Output:

  5

rand

- Returns a random integer.

- **Syntax: rand(min,max)**
  - **min,max:** Optional. Specifies the range the random number should lie within
  - If this function is called without parameters, it returns a random integer between 0 and RAND_MAX. (RAND_MAX depends on platform. For Windows value is 32768).

Example:

```
<?php
    echo rand()."<br>";
    echo rand(40,500);
?>
```

Output:

18269

417

Date Function

date

- Formats a local time/date.
- **Syntax: date(format,timestamp)**
  - **format:** Required. Specifie the format of date and time to be returned.
- **d** - The day of the month (from 01 to 31)
- **m** - A numeric representation of a month (from 01 to 12)
- **M** - A short textual representation of a month (three letters)
- **Y** - A four digit representation of a year
- **y** - A two digit representation of a year

Example:

```
<?php
    echo date("d/m/y")."<br>";
    echo date("d M, Y");
?>
```

Output:

11/08/15

11 Aug, 2015

getdate

- Returns an array that contains date and time information for a UNIX timestamp.
- The returning array contains ten elements with relevant information needed when formatting a date string: [seconds] – seconds, [minutes] – minutes, [hours] – hours, [mday] - day of the month, [wday] - day of the week, [year] – year, [yday] - day of the year, [weekday] - name of the weekday, [month] - name of the month.

- **Syntax: getdate(timestamp)**
  - Timestamp:- Optional. Specifies the time in Unix time format

Example:

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

Output:

Array ([seconds] => 3 [minutes] => 49 [hours] =>17 [mday] => 11 [wday] => 2 [mon] => 8 [year] => 2015 [yday] => 222 [weekday] => Tuesday [month] => August [0] => 1439308143)

checkdate

- The checkdate() function returns true if the specified date is valid, and false otherwise.
- **Syntax: checkdate(month,day,year)**
  - **month, day, year:** Required. Specifies the month, day and year.
- A date is valid if: month is between 1 to 12 , day is within the allowed number of days for the particular month, year is between 1 to 32767.

Example:

```
<?php
    var_dump(checkdate(2,29,2003));
    var_dump(checkdate(2,29,2004));
?>
```

Output:

boolean false
boolean true

time

- The time() function returns the current time as a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).
- **Syntax: time(void)**

Example:

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

Output:

1439309922

mktime

- The mktime() function returns the Unix timestamp for a date.
- This 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,is_dst)**
  - **hour, minute, second, month, day, year:** Optional. Specifies the hour, minute, second, month, day, year.
  - **is_dst:** Optional. Set this parameter to 1 if the time is during daylight savings time (DST), 0 if it is not, or -1 (the default) if it is unknown.

Example:

```
<?php
    echo(date("M-d-Y",mktime(0,0,0,12,36,2001))."<br />");
    echo(date("M-d-Y",mktime(0,0,0,1,1,99))."<br />");
?>
```

Output:

Jan-05-2002

Jan-01-1999

File Function

fwrite()

- The fwrite() writes to an open file.
- The function will stop at the end of the file or when it reaches the specified length, whichever comes first.
- This function returns the number of bytes written, or FALSE on failure.
- This function is binary-safe.
- **Syntax: fwrite(file,string,length)**
  - **file:** Required. Specifies the open file to write to
  - **string:** Required. Specifies the string to write to the open file
  - **length:** Optional. Specifies the maximum number of bytes to write

Example:

```
<?php
    $file = fopen("test.txt","w");
    echo fwrite($file,"Hello World. Testing!");
    fclose($file);
?>
```

Output:

21

fopen()

- The fopen() function opens a file or URL.
- If fopen() fails, it returns FALSE and an error on failure.
- **Syntax: fopen(filename,mode,include_path,context)**
- The file may be opened in one of the following modes: r,w,a,x,r+,w+,a+,x+

Example:

```
<?php
    $file = fopen("test.txt","r");
    $file = fopen("/home/test/test.txt","r");
    $file = fopen("http://www.example.com/","r");
?>
```

fclose()

- The fclose() function is used to close an open file.
- The fclose() requires the name of the file (or a variable that holds the filename).
- **Syntax: fclose(file)**

Example:

```
<?php
    $myfile = fopen("webdictionary.txt", "r");
    // some code to be executed....
    fclose($myfile);
?>
```