Functions In PHP

• Functions are at the heart of a well-organized script and will make your code easy to read and reuse.

• Large projects would be unmanageable without functions because the problem of repetitive code would bog down the development process.

• If you haven’t had much experience using functions, you can think of a function as an input/output machine. The machine takes the raw materials you feed it (the input) and works with them to produce a product (the output).

• A function accepts values, processes them, and then performs an action (printing to the browser, for example), returns a new value, or both.
Functions In PHP

• If you need to bake a cake, you would probably do it yourself, in your own kitchen with your oven. But if you need to bake thousands of cakes, you would probably build or acquire a special cake-baking machine, built for making cakes in massive quantities.

• Similarly, when deciding whether to create a function for reuse or simply writing in-line code, the most important factor to consider is the extent to which it can save you from writing repetitive code.

• If the code you are writing will be used more than once, it is probably best to create a function to represent the code.
Functions In PHP

• A function is a self-contained block of code that can be called by your script.

• When called (or invoked), the function’s code is executed and performs a particular task. You can pass values to a function (called arguments), which then uses the values appropriately – storing them, transforming them, displaying them, whatever the function is designed to do. When finished, a function can also pass a value back to the original code that called it into action.

• In PHP, functions come in two flavors – those built in to the language, and those that you define yourself.
Functions In PHP

• PHP has hundreds of built-in functions. Consider the example shown on the next page that utilizes the built-in function `strtoupper()`.

• The output from this script is shown below:

![Using a built in function in PHP](image-url)

The original string is: Hello World!
The string after function `strtoupper()` operated on the string is: HELLO WORLD!
<?php

$original_string = "Hello World!";
$capitalized_string = strtoupper($original_string);

echo "<br />
The original string is: ".$original_string."<br />";

echo 'The string after function strtoupper() operated on the string is: ".$capitalized_string."<br />";
Functions In PHP

• In the previous example, the function `strtoupper()` is called and passed a variable whose value is represented by a string. The function goes about its business of changing the contents of the string to uppercase letters.

• A function call consists of the function name followed by parentheses. (Note, even a function that has no parameters requires a set of parentheses.) The information being passed to the function (the arguments) are placed between the parentheses.

• For functions that require more than one argument, the arguments are separated by commas:

```php
some_function ($an_argument, $another_argument);
```
Functions In PHP

- The `strtoupper()` function from the previous example is typical for a function in that it returns a value. Most functions return some information back after they’ve completed their task – they usually at least tell whether their mission was successful.

- The `strtoupper()` function returns a string value so its usage requires the presence of a variable to accept the returned string, as was the case in the line:

  ```
  $capitalized_string = strtoupper($original_string);
  ```

- Functions in PHP that return values use a `return` statement within the body of the function. We’ll use this in a few more pages when we start constructing our own functions.
Defining Functions In PHP

• You can define your own functions in PHP using the function statement:

```php
function someFunction($argument1, .. .. ,argument2) {
    //function code goes here
}
```

• The name of the function follows the function statement and precedes a set of parentheses. If your function requires arguments, you must place the comma-separated variable names within the parentheses. These variables will be filled by the values passed to your function when it is called.

• Even if your function does not require arguments you must still supply the parentheses.
Defining Functions In PHP

• Naming conventions for functions are the same as for normal variables in PHP. As with variables you should apply meaningful names and be consistent in naming and style. Using mixed case in function names is a common convention, thus `myFunction()` instead of `myfunction()` or `my_function()`. (Note: variables names are case sensitive in PHP, function names are not!)

• Let’s define a simple function that simply prints out the word “Hello” in big letters.

```php
function bigHello() {
    echo "<h1> HELLO </h1>"
}
```
Defining Functions In PHP

```php
function bigHello() {
    echo "<h1>HELLO</h1>";
}
bigHello();
```

Function definition

Function call

Function result

HELLO
Defining Functions In PHP

• For the next example, let’s define a function that requires an argument. Actually, let’s define two different functions that each take an argument.

• The first function will take a string and print the string with a `<br />` element appended to the string. The second function will do the same, but append two `<br />` elements to the end of the string.
<html>
<head>
<title>Defining A Function In PHP With An Argument</title>
</head>
<body style="font-family: arial, sans-serif; background-color: #856363" background=image1.jpg>
<!-- <h3> Using a built in function in PHP</h3> -->
<?php

function appendSingleBreak($text) {
    echo $text."<br /";  
}
//end function appendSingleBreak

function appendDoubleBreak($text) {
    echo $text."<br /><br /";  
}
//end function appendDoubleBreak

appendSingleBreak("This is line 1.");
appendSingleBreak("This is line 2.");
appendSingleBreak("This is line 3.");
appendDoubleBreak("This is line 4.");
appendDoubleBreak("This is line 5.");
appendDoubleBreak("This is line 6.");

?>
</body>
</html>
Defining Functions In PHP

- For the next example, let’s define a function that requires two arguments. We’ll basically repeat the exercise from the previous example, but in this case rather than writing two different functions that differ only in the number of `<br />` elements they append to a line of text, the new function will have a second argument that represents the number of `<br />` elements to be appended. Clearly this would be more efficient, in terms of code, than creating a different function for each number of `<br />` elements we might want to append.

- In the first version of this example, shown on the next page, I simply repeated the same effect as in the previous version, so the two screen shots from the browser should look identical.

- The second version of this example, shown on page 15, a different effect is produced by the function calls.
<html>
<head>
<title>Defining A Function In PHP With Two Arguments</title>
</head>
<body style="font-family: arial, sans-serif; background-color: #856363" background=image1.jpg>
<!-- <h3> Using a built in function in PHP </h3> -->
<?php

function appendBreaks($text, $number) {
    echo $text;
    for($i = 1; $i <= $number; $i++){
        echo "<br/>";
    }
} //end function appendBreaks
appendBreaks("This is line 1.", 1);
appendBreaks("This is line 2.", 1);
appendBreaks("This is line 3.", 1);
appendBreaks("This is line 4.", 2);
appendBreaks("This is line 5.", 2);
appendBreaks("This is line 6.", 0);
?
</body>
</html>
<?php

function appendBreaks($text, $number) {
    echo $text;
    for($i = 1; $i <= $number; $i++){
        echo "<br />";
    }
} //end function appendBreaks

appendBreaks("This is line 1.", 4);
appendBreaks("This is line 2.", 1);
appendBreaks("This is line 3.", 3);
appendBreaks("This is line 4.", 10);
appendBreaks("This is line 5.", 2);
appendBreaks("This is line 6.", 0);

?>

</body>
</html>
Defining Functions In PHP

• As a final example of simple function definition, let’s construct a function that returns a value.

• In the previous two examples, the string that had the `<br />` elements appended to it was simply printed out in the browser. Sometimes, however, you will want the function to provided a value that you can work with yourself. For example, if the function had returned the appended string, we could have passed that to another function to further process the amended string before it was printed.

• Let’s construct a function that will take three integer arguments and determine the largest of the argument values and return this value to the caller.
<html>
  <head>
    <title>Defining A Function In PHP With A Return Value</title>
  </head>
  <body style = "font-family: arial, sans-serif; background-color: #856363" background=img1.jpg>
    <?php
      function findMax($one, $two, $three) {
        if (($one >= $two) && ($one >= $three)) {
          return $one;
        } elseif (($two >= $one) && ($two >= $three)) {
          return $two;
        } elseif (($three >= $one) && ($three >= $two)) {
          return $three;
        }
      } //end function findMax
      $currentMax = findMax(12,13,14);
      echo "The maximum value is: \$currentMax <br />
      $currentMax = findMax(2,4,3);
      echo "The maximum value is: \$currentMax <br />
      $currentMax = findMax(24,12,3);
      echo "The maximum value is: \$currentMax <br />
    ?></body>
  </html>
Defining Functions In PHP

• The `return` statement can return a value or nothing at all.

• How you arrive at a value passed by a `return` statement can vary.
  – The value can be hard-coded: `return 4;`
  – It can be the result of an expression: `return $a/$b;`
  – It can be the value returned by yet another function call:
    
    ```
    return anotherFunction($an_argument);
    ```
Variable Scope

- A variable that is declared within a function remains local to that function. In other words, that variable is not available outside of the function or within other functions.

- This is referred to as the scope of a variable.

- This also implies that variable names are not required to be unique across functions. Therefore the same variable can be defined in more than one function.

- The following example illustrates the scope of a variable. Notice that both functions functionOne and functionTwo declare variables named myInt.
<?php

function functionOne($one) {
    $myInt = $one;
    echo 'The value of \$one in functionOne is: \$myInt. \n';
}

function functionTwo($two) {
    $myInt = $two;
    echo 'The value of \$two in functionTwo is: \$myInt. \n';
}

functionOne($arg1);
functionTwo($arg2);
// echo 'The value of \$one is: \$one. \n'; // uncomment this line to see error
// echo 'The value of \$two is: \$two. \n'; // uncomment this line to see error
echo 'The value of \$myInt is: \$myInt. \n';

?>
Defining Functions In PHP

Inside the functions the variable is visible (it is in scope).

Outside the functions the variables are not visible (they are out of scope).
Variable Scope

• Similar to a variable defined inside a function having no scope outside of the function, a variable declared outside of a function is not accessible from inside the function.

• In general, if a function needs information from outside of the function in order to accomplish its task, the information should be passed as an argument to the function.

• Having said this however, there are times when you might want to access an important variable without passing it in as an argument. This is accomplished in PHP with the global statement. The global statement allows a function to access a variable declared outside of the function. More than one variable can be declared global at one time by separating the variable names with commas.

• The following example illustrates this concept using a variation of the previous example.
<!DOCTYPE html>
<html>
<head>
    <title>Variable Scope In PHP Functions - Version 2 - Using Global Statement</title>
</head>
<body style="font-family: arial, sans-serif; background-color: #856366" background=image1.jpg>
<h3>Using the global statement in PHP</h3>
<pre>
<?php

    $alpha = 14; // $alpha is declared outside of any functions
    $beta = 10; // $beta too is declared outside of any functions

    function functionOne($one)
    {
        global $alpha; // allow functionOne access to $alpha
        $myInt = $one;
        $alpha = $alpha + $myInt;
        echo 'The value of'. ' . $myInt . ' in functionOne is: '. $myInt . '.<br />
        echo 'The value of'. ' . $alpha . ' in functionOne is: '. $alpha . '.<br />
    }

    function functionTwo($two)
    {
        global $alpha; // allow functionTwo access to $alpha
        $myInt = $two;
        echo 'The value of'. ' . $two . '.
        echo 'The value of'. ' . $myInt . ' in functionTwo is: '. $myInt . '.<br />
        // $beta is out of scope here - next line generates an error
        echo 'The value of'. ' . $beta . ' in functionTwo is: '. $beta . '.<br />
    }

    $arg1 = 4;
    $arg2 = 6;
    functionOne($arg1);
    functionTwo($arg2);
    // both $alpha and $beta are in scope here
    echo 'The value of $alpha is: '. $alpha . '.<br />
    echo 'The value of $beta is: '. $beta . '.<br />

</pre>
</body>
</html>
Using the global statement in PHP

The value of $myInt in functionOne is: 4
The value of $alpha in functionOne is: 18

The value of $alpha in functionTwo is: 18
The value of $myInt in functionTwo is: 6

Notice: Undefined variable: beta in C:\Apache24\htdocs\variable scope V2.php on line 24
The value of $beta in functionTwo is:
The value of $alpha is: 18
The value of $beta is: 10

Outside of the functions the variables are again in scope and notice the modified value of $alpha.

Inside functionOne the variable $alpha is visible via the global statement. Note that the function modified the value of $alpha.

Inside functionTwo the variable $alpha is also visible via the global statement. The third echo statement in this function will generate the error when it attempts to reference the variable $beta which is not in scope.
Saving State Between Function Calls

• Local variables within functions have a short but happy life – they come into existence when the function is called and die when the execution of the function is finished.

• Occasionally, however, you might want to give a function a rudimentary memory.

• For example, suppose that you’d like a function to keep track of the number of times it has been called so that numbered headings can be created by a script.

• You could of course accomplish this by using the global statement and accessing a variable declared outside of the function. The example on the next page illustrates this technique.
<html>
<head>
<title>Saving State Using Global Variable</title>
</head>
<body style="font-family: arial, sans-serif; background-color: #856363" background=imagel.jpg>
<?php

   $number_of_calls = 0;  // number of times the function has been called
   function numberedHeading($txt) {
      global $number_of_calls;  //allow function access to $number_of_calls
      $number_of_calls++;
      echo "<h1>".$number_of_calls."".$txt."</h1>";
   }  //end numberedHeading

   numberedHeading("First Header");
   echo "<p> Some text goes here. </p>";
   numberedHeading("Second Header");
   echo "<p> Some text goes here. </p>";
   numberedHeading("Third Header");
   echo "<p> Some text goes here. </p>";

?>
</body>
</html>
Saving State Between Function Calls

1 First Header
Some text goes here.

2 Second Header
Some text goes here.

3 Third Header
Some text goes here.
Saving State Between Function Calls

• The previous example illustrated providing a function some “memory” through the use of a global variable.

• This is not a very elegant way to achieve this task. **Why?**

• **Answer:** Functions that use the global statement cannot be read as standalone blocks of code. In reading or reusing them, you must look out for the global variables that they manipulate. Failing to do so will render the function useless.

• This is where the `static` statement comes into play in PHP.

• Declaring a variable within a function to be static, the variable remains local to the function and the function remembers its value from execution to execution. The next example illustrates the `static` statement.
<?php

function numberedHeading($txt) {
    static $number_of_calls = 0; // number of times the function has been called
    $number_of_calls++;
    echo "<h1>$number_of_calls. $txt</h1>";
}

numberedHeading("First Header");
echo "<p> Some text goes here.</p>";
numberedHeading("Second Header");
echo "<p> Some text goes here.</p>";
numberedHeading("Third Header");
echo "<p> Some text goes here.</p>";
numberedHeading("Fourth Header"); //added to differentiate from previous example
echo "<p> Some text goes here.</p>";
?>
1 First Header
Some text goes here.

2 Second Header
Some text goes here.

3 Third Header
Some text goes here.

4 Fourth Header
Some text goes here.
Setting Default Values For Arguments

- PHP provides a nifty feature to help you construct flexible functions. For functions that require one or more arguments, you can specify that some arguments are optional. This makes your functions more flexible.

- To illustrate the concept of the usefulness of setting default argument values, let’s build a function that takes a string of text and an integer that corresponds to the point size in which the string is to be printed in the browser.

- This is shown on the next page.
<?php

function fontWrapper($txt, $fontsize) {
    echo "<span style="font-size: $fontsize" >".$txt."</span>";
}
//end numberedHeading

fontWrapper("A Heading <br />, "24pt");
fontWrapper("Some text in the body <br />, "16pt");
fontWrapper("Some smaller text in the body <br />, "12pt");
fontWrapper("Some even smaller text in the body <br />, "8pt");

?>

</body>

</html>
Setting Default Values For Arguments

- The nifty feature that PHP provides is to allow you to assign a value to a function argument within the function definition’s parentheses.

- The effect of this is to make the argument value passed from the caller optional as the argument will assume the default value if no value is provided by the caller. The next example modifies the previous example to make use of this feature of PHP.

WARNING

You can create as many optional arguments to a function as you wish. However, the arrangement of the arguments becomes important. Once an optional argument is defined in a function definition, all subsequent arguments must also be optional. In other words, you cannot have the first argument be optional, the second argument required, the third argument optional and so on. The ordering must be: all required arguments followed by all optional arguments.
The second parameter has a default value specified making it an optional parameter to the function. The 2nd and 3rd calls make use of this default value.
Passing Variable References To Functions

- When you pass arguments to functions, they are stored as copies in parameter variables. This means that any changes made to these variables by the function is local to the function and are not reflected beyond it.

- The example on the next page illustrates argument passing by value.

```php
<?php
function add10($num) {
    $num += 10;
    echo "In the function the value is: $num <br />
} //end add10
$original_num = 10;
add10($original_num);
echo "The number is: $original_num <br />
?>
</body>
</html>
```
Upon return the value of $original_num is unchanged by the function.
Passing Variable References To Functions

• By default in PHP, variables passed to functions are passed by value. In other words, only local copies of the variables are used by the functions and the original values of the variables are not accessible by the function.

• So how can you allow a function to actually modify a variable sent to it? You must create a reference to the variable.

• The reference operator in PHP is the & (ampersand). Placing an ampersand in front of an argument in a function definition creates a reference to the variable and allows the function to modify the original variable.

• The following example modifies the previous example to make use of passing an argument by reference.
The argument \$num is passed by reference since it is preceded with the & operator. Upon return from the function the value of \$original_num has been changed.
Arrays In PHP

• Most of our PHP examples to this point have involved scalar variables (we did see a couple of example in the first section of notes that made use of one of PHP’s global associative arrays).

• Scalar variables can only hold a single value at a time. For example, a variable $color could hold only a single value such as red, at any point in time. The variable could not be used to hold more than one color.

• Arrays are special types of variables that enable you to store as many values as you want.

Note: Although you can technically make an array as large as you’d like, some built-in array handling functions in PHP have an upper limit of 100,000 values. If you are storing more data that this in your arrays and you need to use one of these functions, you will either need to write your own function or split the data into multiple arrays.
Arrays In PHP

- Arrays are indexed, which means that each entry in the array, called an element, is made up of a key and a value.

- The key is the index position, beginning with 0 and increasing incrementally by 1 with each new element in the array.

- The value is whatever value you associate with that position – a string, an integer, or whatever you want.

- In PHP you can think of an array as a filing cabinet and each key/value pair as a file folder. The key is the label written on the tab of the folder, and the value is what is inside. What’s inside each folder can vary from folder to folder.
Creating Arrays In PHP

• You can create an array using either the \texttt{array()} function or the array operator [ ].

• The \texttt{array()} function is usually used when you want to create a new array and populate it with more than one element, all at the same time.

• The array operator is more often used when you want to create a new array with just one element at the outset or when you want to add to an existing array element.

• The examples on the following couple of pages illustrate creating an array in PHP using these two techniques.
This version uses the array() function to create the array.
This version uses the array operator [ ] to create the array. Note that no index values are specified, PHP will auto number for you.
This version also uses the array operator \[ \] to create the array. Note that index values are specified in this case.
Creating Arrays In PHP

- As shown in the example on page 44, PHP can automatically index the array for you when you use the [ ] operator to create the array.

- This is useful in that it eliminates the possibility that you might misnumber the elements. The example on the next page illustrates what happens if you misnumber the elements in an array.
```php
<?php

$rainbow[0] = "red";
$rainbow[1] = "orange";
$rainbow[2] = "yellow";
$rainbow[3] = "green";
$rainbow[5] = "blue";
$rainbow[7] = "indigo";
$rainbow[8] = "violet";

for ($i=0; $i<=6; $i++) {
    echo 'Rainbow[. . $i. ] color is: '. $rainbow[$i]."<br />
}
?

</body>
</html>
```

Misnumbering starts here with no element 4 defined and then 6 too is missed.
Rainbow[0] color is: red
Rainbow[1] color is: orange
Rainbow[2] color is: yellow
Rainbow[3] color is: green

**Notice:** Undefined offset: 4 in `C:\Apache24\htdocs\array definition V4 misnumbering.php` on line 16
Rainbow[4] color is:
Rainbow[5] color is: blue

**Notice:** Undefined offset: 6 in `C:\Apache24\htdocs\array definition V4 misnumbering.php` on line 16
Rainbow[6] color is:
Creating Associative Arrays In PHP

• The arrays we’ve seen so far have been numerically indexed, meaning that they use an integer index position as the key.

• Associative arrays utilize actual named keys. In PHP, the named keys of an associative array are character strings rather than numerical values. The string value is used to look up or provide a cross-reference to the data value.

• The following example creates an associative array named $instructor with three elements.

```
    $instructor[“CNT 4714”] = “Llewellyn”;  
    $instructor[“CIS 3003”] = “Eisler”;    
    $instructor[“CIS 3360”] = “Guha”;     
```
Creating Associative Arrays In PHP

- The same array could also be created using the `array()` function instead of the array operator `[ ]`. This is shown below:

  ```php
  $instructor = array (“CNT 4714” => “Llewellyn”, “CIS 3003” => “Eisler”, “CIS 3360” => “Guha”);
  ```

- When using the `array()` function, items are assigned in index/value pairs using the `=>` operator.

- When you want to access an item in an associative array, a syntax similar to that used with sequential (numerically indexed) arrays is employed, however, a string value or variable is used for the index.
Creating Associative Arrays In PHP

- Suppose you wanted to retrieve the instructor for CIS 4004. The following expression would achieve this:

  \[
  \text{\$teacher} = \text{\$instructor["CNT 4714"]};
  \]

- The variable $\text{\$teacher}$ would be assigned the data value associated with “CNT 4714” which would be “Llewellyn”.

Note: You might be tempted to do the following with an associative array, where you are trying to determine which course is taught by the instructor named “Llewellyn”:

  \[
  \text{\$course} = \text{\$instructor["Llewellyn"]};
  \]

Don’t do this! An associative array can fetch data values only via the keys and not the values associated with the keys. Therefore, it cannot find and entry in the array with an index value of “Llewellyn” and will return nothing and the value of $\text{\$course}$ will be undefined. The example on the following page illustrates this.
<?php

$instructor = array("CNT 4714" => "Llewellyn",
                   "CIS 3003" => "Eisler",
                   "CIS 3360" => "Guha");

echo 'The instructor of CNT 4714 is: ' . $instructor["CNT 4714"] . "<br />

echo 'The course taught by Eisler is: ' . $instructor["Eisler"] . "<br />

?>

<!DOCTYPE/html>
<html>
<head>
<title>Creating An Associative Array - Incorrect Version</title>
</head>
<body style = "font-family:arial, sans-serif; background-color: #856363" background=image1.jpg>
</body>
</html>
<html>
<head>
<title>Creating An Associative Array - Correct Version</title>
</head>
<body style="font-family: arial, sans-serif; background-color: #856363" background="image1.jpg">

<?php

    $instructor = array( "CNT 4714" => "Llewellyn",
                        "CIS 3003" => "Eisler",
                        "CIS 3360" => "Guha" );

    echo 'The instructor of CNT 4714 is: '. $instructor["CNT 4714"]. "<br />

    echo 'The instructor of CIS 3003 is: '. $instructor["CIS 3003"]. "<br />

?>

</body>
</html>

The instructor of CNT 4714 is: Llewellyn
The instructor of CIS 3003 is: Eisler
Using Associative Arrays In PHP

- A common iterative statement used with both sequential and associative arrays is the `foreach` statement.
- The general syntax of the `foreach` statement is:

  ```php
  foreach ( arrayname as variable ) {
    . . . Statements to repeat
  }
  ```

- The first variable inside the parentheses is the variable name representing the array and the second variable is automatically set to the next array item at each iteration of the loop. An example using a sequential array is shown on the next page and one with an associative array array on the following page.
<html>
<head>
<title>Foreach statement</title>
</head>
<body style="font-family: arial, sans-serif; background-color: #856363" background=imagel.jpg>

<?php

$languages = array( "PHP", "C", "Java", "Python", "Ruby", "Pascal", "Fortran" );

echo "<b>A list of computer languages</b> <br /> <br />";
foreach ($languages as $item) {
    echo $item . "<br />
    
}

?>
</body>
</html>
```php
<?php

$inventory = array("hard drives" => 10, "printers" => 4, "monitors" => 23);

echo "<b> Current Inventory </b> <br /> <br />";

foreach ($inventory as $index => $item) {
    echo $index . ': ' . $item . ' ';
}

?>

</body>

</html>
```
Using Associative Arrays In PHP

• Changing values, adding elements, deleting elements, and verifying an element are all among the common operations that you’ll need to perform on an associative array.

• Changing an existing value is done through simple assignment. For example, to update the number of monitors in the previous example from 23 to 5, the following statement would be used: `$inventory["monitors"] = 5;`.

• To add a new element to an associative array, use the array operator `[]` as in: `$inventory["keyboards"] = 12;`

• Deleting an element from an associative array is done using the `unset()` function.
<html>
<head>
<title>Some operations on an associative array</title>
</head>
<body style = "font-family: arial, sans-serif;
background-color: #856363" background=image1.jpg>

<?php

$inventory = array( "hard drives" => 10, "printers" => 4, "monitors" => 23);
echo "<b><u>Current Inventory</u></b></br></br>
foreach ($inventory as $index => $item) {
    echo "$index : $item</b></br>
}

$inventory["monitors"] = 5;
$inventory["keyboards"] = 12;
foreach ($inventory as $index => $item) {
    echo "$index : $item</b></br>
}
unset($inventory["printers"]);
foreach ($inventory as $index => $item) {
    echo "$index : $item</b></br>
}

?>
</body>
</html>
Using Associative Arrays In PHP

• To verify if a particular index exists in an associative array, use the `isset()` function.

• The `isset()` function returns true if index passed as an argument appears in the associative array and false otherwise.

• The example on the following page illustrates using the `isset()` function.
<?php

$inventory = array("hard drives" => 10, "printers" => 4, "monitors" => 23);

foreach ($inventory as $index => $item) {
    echo $index . ' = ' . $item . '<br />
    }

echo '<br />';

if (isset($inventory["monitors"])) {
    echo "monitors is in the array. <br />
} else {
    echo "monitors is not in the array. <br />
}

if (isset($inventory["keyboards"])) {
    echo "keyboards is in the array. <br />
} else {
    echo "keyboards is not in the array. <br />
}

?>

<title>Using the isset() function on an associative array</title>
<body style="font-family: arial, sans-serif; background-color: #856363" background-image=image1.jpg>

Current Inventory

hard drives = 10
printers = 4
monitors = 23

monitors is in the array.
keyboards is not in the array.
As with many things in PHP, associative array indices are case-sensitive. Thus, in the previous example, if the call to the `isset()` function were passed the parameter “Monitors” instead of “monitors” it would return false instead of true.

See next page.
<?php

$inventory = array( "hard drives" => 10, "printers" => 4, "monitors" => 23);

foreach ($inventory as $index => $item) {
    echo $index . ' = ' . $item . '<br />
);
}

if (isset($inventory["Monitors"])) {
    echo "Monitors is in array. <br />
} else {
    echo "Monitors is not in array. <br />
}

if (isset($inventory["keyboards"])) {
    echo "keyboards is in array. <br />
} else {
    echo "keyboards is not in array. <br />
}

?>

// Associative array indices are case-sensitive - monitors is in array
// Monitors is not
Sorting Associative Arrays In PHP

- PHP has a special set of functions for sorting associative arrays.
- The `asort()` function sorts an associative array and maintains the relationship between the indices and the values. The sort is based upon the values in the associative array passed as an argument to the function. The sort order is ascending based on the value. The `arsort()` function sorts in descending order based on value.
- The `ksort()` function is similar to the `asort()` function but it sorts an associative array using the indices (in ascending order) as the sort field. The `krsort()` function sorts in descending order using the indices.
- These various sort functions are shown on the next few pages.
<?php
$cities['Boston'] = 1302;
$cities['Dallas'] = 1093;
$cities['Toronto'] = 1282;
$cities['Chicago'] = 1154;
$cities['Los Angeles'] = 2512;
$cities['San Francisco'] = 2825;
$cities['Washington'] = 853;
$cities['Miami'] = 235;
$cities['Pittsburgh'] = 972;
$cities['Tampa'] = 85;

asort($cities);
print("<b>Cities sorted by distance from Orlando.</b> <br />"");
foreach ($cities as $index => $value) {
    print(" $index = $value <br />");
}
?>
</html>
<?php

$cities['Boston'] = 1302;
$cities['Dallas'] = 1093;
$cities['Toronto'] = 1282;
$cities['Chicago'] = 1154;
$cities['Los Angeles'] = 2512;
$cities['San Francisco'] = 2825;
$cities['Washington'] = 853;
$cities['Miami'] = 235;
$cities['Pittsburgh'] = 972;
$cities['Tampa'] = 85;

arsort($cities);
print("<b><u>Cities sorted by distance from Orlando.</u></b> <br />

foreach ($cities as $index => $value) {
    print(" $index = $value <br />");
}

?>

</body>
</html>
Using `ksort()`

```php
<?php
$cities['Boston'] = 1302;
$cities['Dallas'] = 1093;
$cities['Toronto'] = 1282;
$cities['Chicago'] = 1154;
$cities['Los Angeles'] = 2512;
$cities['San Francisco'] = 2825;
$cities['Washington'] = 853;
$cities['Miami'] = 235;
$cities['Pittsburgh'] = 972;
$cities['Tampa'] = 85;

ksort($cities);
print("<b><u>Cities sorted by distance from Orlando.</u></b> <br />

foreach ($cities as $index => $value) {
    print(" $index = $value <br />");
}
?>
</html>
```
Using `krsort()`

```php
<?php
$cities['Boston'] = 1302;
cities['Dallas'] = 1093;
cities['Toronto'] = 1282;
cities['Chicago'] = 1154;
cities['Los Angeles'] = 2512;
cities['San Francisco'] = 2825;
cities['Washington'] = 853;
cities['Miami'] = 235;
cities['Pittsburgh'] = 972;
cities['Tampa'] = 85;

krsort($cities);
print("<br />

Cities sorted by distance from Orlando.
</u></b> <br />(br /)<br />

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Washington</td>
<td>853</td>
</tr>
<tr>
<td>Toronto</td>
<td>1282</td>
</tr>
<tr>
<td>Tampa</td>
<td>85</td>
</tr>
<tr>
<td>San Francisco</td>
<td>2825</td>
</tr>
<tr>
<td>Pittsburgh</td>
<td>972</td>
</tr>
<tr>
<td>Miami</td>
<td>235</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>2512</td>
</tr>
<tr>
<td>Chicago</td>
<td>1154</td>
</tr>
<tr>
<td>Dallas</td>
<td>1093</td>
</tr>
<tr>
<td>Boston</td>
<td>1302</td>
</tr>
</tbody>
</table>

?>
```
Using Multidimensional Arrays In PHP

• Some data are best represented by creating a list of lists (a multidimensional array).

• Consider the following table listing the inventory for a hardware store:

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Part Name</th>
<th>Count</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC1000</td>
<td>Hammer</td>
<td>122</td>
<td>28.50</td>
</tr>
<tr>
<td>AC1001</td>
<td>Wrench</td>
<td>25</td>
<td>14.00</td>
</tr>
<tr>
<td>AC1002</td>
<td>Saw</td>
<td>18</td>
<td>25.00</td>
</tr>
<tr>
<td>AC1003</td>
<td>Screwdriver</td>
<td>34</td>
<td>4.50</td>
</tr>
</tbody>
</table>

• The example on the next page represents this data in a two-dimensional associative array.
<html>
<head>
<title>Hardware Inventory</title>
</head>
<body style="font-family: arial, sans-serif; background-color: #856363" background=image1.jpg>
<form action="show_inventory.php" method="post">
<font size=4 color="blue">Mark's Hardware Inventory Information</font><br>
Select part number for more information<br>
</form>
<?php
    $Inventory = array('AC1000', 'AC1001', 'AC1002', 'AC1003');
    foreach ($Inventory as $item) {
        print "<input type=radio name="id" value=$item > $item ";
    }
    print '<br/>'<br/>input type=submit value="Submit"&nbsp;&nbsp; &nbsp;';
    print '<input type=reset value="Erase ">';
    print '</form> </body></html>';
PHP script to find the correct entry in the associative array based on the user’s selection, then display the associative array entries for that item.
Mark's Hardware Inventory Information

Select part number for more information

- AC1000
- AC1001
- AC1002
- AC1003

1. User selects a part number
2. PHP script displays part details.

Inventory Information for Part AC1001

<table>
<thead>
<tr>
<th>ID</th>
<th>Part</th>
<th>Count</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC1001</td>
<td>Wrench</td>
<td>25</td>
<td>14.00</td>
</tr>
</tbody>
</table>