[Arrays & Functions]

PHP & MySql

Neetin Sharma
(Email: info@neetin.com.np)
For more information
Please visit: neetin.com.np/web

Arrays

Arrays are indexed, which means that each entry is made up of a key and a value. The key is the index position, beginning with 0. The value is whatever value we associate with that position—a string, an integer, or whatever we want.

We can create an array using either the `array()` function or the array operator `[ ]`. The `array()` function is usually used when we want to create a new array and populate it with more than one element at the same time.

Example:

```
$rainbow = array("red", "orange", "yellow", "green", "blue", "indigo", "violet");
```

//here $rainbow is an array having 7 items.

We can also create array like this

```
$rainbow[0] = "red";
$rainbow[1] = "orange";
$rainbow[2] = "yellow";
$rainbow[3] = "green";
$rainbow[4] = "blue";
$rainbow[5] = "indigo";
$rainbow[6] = "violet";
```

index [ ] value is not mandatory , we can simply write like this

```
$rainbow[] = "red";
$rainbow[] = "orange";
$rainbow[] = "yellow";
$rainbow[] = "green";
$rainbow[] = "blue";
$rainbow[] = "indigo";
$rainbow[] = "violet";
```

Some Array-Related Functions:

`count()` and `sizeof()`— Each counts the number of elements in an array. Given the following array:

```
$colors = array("blue", "black", "red", "green");
```

both count($colors); and sizeof($colors); return a value of 4.

`shuffle()`— This randomizes the elements of a given array. The syntax of this function is simply as follows:

```
shuffle($colors);
```

Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
     <title>Php arrays</title>
</head>
<body>
     <?php
          $colors = array("blue", "black", "red", "green");
          echo $colors[2];
          shuffle($colors);
          echo "<br />". $colors[2];
     ?>
</body>
</html>
```

Working with Functions

Functions are at the heart of a well-organized script, and will make our code easy to read and reuse. No large project would be manageable without them. Throughout this chapter, we will investigate functions and demonstrate some of the ways in which they can save us from repetitive work.

What is a Function?

A function is a self-contained block of code that can be called by our scripts. When called, the function's code is executed. We can pass values to a function, which will then use them appropriately. When finished, a function can pass a value back to the calling code.

Calling Built in Functions:

```
print ("Hello Web!");
```

Example:

```
<?php
          $num = -321;
          $newnum = abs( $num );
          echo $newnum;
          //prints "321"
?>
```

Defining a Function:

We can define our own functions using the `function` statement:

Syntax:

```
function some_function($argument1, $argument2)
{
    //function code here
}
```

Example:

```
<?php
        function hello()
        {
                echo "HELLO! ";
        }
        hello();
?>
```

Passing arguments to a function:

Example:

```
<?php
        function hello($name)
        {
                echo "HELLO $name !";
        }
                hello("Ram Prasad");
?>
```

Returning Values from User-Defined Functions

We can return single value using ***return*** keyword.
Example:

```
<?php
        function addNums($firstnum, $secondnum)
        {
                $result = $firstnum + $secondnum;
                return $result;
        }
        echo addNums(3,5);    //will print "8"
?>
```

Variable Scope

A variable declared within a function remains local to that function. In other words, it will not be available outside the function or within other functions. In larger projects, this can save us from accidentally overwriting the contents of a variable when we declare two variables with the same name in separate functions.

Example:

```
<?php
        function test()
        {
                $testvariable = "this is a test variable";
        }
        echo "test variable: $testvariable<br>";
?>
```

Output text in a browser

test variable:

Passing argument by reference

When reference of a variable is passed to a function then any modification to a variable is retained outside of the function as well. We receive arguments by prefixing with & symbol in a function definition.

Example:

```
<?php
        function one(&$a, &$b)
        {
                echo "Value of A and B in function.<br />";
                echo "A= $a <br />B= $b";
                $a=$b*2;
        }
$c=2;
$d=5;

echo "value of C and D before function call: $c and $d respectively.<br />";
one($c, $d);
echo "<br />value of C and D after function call: $c and $d respectively.";
?>
```

Output in browser

value of C and D before function call: 2 and 5 respectively.
Value of A and B in function.
A= 2
B= 5
value of C and D after function call: 10 and 5 respectively.

Accessing global variable within function with a global ***statement***

```
<?php
        $life=42;
        function meaningOfLife()
        {
                global $life;
                echo "The meaning of life is $life<br>";
```

```
        }
        meaningOfLife();
?>
```

Recursive function

A function which call itself is called a recursive function:

Example:

```
<?php

/* finding a factorial of an integer , we use recursive
function */

function fact($n)
{
      if($n <=1){        //less than or equal to 1 is exit
condition
            return 1;
      }
      else
      {
            return ($n * fact($n-1));
      }
}
echo fact(5);
?>
```

Php Built In Function

String manipulating functions

Finding the Length of a String with strlen()

We can use `strlen()` to determine the length of a string. `strlen()` requires a string and returns an integer representing the number of characters in the variable we have passed it. `strlen()` might typically be used to check the length of user input. The following fragment tests a membership code to ensure that it is four digits long:

```
if (strlen($membership) == 4) {
    echo "<p>Thank you!</p>";
} else {
    echo "<p>Your membership number must have 4 digits</p>";
}
```

Finding a Substring Within a String with strstr()

We can use `strstr()` to test whether a string exists embedded within another string. `strstr()` requires two arguments: a source string and the substring we want to find within it.

The function returns `false` if the substring is absent. Otherwise, it returns the portion of the source string beginning with the substring. For the following example, imagine that we want to treat membership codes that contain the string `AB` differently from those that do not:

```
$membership = "pAB7";
if (strstr($membership, "AB")) {
  echo "<p>Thank you. Don't forget that your membership expires soon!</p>";
} else {
  echo "<p>Thank you!</p>";
}
```

Finding the Position of a Substring with strpos()

The `strpos()` function tells us both whether a string exists within a larger string and where it is to be found. `strpos()` requires two arguments: the source string and the substring we are seeking. The function also accepts an optional third argument, an integer representing the index from which we want to start searching. If the substring does not exist, `strpos()` returns `false`; otherwise, it returns the index at which the substring begins. The following fragment uses `strpos()` to ensure that a string begins with the string `mz`:

```
$membership = "mz00xyz";
if (strpos($membership, "mz") === 0) {
  echo "hello mz";
}
```

Extracting Part of a String with substr()

The `substr()` function returns a portion of a string based on the start index and length of the portion we are looking for. This function demands two arguments: a source string and the starting index. It returns all characters from the starting index to the end of the string you are searching. It optionally accepts a third argument, which should be an integer representing the length of the string we want returned. If this argument is present, `substr()` returns only the number of characters specified from the start index onwards.

```
$test = "scallywag";
echo substr($test,6);  // prints "wag"
echo substr($test,6,2) // prints "wa"
```

If we pass `substr()` a negative number as its second (starting index) argument, it will count from the end rather than the beginning of the string. The following fragment writes a specific message to people who have submitted an email address ending in `.uk`:

```
$test = "matt@corrosive.co.uk";
if ($test = substr($test, -3) == ".uk") {
        echo "<p>Don't forget our special offers for British customers!</p>";
} else {
  echo "<p>Welcome to our shop!</p>";
```

```
}
```

Changing case

The following PHP functions return a copy of the *subject* string with changes in the case of the characters :

string strtolower(string subject)

string strtoupper(string subject)

string ucfirst(string subject)

string ucwords(string subject)

The following fragment shows how each operates:

```
print strtolower("PHP and MySQL"); // php and mysql
print strtoupper("PHP and MySQL"); // PHP AND MYSQL
print ucfirst("now is the time");  // Now is the time
print ucwords("now is the time");  // Now Is The Time
```

Trimming whitespace

PHP provides three functions that trim leading or trailing whitespace characters from strings:

string ltrim(string subject [, string character_list])

string rtrim(string subject [, string character_list])

string trim(string subject [, string character_list])

The three functions return a copy of the subject string: *trim( )* removes both leading and trailing whitespace characters, *ltrim( )* removes leading whitespace characters, and *rtrim( )* removes trailing whitespace characters. The following example shows the effect of each:

```
$var = trim(" Tiger Land \n");   // "Tiger Land"
$var = ltrim(" Tiger Land \n");  // "Tiger Land \n"
$var = rtrim(" Tiger Land \n");  // " Tiger Land"
```

By default these functions trim space, tab (`\t`), newline (`\n`), carriage return (`\r`), NULL (`\x00`), and the vertical tab (`\x0b`) characters. The optional *character_list* parameter allows you to specify the characters to trim. A range of characters can be specified using two periods (..) as shown in the following example:

```
$var = trim("16 MAY 2004", "0..9 ");  // Trims digits and spaces

print $var;                    // prints "MAY"
```

Comparing Strings

PHP provides the string comparison functions *strcmp( )* and *strncmp( )* that compare two strings in alphabetical order, str1 and str2:

integer strcmp(string str1, string str2)

integer strncmp(string str1, string str2, integer length)

While the equality operator `==` can compare two strings, the result isn't always as expected for strings with binary content or multi-byte encoding: *strcmp( )* and *strncmp( )* provide binary safe string comparison. Both *strcmp( )* and *strncmp( )* take two strings as parameters, *str1* and *str2*, and return 0 if the strings are identical, 1 if *str1* is less than *str2*, and -1 if *str1* is greater that *str2*. The function *strncmp( )* takes a third argument *length* that restricts the comparison to *length* characters. String comparisons are often used as a conditional expression in an `if` statement like this:

```
$a = "aardvark";
$z = "zebra";

// Test if $a and $z are not different (i.e. the same)
if (!strcmp($a, $z))
   print "a and z are the same";
```

When *strcmp( )* compares two different strings, the function returns either -1 or 1 which is treated as `true` in a conditional expression. These examples show the results of various comparisons:

```
print strcmp("aardvark", "zebra");       // -1
print strcmp("zebra", "aardvark");       //  1
print strcmp("mouse", "mouse");          //  0
print strcmp("mouse", "Mouse");          //  1
print strncmp("aardvark", "aardwolf", 4); //  0
print strncmp("aardvark", "aardwolf", 5); // -1
```

The functions *strcasecmp( )* and *strncasecmp( )* are case-insensitive versions of *strcmp( )* and *strncmp( )*. For example:

```
print strcasecmp("mouse", "Mouse");      //  0
```

Php Date And Time Functions

Current time

PHP provides several functions that generate a Unix timestamp. The simplest:

integer time( )

returns the timestamp for the current date and time, as shown in this fragment:

```
// prints the current timestamp: e.g., 1064699133
print time( );
```

Creating timestamps with mktime( ) and gmmktime( )

To create a timestamp for a past or future date in the range December 13, 1901 through January 19, 2038, the *mktime( )* and *gmmktime( )* functions are defined:

int mktime(int hour, int minute, int second, int month, int day, int year [, int is_dst])

int gmmktime(int hour, int minute, int second, int month, int day, int year [, int is_dst])

Both create a timestamp from the parameters supplied; the parameters supplied to *gmmktime( )* represent a GMT date and time, while the parameters supplied to *mktime( )* represent the local time. This example creates a timestamp for 9:30 A.M. on June 18, 1998:

```
$aDate = mktime(9, 30, 0, 6, 18, 1998);
```

Both functions correctly handle parameter values that you might consider out-of-range. For example, the following call passes 14 for the month value, 29 for the day, and 2004 for the year, creating a time stamp for 1 March 2005:

```
// Creates a time stamp for 1 March 2005
$aDate = mktime(0, 0, 0, 14, 29, 2004);
```

String to timestamp

The *strtotime( )* function generates a timestamp by parsing a human-readable date and time (between December 13, 1901 and January 19, 2038) from the string time:

integer strtotime(string time)

The function interprets several standard representations of a date, as shown here:

```
// Absolute dates and times
$var = strtotime("25 December 2002");
$var = strtotime("14/5/1955");
$var = strtotime("Fri, 7 Sep 2001 10:28:07 -1000");
// The current time: equivalent to time( )
$var = strtotime("now");
// Relative to now
print strtotime("+1 day"); // tomorrow
print strtotime("-2 weeks"); // two weeks ago
print strtotime("+2 hours 2 seconds"); // in two hours and two seconds
```

Care should be taken when using *strtotime( )* with user-supplied dates. It's better to limit the use of *strtotime( )* to cases when the string to be parsed is under the control of the script. For example, it's used here to check a minimum age using a relative date:

```
// date of birth: timestamp for 16 August, 1982
```

```
$dob = mktime(0, 0, 0, 8, 16, 1982);
// Now check that the individual is over 18
if ($dob < strtotime("-18 years"))
    print "Legal to drive in the state of Victoria"; // prints
```

Formatting a Date

While the Unix timestamp is useful in a program, it isn't a convenient display format. The *date( )* and *gmdate( )* functions return a human-readable formatted date and time:

string date(string format [, integer timestamp])

string gmdate(string format [, integer timestamp])

The format of the returned string is determined by the format argument. Passing in the optional timestamp argument can format a predetermined date. Otherwise, both functions format the current time. The format string uses the formatting characters listed in Table below to display various components or characteristics of the timestamp. To include characters without having the function interpret them as formatting characters, escape them with a preceding backslash character. The following examples show various combinations:

```
// Set up a timestamp for 08:15am 24 Aug 1974
$var = mktime(8, 15, 25, 8, 24, 1974);
// "24/08/1974"
print date('d/m/Y', $var);
// "08/24/74"
print date('m/d/y', $var);
// prints "Born on Saturday 24th of August"
print date('\B\o\r\n \o\n l jS \of F', $var);
```

Formatting characters used by the date( ) function

Formatting character	Meaning
a, A	"am" or "pm"; "AM" or "PM"
S	Two-character English ordinal suffix: "st", "nd", "rd", "th"
d, j	Day of the month: with leading zeros e.g., "01"; without e.g., "1"
D, l	Day of the week: abbreviated e.g., "Mon"; in full e.g., "Monday"
M, F	Month: abbreviated e.g., "Jan"; in full e.g., "January"
m, n	Month as decimal: with leading zeros: "01"-"12"; without: "1"-"12"
h, g	Hour 12-hour format: with leading zeros e.g., "09"; without e.g., "9"
H, G	Hour 24-hour format: with leading zeros e.g., "01"; without e.g., "1"
I	Minutes:"00" to "59"

Formatting characters used by the date( ) function	
Formatting character	**Meaning**
s	Seconds: "00" to "59"
Y, y	Year: with century e.g., "2004"; without century e.g., "04"
r	RFC-2822 formatted date: e.g., "Tue, 29 Jan 2002 09:15:33 +1000" (added in PHP 4.0.4)
w	Day of the week as number: "0" (Sunday) to "6" (Saturday)
t	Days in the month: "28" to "31"
z	Days in the year: "0" to "365"
B	Swatch Internet time
L	Leap year: "0" for normal year; "1" for leap year
I	Daylight savings time: "0" for standard time; "1" for daylight savings
O	Difference to Greenwich Mean Time in hours: "+0200"
T	Time zone setting of this machine
Z	Time zone offset in seconds: "-43200" to "43200"
U	Seconds since the epoch: 00:00:00 1/1/1970

Validating a Date

The function *checkdate( )* returns `true` if a given month, day, and year form a valid Gregorian date:

boolean checkdate(integer month, integer day, integer year)

This function isn't based on a timestamp and so can accept any dates in the years 1 to 32767. It automatically accounts for leap years.

```
// Works for a wide range of dates
$valid = checkdate(1, 1, 1066); // true
$valid = checkdate(1, 1, 2929); // true
// Correctly identify bad dates
$valid = checkdate(13, 1, 1996); // false
$valid = checkdate(4, 31, 2001); // false
// Correctly handles leap years
$valid = checkdate(2, 29, 1996); // true
$valid = checkdate(2, 29, 2001); // false
```

Some Mathematical Functions

Rounding up a number:

ceil() – it accepts one parameter, a number and returns the round to the nearest integer greater than the number.
Syntax:

```
float ceil(float <value>)
```

floor() – it accepts one parameter, a number and returns the round to the nearest integer smaller than the number.
Syntax:

```
float floor(float <value>)
```

example:

```
<?php
  echo ceil(5.1);      //prints 6
  echo floor(2.9);     //prints 2
?>
```

round() – it accepts two parameter, a number and returns the round to the nearest integer smaller than the number.
Syntax:

```
float round(float <calue>[, int <precision>] )
```

examples:

```
<?php
  echo round(1.6);    //prints 2
  echo round(1.4);    //prints 1
  echo round(8.4519, 2);        //prints 8.45
?>
```

Some other functions:

pow() – this functions take two argument (both integer) and returns single integer value.

sqrt() – this functions take one argument as number and returns square root of a number.

abs() - this functions take one argument as number and returns absolute value of a number.

Examples:

```
<?php
  echo "<br />".pow(2, 3);              //prints 8
  echo "<br />".sqrt(9);                //prints 3
  echo "<br />".abs(-69)                //prints 69
?>
```

Randomization:

rand() function is a basic randomization function that is very quick but not very random as the numbers it generates are slightly more predictable.

Syntax: int rand(int min, int max)
Example:

```
<?php
  echo "<br />". rand();                    //prints 14765
  echo "<br />". rand(1, 100); //prints 92
?>
```

Functions For Variables

There are three basic functions that are useful while using variables. These are

a) isset()
b) unset()
c) empty()

isset() and unset()
unset() is used to destroy a variable, freeing all the memory that is associated with the variable so it is then available again. The isset() function is used to determine whether a variable has been given a value. If a value has been set then it returns true.
Example:

```
<?php
        $ProductID = "432BB";
        if (isset($ProductID)) {
                echo "This will print";
        }
        unset($ProductID);
        if (isset ($ProductID)) {
                echo "This will NOT print";
        }
?>
```

empty()
empty() is nearly the opposite of isset(). It returns true if the variable is not set, or has a value of zero or an empty string. Otherwise it returns false.

```
<?php
        echo empty($new);       // true
        $new = 1;
        echo empty($new);       // false
        $new = "";
        echo empty($new);       // true
        $new = 0;
        echo empty($new);       // true
        $new = "Neetin Sharma";
        echo empty($new);       // false
        unset ($new);
        echo empty($new);       // true
?>
```