

CNT 4714: Enterprise Computing Fall 2013

Introduction to PHP – Part 2 Functions and Arrays

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
<http://www.cs.ucf.edu/courses/cnt4714/fall2013>

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida



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.



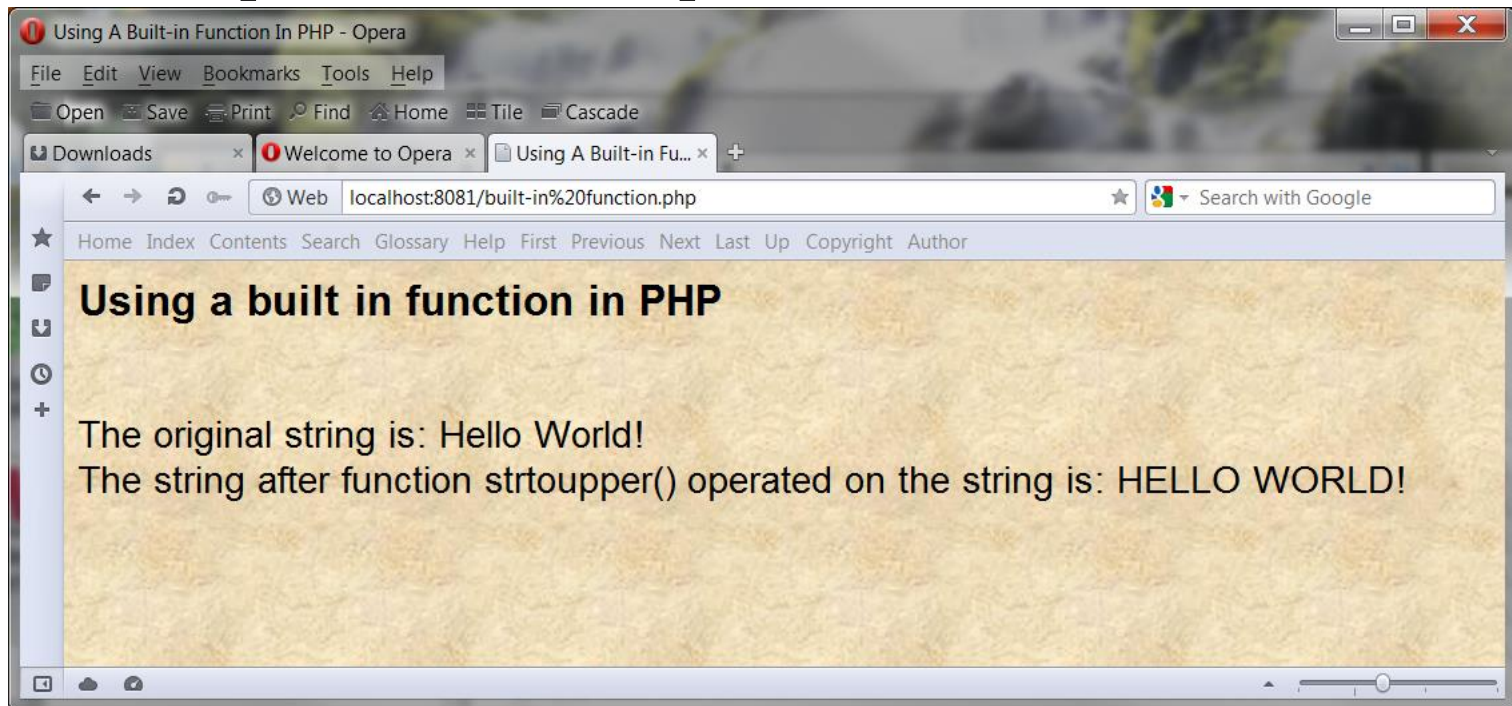
Functions In PHP

- 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.

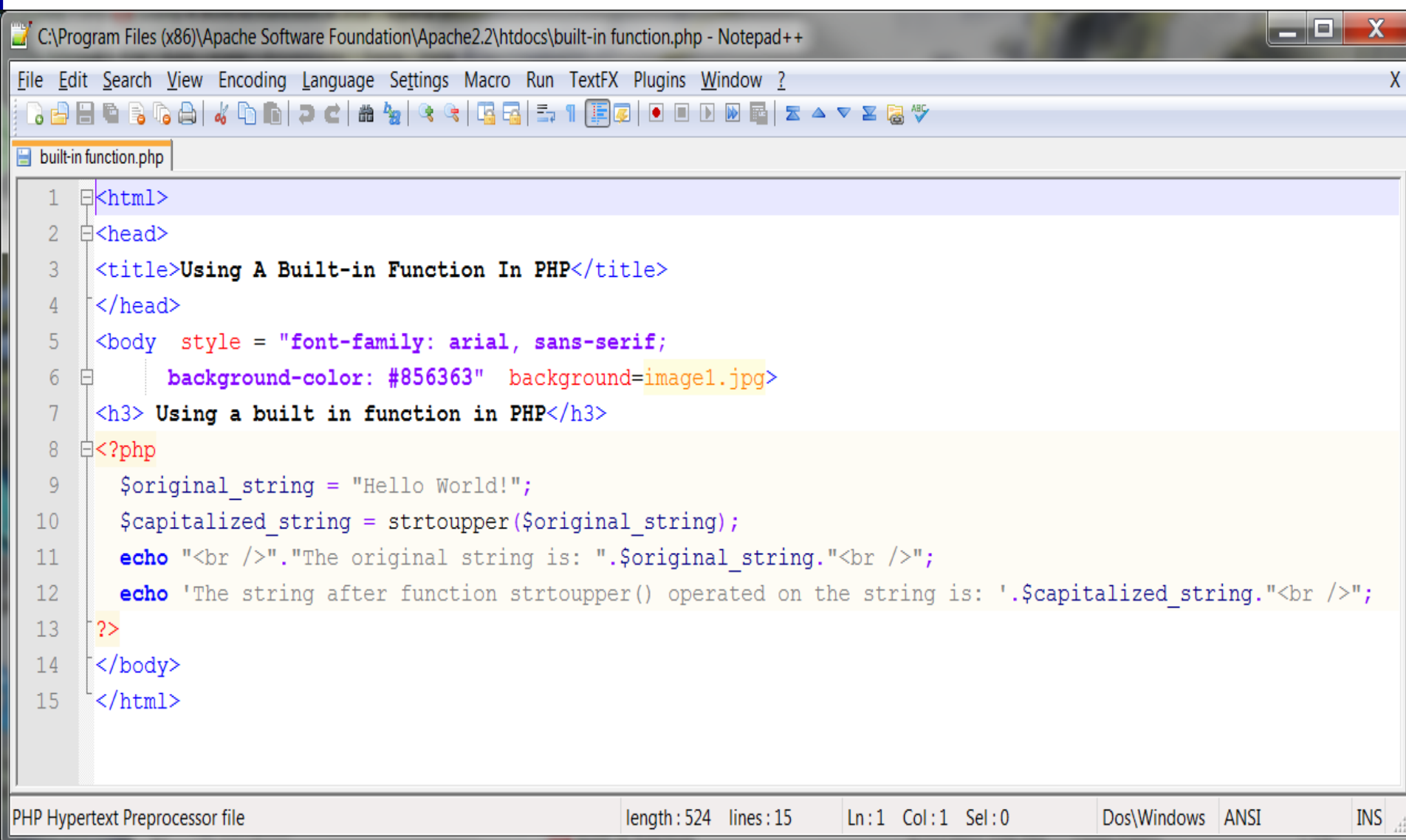


Functions In PHP

- 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:



Functions In PHP



```
C:\Program Files (x86)\Apache Software Foundation\Apache2.2\htdocs\built-in function.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
built-in function.php
1 <html>
2 <head>
3 <title>Using A Built-in Function In PHP</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <h3> Using a built in function in PHP</h3>
8 <?php
9     $original_string = "Hello World!";
10    $capitalized_string = strtoupper($original_string);
11    echo "<br />". "The original string is: ".$original_string."<br />";
12    echo 'The string after function strtoupper() operated on the string is: '.$capitalized_string."<br />";
13    ?>
14 </body>
15 </html>
PHP Hypertext Preprocessor file length: 524 lines: 15 Ln:1 Col:1 Sel:0 Dos\Windows ANSI INS
```



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:

```
some_function ($an_argument, $another_argument);
```



Functions In PHP

- The `strtoupper()` 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:

```
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.

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



Defining Functions In PHP

The image shows a Notepad++ editor window titled "C:\Program Files (x86)\Apache Software Foundation\Apache2.2\htdocs\bigHello.php - Notepad++". The editor contains the following PHP code:

```
1 <html>
2 <head>
3 <title>Defining A Function In PHP</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=image1.jpg>
7 <!-- <h3> Using a built in function in PHP</h3> -->
8 <?php
9     function bigHello() {
10         echo "<h1> HELLO </h1>";
11     }
12     bigHello();
13 ?>
14 </body>
15 </html>
```

Annotations in the image point to specific parts of the code:

- Function definition:** Points to the function definition block on lines 9-11.
- Function call:** Points to the function call `bigHello();` on line 12.
- Function result:** Points to the output "HELLO" displayed in the browser window.

The browser window (Opera) shows the rendered output of the PHP code, which is "HELLO". The browser's address bar shows `localhost:8081/1`.



Defining Functions In PHP

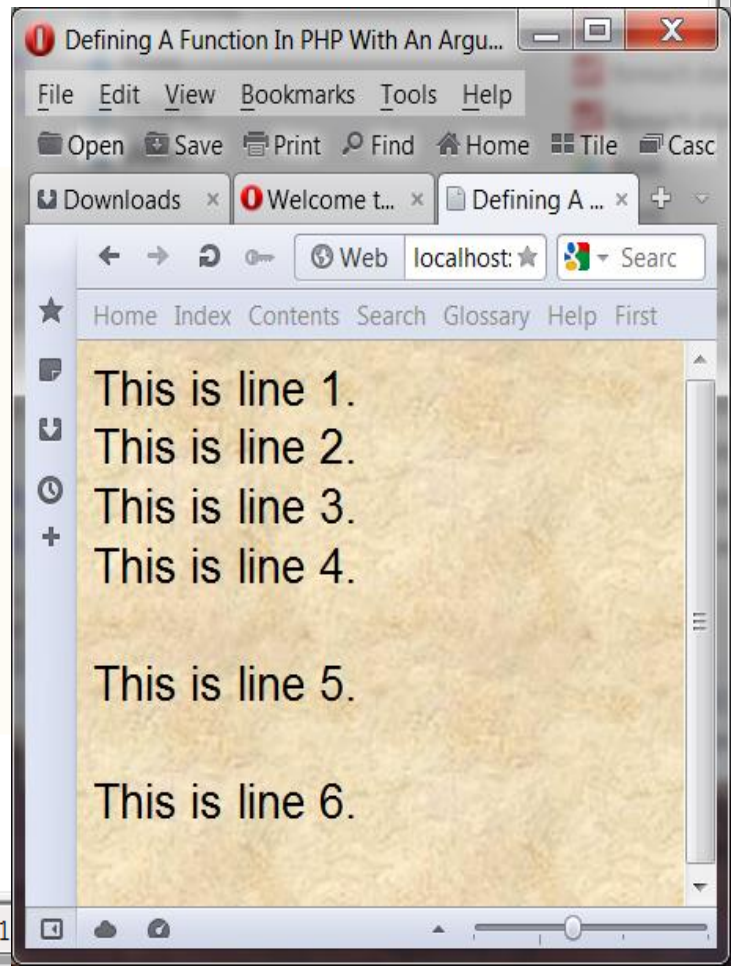
- For the next example, let's define a function that requires an argument. Actually, let's define two different functions that each take an argument.
- The first function will take a string and print the string with a `
` element appended to the string. The second function will do the same, but append two `
` elements to the end of the string.



```

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
built-in function.php bigHello.php functionWithArgument.php
1 <html>
2 <head>
3 <title>Defining A Function In PHP With An Argument</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imageUrl.jpg>
7 <!-- <h3> Using a built in function in PHP</h3> -->
8 <?php
9     function appendSingleBreak($text) {
10         echo $text."<br />";
11     } //end function appendSingleBreak
12     function appendDoubleBreak($text) {
13         echo $text."<br /><br />";
14     } //end function appendDoubleBreak
15     appendSingleBreak("This is line 1.");
16     appendSingleBreak("This is line 2.");
17     appendSingleBreak("This is line 3.");
18     appendDoubleBreak("This is line 4.");
19     appendDoubleBreak("This is line 5.");
20     appendDoubleBreak("This is line 6.");
21 ?>
22 </body>
23 </html>
PHP Hypertext Preprocessor file length: 735 lines: 23 Ln: 1

```



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 `
` elements they append to a line of text, the new function will have a second argument that represents the number of `
` elements to be appended. Clearly this would be more efficient, in terms of code, than creating a different function for each number of `
` 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.

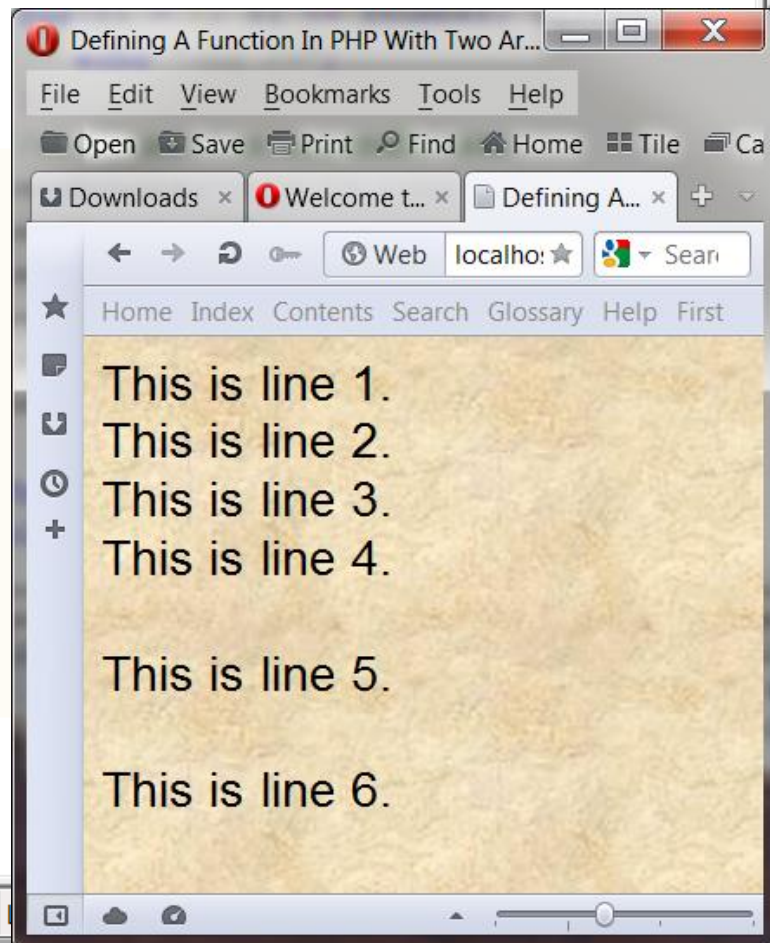


```

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
built-in function.php bigHello.php functionWithArgument.php functionWithTwoArguments.php
1 <html>
2 <head>
3 <title>Defining A Function In PHP With Two Arguments</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imageUrl.jpg>
7 <!-- <h3> Using a built in function in PHP</h3> -->
8 <?php
9 function appendBreaks($text, $number) {
10     echo $text;
11     for($i = 1; $i <= $number; $i++){
12         echo "<br />";
13     }
14 } //end function appendBreaks
15 appendBreaks("This is line 1.", 1);
16 appendBreaks("This is line 2.", 1);
17 appendBreaks("This is line 3.", 1);
18 appendBreaks("This is line 4.", 2);
19 appendBreaks("This is line 5.", 2);
20 appendBreaks("This is line 6.", 0);
21 ?>
22 </body>
23 </html>

```

PHP Hypertext Preprocessor file length : 668 lines : 23



```

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
functionWithTwoArgumentsV2.php
1 <html>
2 <head>
3 <title>Defining A Function In PHP With Two Arguments - Version 2</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=image1.jpg>
7 <!-- <h3> Using a built in function in PHP</h3> -->
8 <?php
9 function appendBreaks($text, $number) {
10     echo $text;
11     for($i = 1; $i <= $number; $i++){
12         echo "<br />";
13     }
14 } //end function appendBreaks
15 appendBreaks("This is line 1.", 4);
16 appendBreaks("This is line 2.", 1);
17 appendBreaks("This is line 3.", 3);
18 appendBreaks("This is line 4.", 10);
19 appendBreaks("This is line 5.", 2);
20 appendBreaks("This is line 6.", 0);
21 ?>
22 </body>
23 </html>

```

PHP Hypertext Preprocessor file length : 681 lines : 23 Ln : 1 Col : 1 Sel : 0

```

Defining A Function In PH...
File Edit View Bookmarks Tools Help
Open Save Print Find Home
Downl... Welco... Defini...
Web localhost:
Home Index Contents Search Glossary
This is line 1.
This is line 2.
This is line 3.
This is line 4.
This is line 5.
This is line 6.

```



Defining Functions In PHP

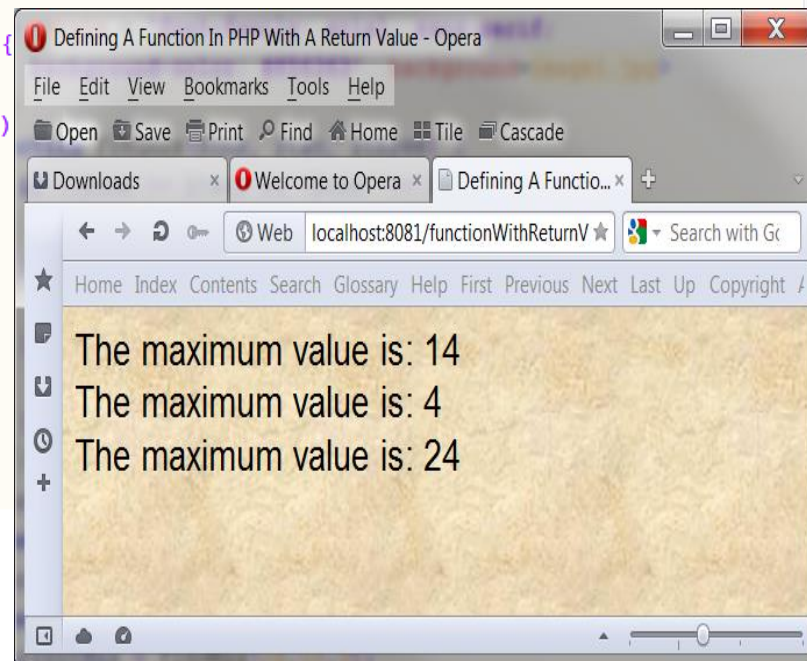
- As a final example of simple function definition, let's construct a function that returns a value.
- In the previous two examples, the string that had the `
` elements appended to it was simply printed out in the browser. Sometimes, however, you will want the function to provide a value that you can 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.



```

1 <html>
2 <head>
3 <title>Defining A Function In PHP With A Return Value</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=image1.jpg>
7 <?php
8     function findMax($one, $two, $three) {
9         if (($one >= $two) && ($one >= $three)) {
10            return $one;
11        } elseif (($two >= $one) && ($two >= $three)) {
12            return $two;
13        } elseif (($three >= $one) && ($three >= $two)) {
14            return $three;
15        }
16    } //end function findMax
17    $currentMax = findMax(12,13,14);
18    echo "The maximum value is: $currentMax <br />";
19    $currentMax = findMax(2,4,3);
20    echo "The maximum value is: $currentMax <br />";
21    $currentMax = findMax(24,12,3);
22    echo "The maximum value is: $currentMax <br />";
23    ?>
24 </body>
25 </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`.



```
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
functionWithTwoArgumentsV2.php functionWithReturnValue.php variable scope.php
1 <html>
2 <head>
3 <title>Variable Scope In PHP Functions</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6 background-color: #856363" background=image1.jpg>
7 <?php
8 function functionOne($one) {
9     $myInt = $one;
10    echo 'The value of '. '$myInt in functionOne is: '. $myInt. "<br />";
11 } //end functionOne
12 function functionTwo($two) {
13     $myInt = $two;
14    echo 'The value of '. '$myInt in functionTwo is: '. $myInt. "<br />";
15 } //end functionTwo
16 $arg1 = 4;
17 $arg2 = 6;
18 functionOne($arg1);
19 functionTwo($arg2);
20 // echo 'The value of $one is: '. $one. "<br />"; //uncomment this line to see error
21 // echo 'The value of $two is: '. $two. "<br />"; //uncomment this line to see error
22 echo 'The value of $myInt is: '. $myInt. "<br />";
23 ?>
24 </body>
25 </html>
```



Defining Functions In PHP

The screenshot shows a web browser window with the address bar at `localhost:8081/variable%20scope.php`. The page content is as follows:

```
The value of $myInt in functionOne is: 4  
The value of $myInt in functionTwo is: 6
```

Below this, three error messages are displayed:

```
Notice: Undefined variable: one in C:\Apache24\htdocs\variable scope.php on line 20  
The value of $one is:  
Notice: Undefined variable: two in C:\Apache24\htdocs\variable scope.php on line 21  
The value of $two is:  
Notice: Undefined variable: myInt in C:\Apache24\htdocs\variable scope.php on line 22  
The value of $myInt is:
```

Two callout boxes provide context:

- The top callout box, pointing to the function output, states: "Inside the functions the variable is visible (it is in scope)".
- The bottom callout box, pointing to the error messages, states: "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.



```
C:\Program Files (x86)\Apache Software Foundation\Apache2.2\htdocs\variable scope V2.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
functionWithTwoArgumentsV2.php functionWithReturnValue.php variable scope.php php.ini variable scope V2.php
1 <html>
2 <head>
3 <title>Variable Scope In PHP Functions - Version 2 - Using Global Statement</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6 background-color: #856363" background=imageUrl.jpg>
7 <h3> Using the global statement in PHP </h3>
8 <?php
9 $alpha = 14; // $alpha is declared outside of any functions
10 $beta = 10; // $beta too is declared outside of any functions
11 function functionOne($one) {
12     global $alpha; // allow functionOne access to $alpha
13     $myInt = $one;
14     $alpha = $alpha + $myInt;
15     echo 'The value of '. '$myInt in functionOne is: '. $myInt. "<br />";
16     echo 'The value of '. '$alpha in functionOne is: '. $alpha. "<br /> <br />";
17 } //end functionOne
18 function functionTwo($two) {
19     global $alpha; // allow functionTwo access to $alpha
20     $myInt = $two;
21     echo 'The value of '. '$alpha in functionTwo is: '. $alpha. "<br />";
22     echo 'The value of '. '$myInt in functionTwo is: '. $myInt. "<br />";
23     // $beta is out of scope here - next line generates an error
24     echo 'The value of '. '$beta in functionTwo is: '. $beta. "<br />";
25 } //end functionTwo
26 $arg1 = 4;
27 $arg2 = 6;
28 functionOne($arg1);
29 functionTwo($arg2);
30 //both $alpha and $beta are in scope here
31 echo 'The value of $alpha is: '. $alpha. "<br />"; //note value of $alpha changed by functi
32 echo 'The value of $beta is: '. $beta. "<br />";
33 ?>
34 </body>
```

PHP Hypertext Preprocessor file length : 1420 lines : 35 Ln : 1 Col : 1 Sel : 0 Dos\Windows ANSI INS





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

Using the global statement in PHP

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

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.

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`.



Saving State Between Function Calls

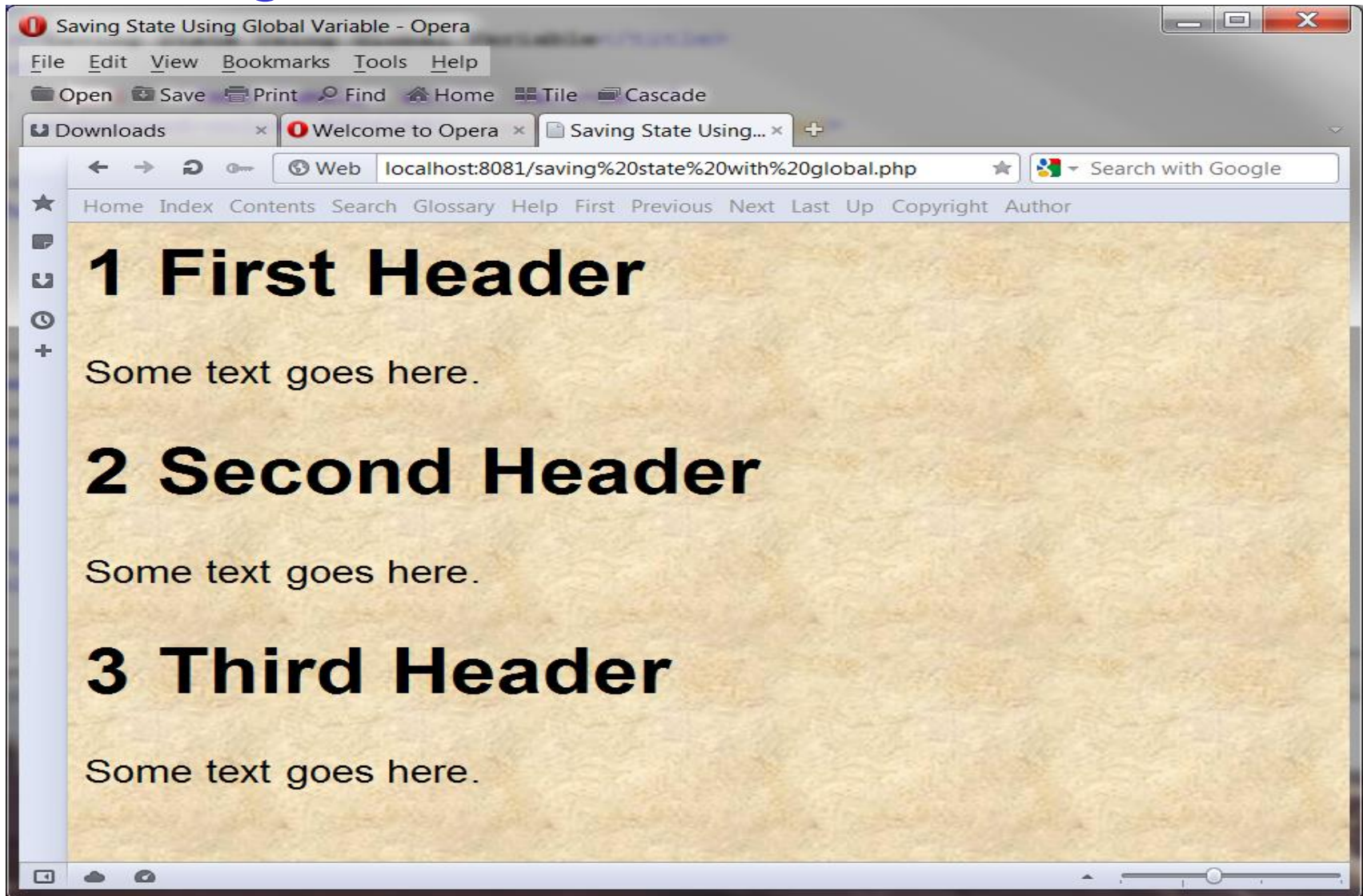
- Local variables within functions have a short but happy life – they come into existence when the function is called and die when the execution of the function is finished.
- Occasionally, however, you might want to give a function a rudimentary memory.
- For example, suppose that you'd like a function 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.



```
C:\Program Files (x86)\Apache Software Foundation\Apache2.2\htdocs\saving state with global.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
functionWithTwoArgumentsV2.php functionWithReturnValue.php variable scope.php php.ini variable scope V2.php saving state with global.php
1 <html>
2 <head>
3 <title>Saving State Using Global Variable</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6 background-color: #856363" background=imageUrl.jpg>
7 <?php
8 $number_of_calls = 0; // number of times the function has been called
9 function numberedHeading($txt) {
10     global $number_of_calls; //allow function access to $number_of_calls
11     $number_of_calls++;
12     echo "<h1>".$number_of_calls." ".$txt."</h1>";
13 } //end numberedHeading
14 numberedHeading("First Header");
15 echo "<p> Some text goes here.</p>";
16 numberedHeading("Second Header");
17 echo "<p> Some text goes here.</p>";
18 numberedHeading("Third Header");
19 echo "<p> Some text goes here.</p>";
20 ?>
21 </body>
22 </html>
PHP Hypertext Preprocessor file length : 722 lines : 22 Ln : 1 Col : 1 Sel : 0 Dos\Windows ANSI INS
```



Saving State Between Function Calls



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.



```

1 <html>
2 <head>
3 <title>Saving State Using the Static Statement</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imageUrl.jpg>
7 <?php
8     function numberedHeading($txt) {
9         static $number_of_calls = 0; // number of times the function has been called
10        $number_of_calls++;
11        echo "<h1>".$number_of_calls." ".$txt."</h1>";
12    } //end numberedHeading
13    numberedHeading("First Header");
14    echo "<p> Some text goes here.</p>";
15    numberedHeading("Second Header");
16    echo "<p> Some text goes here.</p>";
17    numberedHeading("Third Header");
18    echo "<p> Some text goes here.</p>";
19    numberedHeading("Fourth Header"); //added to differentiate from previous example
20    echo "<p> Some text goes here.</p>";
21 ?>
22 </body>
23 </html>

```



The screenshot shows an Opera browser window with the title "Saving State Using the Static Statement - Opera". The address bar displays "localhost:8081/saving%20state%20using%20static.php". The page content consists of four sections, each with a large header and a line of text below it:

- 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.



File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window

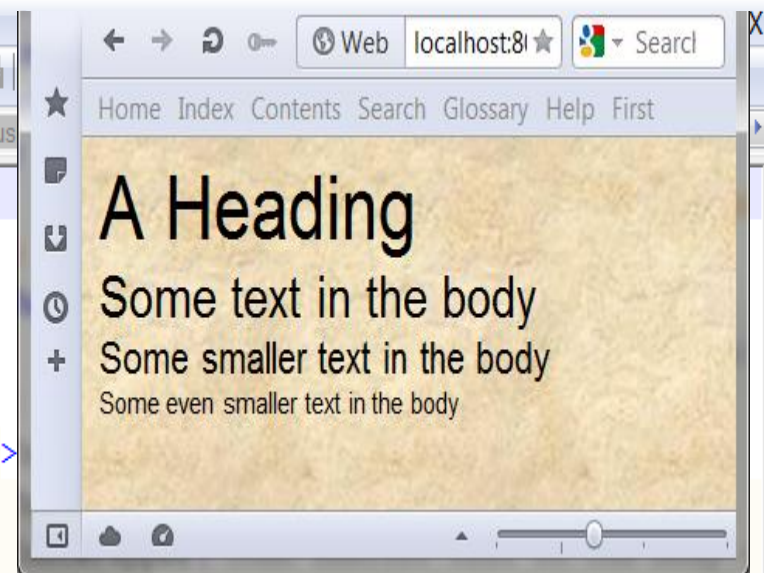


variable scope.php php.ini variable scope V2.php saving state with global.php saving state us

```

1 <html>
2 <head>
3 <title>Default Argument Values - V1</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     function fontWrapper($txt, $fontsize) {
9         echo "<span style=\"font-size:$fontsize\" >\".$txt.\"</span>";
10    } //end numberedHeading
11    fontWrapper("A Heading <br />", "24pt");
12    fontWrapper("Some text in the body <br />", "16pt");
13    fontWrapper("Some smaller text in the body <br />", "12pt");
14    fontWrapper("Some even smaller text in the body <br />", "8pt");
15    ?>
16 </body>
17 </html>

```



PHP Hypertext Preprocessor file

length : 571 lines : 17

Ln : 1 Col : 1 Sel : 0

Dos\Windows ANSI

INS



Setting Default Values For Arguments

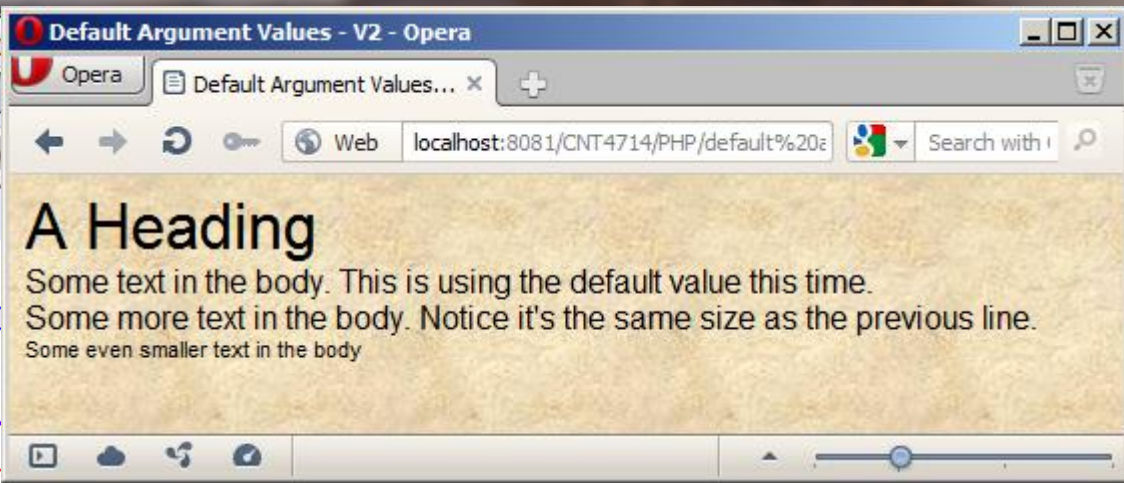
- The nifty feature that PHP provides is to allow you to assign a value to a function argument within the function definition's parentheses.
- The effect of this is to make the argument value passed from the caller optional as the argument will assume the default value if no value is provided by the 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.



```
1 <html>
2 <head>
3 <title>Default Argument Values - V2</title>
4 </head>
5 <body style = "font-family: arial, sans-serif; background-color: #856363" background-color: #856363" >
6
7 <?php
8     function fontWrapper($txt, $fontsize = "12pt") {
9         echo "<span style=\"font-size:$fontsize\" >".$txt."</span>";
10    } //end numberedHeading
11    fontWrapper("A Heading <br />", "24pt");
12    fontWrapper("Some text in the body. This is using the default value this time.<br />");
13    fontWrapper("Some more text in the body. Notice it's the same size as the previous line. <br />");
14    fontWrapper("Some even smaller text in the body <br />", "8pt");
15 ?>
16 </body>
17 </html>
```



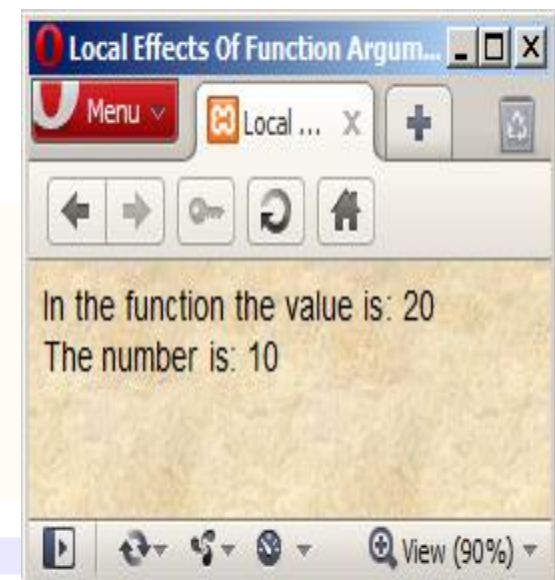
The second parameter has a default value specified making it an optional parameter to the function. The 2nd and 3rd calls make use of this default value.



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.

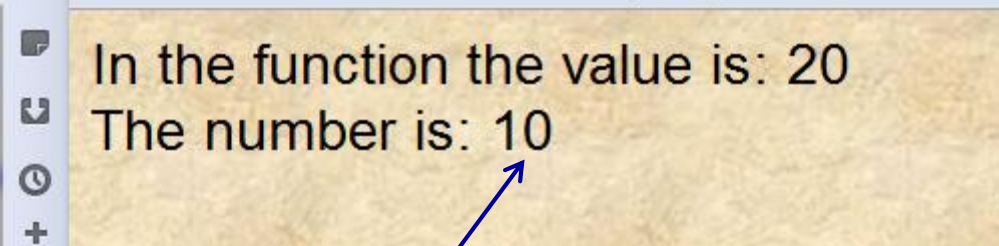
```
2 <head>
3 <title>Local Effects Of Function Arguments</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6   background-color: #856363" background=image1.jpg>
7 <?php
8   function add10($num) {
9       $num += 10;
10      echo "In the function the value is: $num <br />";
11  } //end add10
12  $original_num = 10;
13  add10($original_num);
14  echo "The number is: $original_num <br/>";
15  ?>
16 </body>
17 </html>
```



```

1 <html>
2 <head>
3 <title>Local Effects Of Function
4 </head>
5 <body style = "font-family: arial, sans-serif;
6 background-color: #856363" background=image1.jpg>
7 <?php
8 function add10($num) {
9     $num += 10;
10    echo "In the function the value is: $num <br />";
11 } //end add10
12 $original_num = 10;
13 add10($original_num);
14 echo "The number is: $original_num <br/>";
15 ?>
16 </body>
17 </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.

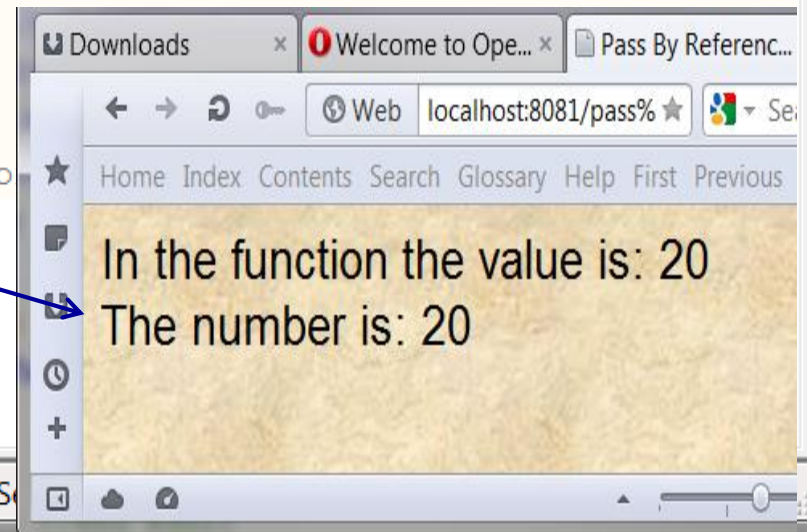


```

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
saving state using static.php default arguments V1.php default arguments V2.php local effects.php pass by reference.php
1 <html>
2 <head>
3 <title>Pass By Reference Example</title>
4 </head>
5 <body style = "font-family: arial, sans-se
6 background-color: #856363" background
7 <?php
8 function add10 (&$num) {
9     $num += 10;
10    echo "In the function the value is: $num <br />";
11 } //end add10
12 $original_num = 10;
13 add10 ($original_num);
14 echo "The number is: $original_num <br />";
15 ?>
16 </body>
17 </html>
PHP Hypertext Preprocessor file length : 417 lines : 17 Ln : 1 Col : 1 S

```

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 than 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.



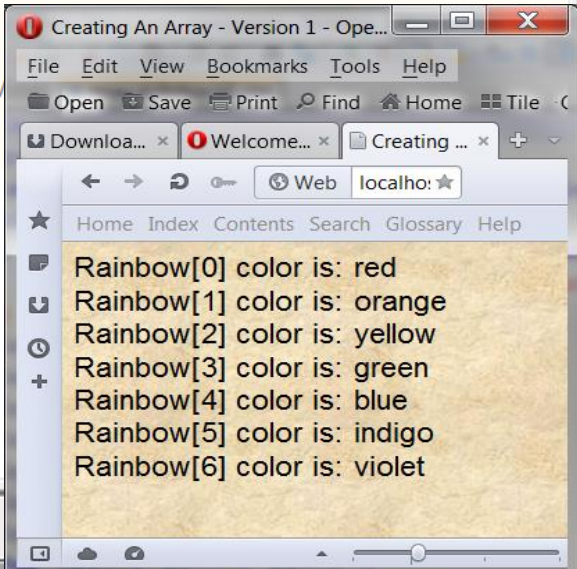
Creating Arrays In PHP

- You can create an array using either the `array()` function or the array operator `[]`.
- The `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.

```
1 <html>
2 <head>
3 <title>Creating An Array - Version 1</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     $rainbow = array("red","orange","yellow","green","blue","indigo","violet");
9     for ($i=0; $i<=6; $i++) {
10         echo 'Rainbow['. $i.'] color is: '. $rainbow[$i]."<br />
11     }
12 ?>
13 </body>
14 </html>
```



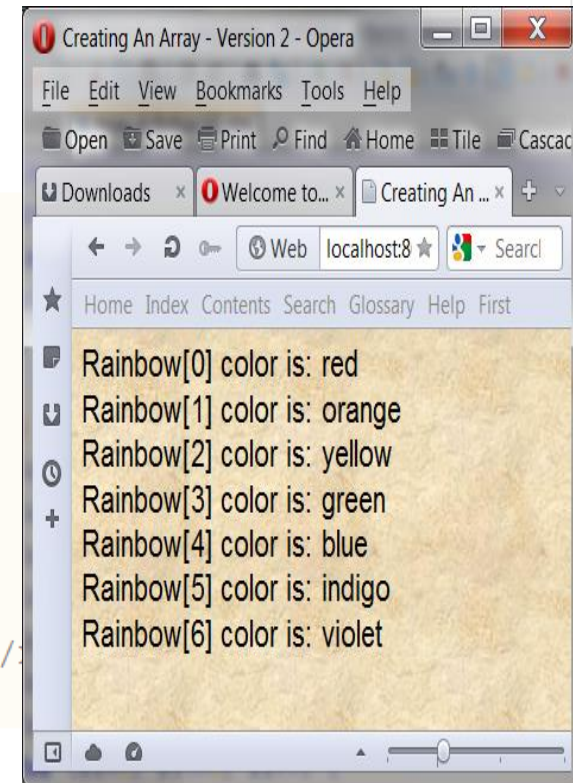
```

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Win
array definition V1.php array definition V2.php
1 <html>
2 <head>
3 <title>Creating An Array - Version 2</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     $rainbow[] = "red";
9     $rainbow[] = "orange";
10    $rainbow[] = "yellow";
11    $rainbow[] = "green";
12    $rainbow[] = "blue";
13    $rainbow[] = "indigo";
14    $rainbow[] = "violet";
15    for ($i=0; $i<=6; $i++) {
16        echo 'Rainbow['. $i.'] color is: '. $rainbow[$i]."<br /
17    }
18 ?>
19 </body>
20 </html>

```

This version uses the array operator [] to create the array.

Note that no index values are specified, PHP will auto number for you

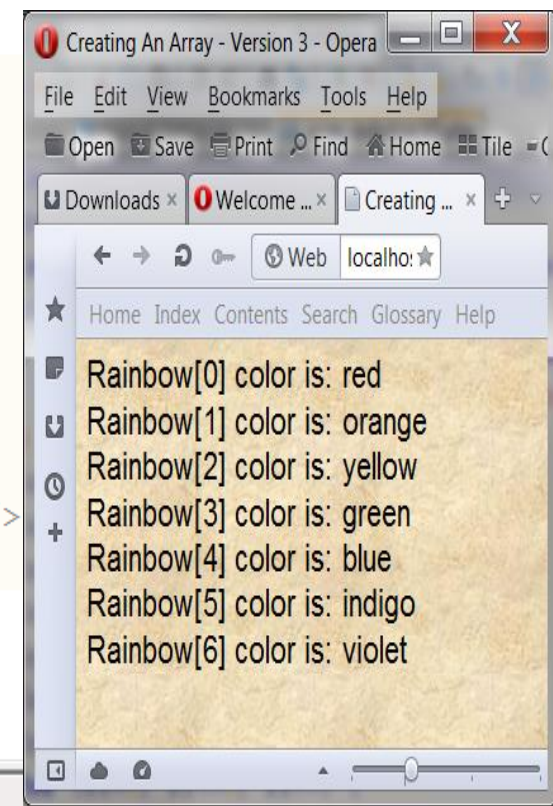


```

1 <html>
2 <head>
3 <title>Creating An Array - Version 3</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     $rainbow[0] = "red";
9     $rainbow[1] = "orange";
10    $rainbow[2] = "yellow";
11    $rainbow[3] = "green";
12    $rainbow[4] = "blue";
13    $rainbow[5] = "indigo";
14    $rainbow[6] = "violet";
15    for ($i=0; $i<=6; $i++) {
16        echo 'Rainbow['. $i.'] color is: '. $rainbow[$i]."<br />";
17    }
18 ?>
19 </body>
20 </html>

```

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.



```
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ? X
array definition V1.php array definition V2.php array definition V3.php array definition V4 misnumbering.php
1 <html>
2 <head>
3 <title>Creating An Array - Version 4 - misnumbered elements</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagel.jpg>
7 <?php
8     $rainbow[0] = "red";
9     $rainbow[1] = "orange";
10    $rainbow[2] = "yellow";
11    $rainbow[3] = "green";
12    $rainbow[5] = "blue";
13    $rainbow[7] = "indigo";
14    $rainbow[8] = "violet";
15    for ($i=0; $i<=6; $i++) {
16        echo 'Rainbow['. $i.'] color is: '. $rainbow[$i]."<br />";
17    }
18 ?>
19 </body>
20 </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:

```
$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:

```
$teacher = $instructor["CNT 4714"];
```

- The variable `$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”:

```
$course = $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 an entry in the array with an index value of “Llewellyn” and will return nothing and the value of `$course` will be undefined. The example on the following page illustrates this.





```

1 <html>
2 <head>
3 <title>Creating An Associative Array - Incorrect Version</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     $instructor = array( "CNT 4714" => "Llewellyn",
9                         "CIS 3003" => "Eisler",
10                        "CIS 3360" => "Guha" );
11     echo 'The instructor of CNT 4714 is: '. $instructor["CNT 4714"]. "<br />";
12     echo 'The course taught by Eisler is: '. $instructor["Eisler"]. "<br />";
13
14     ?>
15 </body>
16 </html>

```

Incorrect Version

PHP Hypertext Preprocessor

Creating An Associative A x

localhost:8081/associative%20array%20-%20incorrect%20version.php

The instructor of CNT 4714 is: Llewellyn

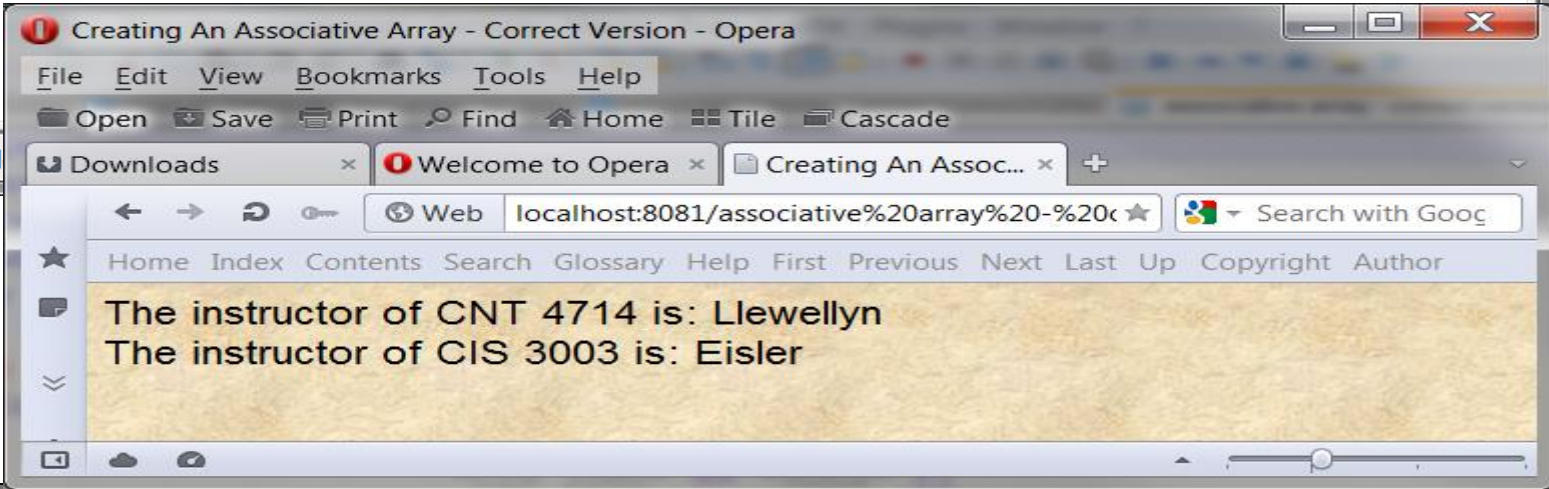
Notice: Undefined index: Eisler in C:\Apache24\htdocs\associative array - incorrect version.php on line 12

The course taught by Eisler is:



```
C:\Program Files (x86)\Apache Software Foundation\Apache2.2\htdocs\associative array - correct version.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
array definition V3.php array definition V4 misnumbering.php associative array - incorrect version.php associative array - correct version.php
1 <html>
2 <head>
3 <title>Creating An Associative Array - Correct Version</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=image1.jpg>
7 <?php
8     $instructor = array( "CNT 4714" => "Llewellyn",
9                         "CIS 3003" => "Eisler",
10                        "CIS 3360" => "Guha" );
11     echo 'The instructor of CNT 4714 is: '. $instructor["CNT 4714"]. "<br />";
12     echo 'The instructor of CIS 3003 is: '. $instructor["CIS 3003"]. "<br />";
13
14 ?>
15 </body>
16 </html>
```

Correct Version



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:

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

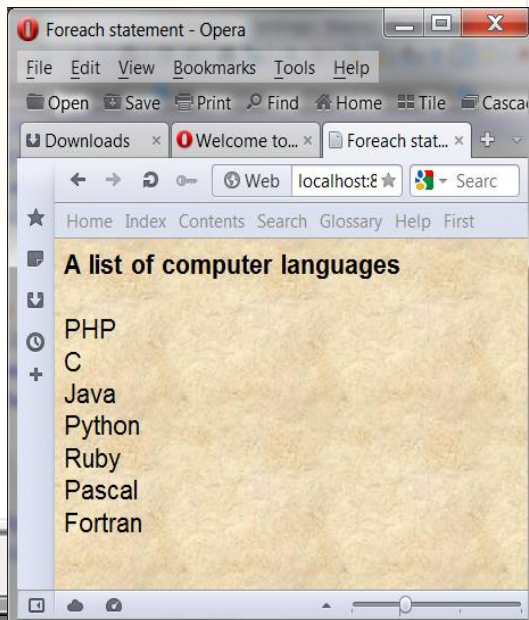
- The first variable inside the parentheses is the variable name representing the array and the second variable is automatically set to the next array item at each iteration of the loop. An example using a sequential array is shown on the next page and one with an associative array on the following page.



```

1 <html>
2 <head>
3 <title>Foreach statement</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     $languages = array( "PHP", "C", "Java", "Python", "Ruby", "Pascal", "Fortran");
9     echo "<b> A list of computer languages </b> <br /> <br />";
10    foreach ($languages as $item) {
11        echo $item . "<br />";
12    }
13 ?>
14 </body>
15 </html>

```

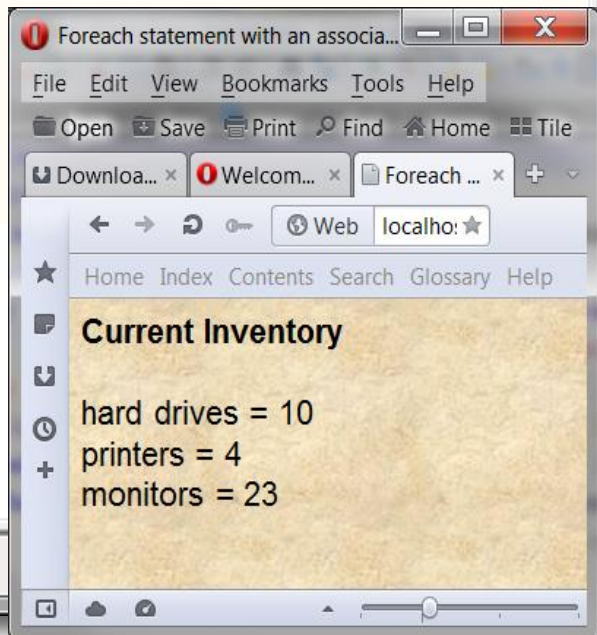




```

1 <html>
2 <head>
3 <title>Foreach statement with an associative array</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     $inventory = array( "hard drives" => 10, "printers" => 4, "monitors" => 23);
9     echo "<b> Current Inventory </b> <br /> <br />";
10    foreach ($inventory as $index => $item) {
11        echo $index . ' = ' . $item . "<br />";
12    }
13 ?>
14 </body>
15 </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.



File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?

associative array - correct version.php foreach statement.php foreach statement - associative array.php associative array operations.php

```

1 <html>
2 <head>
3 <title>Some operations on an associative array</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=image1.jpg>
7 <?php
8     $inventory = array( "hard drives" => 10, "printers" => 4, "monitors" => 23);
9     echo "<b><u>Current Inventory</u></b> <br /> <br />";
10    foreach ($inventory as $index => $item) {
11        echo $index . ' = ' . $item . "<br />";
12    }
13    $inventory["monitors"] = 5;
14    $inventory["keyboards"] = 12;
15    echo "<br /><b><u> Current Inventory</u></b> <br /> <br />";
16    foreach ($inventory as $index => $item) {
17        echo $index . ' = ' . $item . "<br />";
18    }
19    unset($inventory["printers"]);
20    echo "<br /><b><u> Current Inventory</u></b> <br /> <br />";
21    foreach ($inventory as $index => $item) {
22        echo $index . ' = ' . $item . "<br />";
23    }
24    ?>
25 </body>
26 </html>

```

PHP Hypertext Preprocessor file length : 857 lines : 26 Ln : 1 Col : 1 Sel : 0

Some operations on an a... -

File Edit View Bookmarks Tools Help

Open Save Print Find Home

Down... x Welco... x Some... x +

Web localho:*

Home Index Contents Search

Current Inventory

hard drives = 10
printers = 4
monitors = 23

Current Inventory

hard drives = 10
printers = 4
monitors = 5
keyboards = 12

Current Inventory

hard drives = 10
monitors = 5
keyboards = 12



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.



C:\Program Files (x86)\Apache Software Foundation\Apache2.2\htdocs\using the isset() function.php - Notepad++

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?

foreach statement.php | foreach statement - associative array.php | associative array operations.php | using the isset() function.php

```
1 <html>
2 <head>
3 <title>Using the isset() function on an associative array</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     $inventory = array( "hard drives" => 10, "printers" => 4, "monitors" => 23);
9     echo "<b><u>Current Inventory</u></b> <br /> <br />";
10    foreach ($inventory as $index => $item) {
11        echo $index . ' = ' . $item . "<br />";
12    }
13    echo "<br />";
14    if ( isset($inventory["monitors"]) ) {
15        echo "monitors is in the array. <br />";
16    }else { echo "monitors is not in the array. <br />";
17    }
18    if ( isset($inventory["keyboards"]) ) {
19        echo "keyboards is in the array. <br />";
20    }else { echo "keyboards is not in the array. <br />";
21    }
22    ?>
23 </body>
24 </html>
```

PHP Hypertext Preprocessor file | length: 773 | lines: 24 | Ln: 1 | Col: 1 | Sel: 0

Using the isset() function on an ...

File Edit View Bookmarks Tools Help

Open Save Print Find Home

Downlo... x Welcom... x Using th... x

Web localho: *

Home Index Contents Search Glossary

Current Inventory

hard drives = 10
printers = 4
monitors = 23

monitors is in the array.
keyboards is not in the array.



Using Associative Arrays In PHP

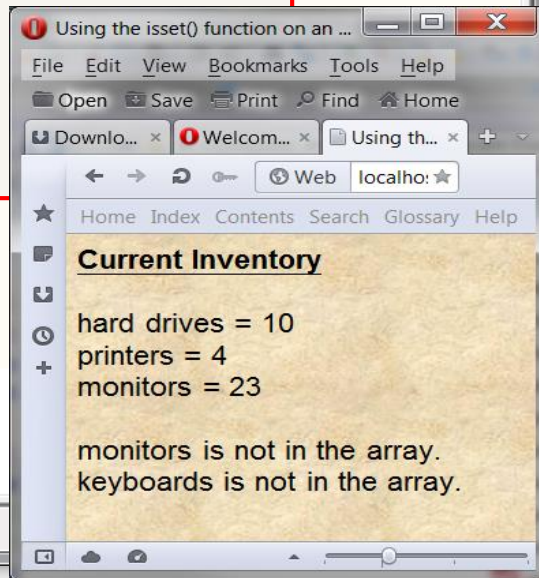
- As with many things in PHP, associative array indices are case-sensitive. Thus, in the previous example, if the call to the `isset()` function were passed the parameter “Monitors” instead of “monitors” it would return false instead of true.
- See next page.



```

1 <html>
2 <head>
3 <title>Using the isset() function on an associative array</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     $inventory = array( "hard drives" => 10, "printers" => 4, "monitors" => 23);
9     echo "<b><u>Current Inventory</u></b> <br /> <br />";
10    foreach ($inventory as $index => $item) {
11        echo $index . ' = ' . $item . "<br />";
12    }
13    echo "<br />";
14    //associative array indices are case-sensitive - monitors is in array
15    //Monitors is not
16    if ( isset($inventory["Monitors"]) ) {
17        echo "monitors is in the array. <br />";
18    }else { echo "monitors is not in the array. <br />";
19    }
20    if ( isset($inventory["keyboards"]) ) {
21        echo "keyboards is in the array. <br />";
22    }else { echo "keyboards is not in the array. <br />";
23    }
24    ?>
25 </body>
26 </html>

```



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.

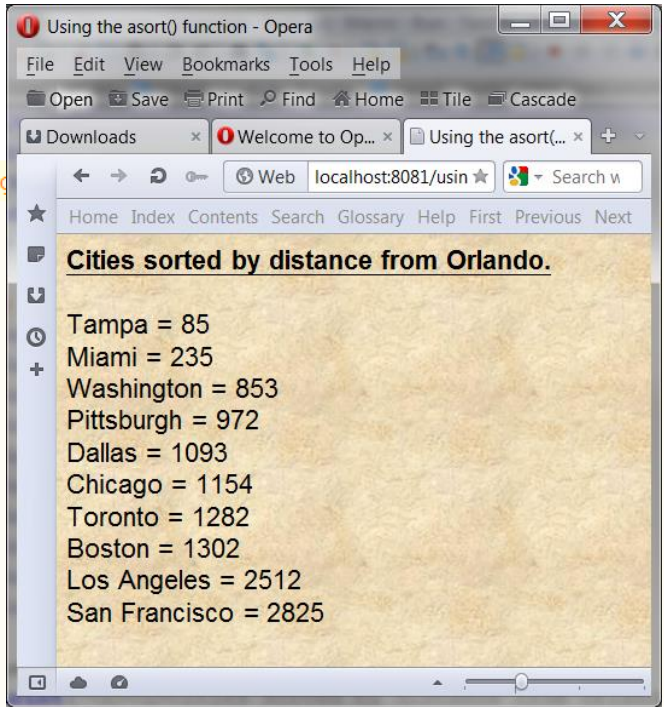


Using asort()

```

1 <html>
2 <head>
3     <title> Using the asort() function </title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg
7 <?php
8     $cities['Boston'] = 1302;
9     $cities['Dallas'] = 1093;
10    $cities['Toronto'] = 1282;
11    $cities['Chicago'] = 1154;
12    $cities['Los Angeles'] = 2512;
13    $cities['San Francisco'] = 2825;
14    $cities['Washington'] = 853;
15    $cities['Miami'] = 235;
16    $cities['Pittsburgh'] = 972;
17    $cities['Tampa'] = 85;
18
19    asort($cities);
20    print("<b><u>Cities sorted by distance from Orlando.</u></b> <br /><br />");
21    foreach ($cities as $index => $value) {
22        print (" $index = $value <br />");
23    }
24 >>
25 </body>
26 </html>

```

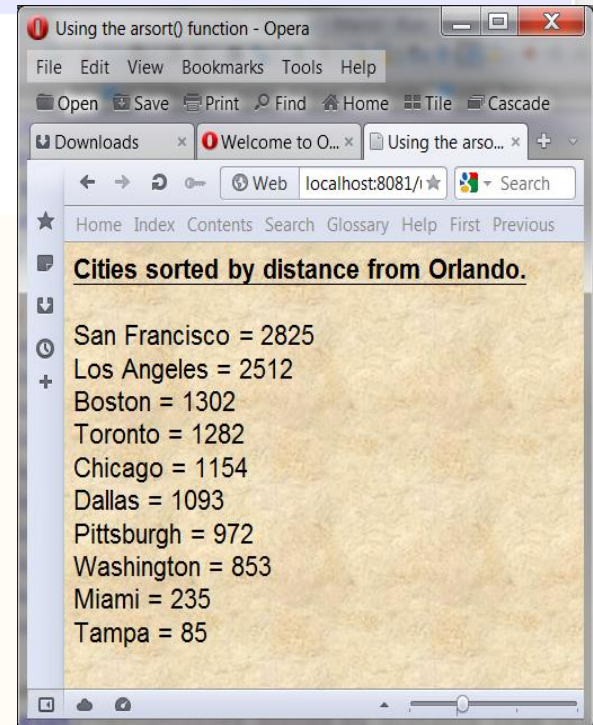


Using arsort()

```

1 <html>
2 <head>
3     <title> Using the arsort() function </title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagel.jpg>
7 <?php
8     $cities['Boston'] = 1302;
9     $cities['Dallas'] = 1093;
10    $cities['Toronto'] = 1282;
11    $cities['Chicago'] = 1154;
12    $cities['Los Angeles'] = 2512;
13    $cities['San Francisco'] = 2825;
14    $cities['Washington'] = 853;
15    $cities['Miami'] = 235;
16    $cities['Pittsburgh'] = 972;
17    $cities['Tampa'] = 85;
18
19    arsort($cities);
20    print("<b><u>Cities sorted by distance from Orlando.</u></b> <br /><br />");
21    foreach ($cities as $index => $value) {
22        print (" $index = $value <br />");
23    }
24    ?>
25 </body>
26 </html>

```

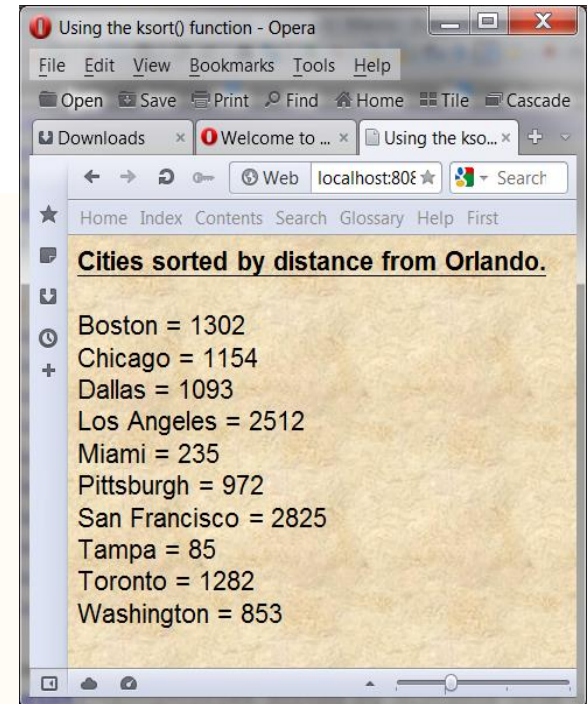


Using ksort()

```

1 <html>
2 <head>
3     <title> Using the ksort() function </title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=image1.jpg>
7 <?php
8     $cities['Boston'] = 1302;
9     $cities['Dallas'] = 1093;
10    $cities['Toronto'] = 1282;
11    $cities['Chicago'] = 1154;
12    $cities['Los Angeles'] = 2512;
13    $cities['San Francisco'] = 2825;
14    $cities['Washington']= 853;
15    $cities['Miami'] = 235;
16    $cities['Pittsburgh'] = 972;
17    $cities['Tampa'] = 85;
18
19    ksort($cities);
20    print("<b><u>Cities sorted by distance from Orlando.</u></b> <br /><br />");
21    foreach ($cities as $index => $value) {
22        print (" $index = $value <br />");
23    }
24    ?>
25 </body>
26 </html>

```

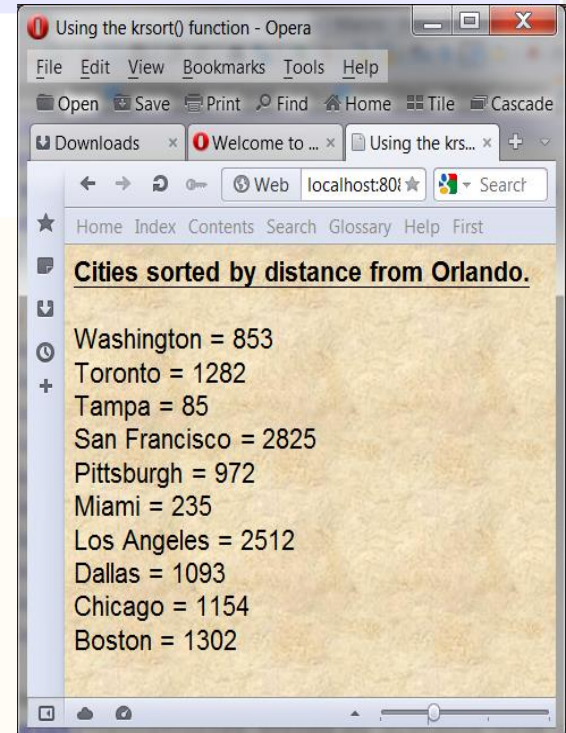


Using krsort()

```

1 <html>
2 <head>
3     <title> Using the krsort() function </title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=imagem1.jpg>
7 <?php
8     $cities['Boston'] = 1302;
9     $cities['Dallas'] = 1093;
10    $cities['Toronto'] = 1282;
11    $cities['Chicago'] = 1154;
12    $cities['Los Angeles'] = 2512;
13    $cities['San Francisco'] = 2825;
14    $cities['Washington'] = 853;
15    $cities['Miami'] = 235;
16    $cities['Pittsburgh'] = 972;
17    $cities['Tampa'] = 85;
18
19    krsort($cities);
20    print("<b><u>Cities sorted by distance from Orlando.</u></b> <br /><br />");
21    foreach ($cities as $index => $value) {
22        print (" $index = $value <br />");
23    }
24 ?>
25 </body>
26 </html>

```



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:

Part Number	Part Name	Count	Price
AC1000	Hammer	122	28.50
AC1001	Wrench	25	14.00
AC1002	Saw	18	25.00
AC1003	Screwdriver	34	4.50

- The example on the next page represents this data in a two-dimensional associative array.





```

1 <html>
2 <head>
3   <title> Hardware Inventory </title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6   background-color: #856363" background=imagem1.jpg>
7
8   <form action="show_inventory.php" method = "post">
9   <font size=4 color = "blue"><b><u>Mark's Hardware Inventory Information</u></b></font><br />
10  <br /><br />Select part number for more information </font> <br /><br />
11  <?php
12     $Inventory = array( 'AC1000', 'AC1001', 'AC1002', 'AC1003' );
13     foreach ( $Inventory as $item ) {
14         print "<input type=radio name=\"id\" value=$item > $item ";
15     }
16     print '<br /><br /><input type=submit value="Submit">&nbsp; &nbsp;';
17     print '<input type=reset value="Erase ">';
18     print '</form> </body></html>';
19  ?>
20 </body>
21 </html>
22

```

Front-end
Provides a set of radio buttons for user to select the part they'd like to see more information about.



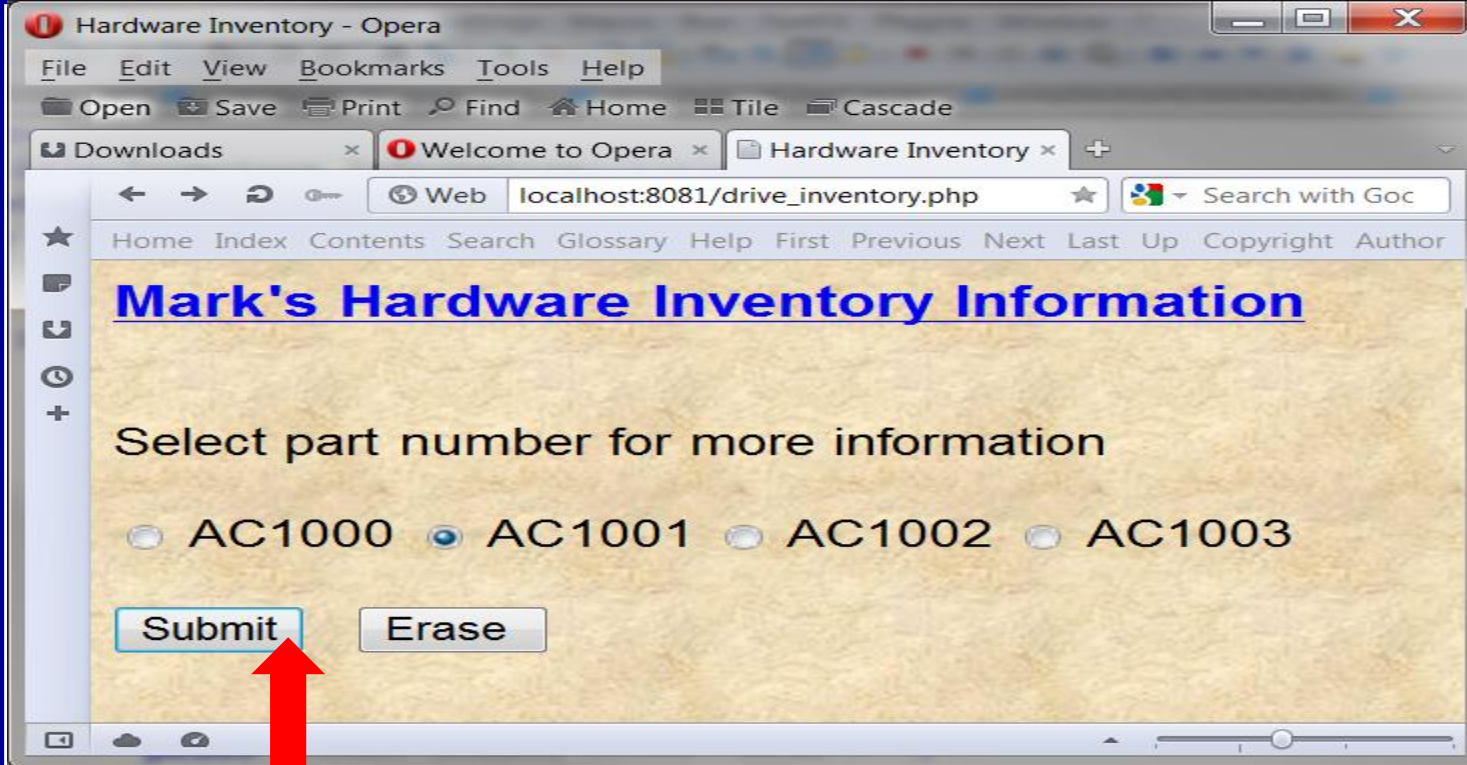
```

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?
using the asort() function.php using the arsort() function.php using the ksort() function.php using the krsort() function.php drive_inventory.php show_inventory.php
2 <head>
3     <title>Inventory Information</title>
4 </head>
5 <body style = "font-family: arial, sans-serif;
6     background-color: #856363" background=image1.jpg>
7 <?php
8     $inventory = array (
9         'AC1000'=>array('Part'=>'Hammer', 'Count'=>122,
10             'Price'=> 28.50 ),
11         'AC1001' => array('Part' =>'Wrench', 'Count' =>25,
12             'Price'=>14.00 ),
13         'AC1002'=>array('Part' =>'Handsaw', 'Count' =>18,
14             'Price'=>25.00 ),
15         'AC1003'=>array('Part' =>'Screwdrivers', 'Count'=>34,
16             'Price'=>4.50)
17     );
18     $id = $_POST["id"];
19     if (isset($inventory[$id])){
20         print '<font size=4 color="blue"> ';
21         print "<b><u>Inventory Information for Part $id </u></b></font><br /><br />";
22         print '<table border=1> <th> ID <th> Part <th> Count <th> Price ';
23         print "<tr> <td> $id </td>";
24             print "<td> {$inventory[$id]['Part']} </td>";
25             print "<td> {$inventory[$id]['Count']} </td>";
26             print("<td>");
27             printf( "%4.2f", $inventory[$id]['Price'] );
28             print("</td></tr>");
29         } else {
30             print "Illegal part ID = $id ";
31         }
32     }
33 </body>

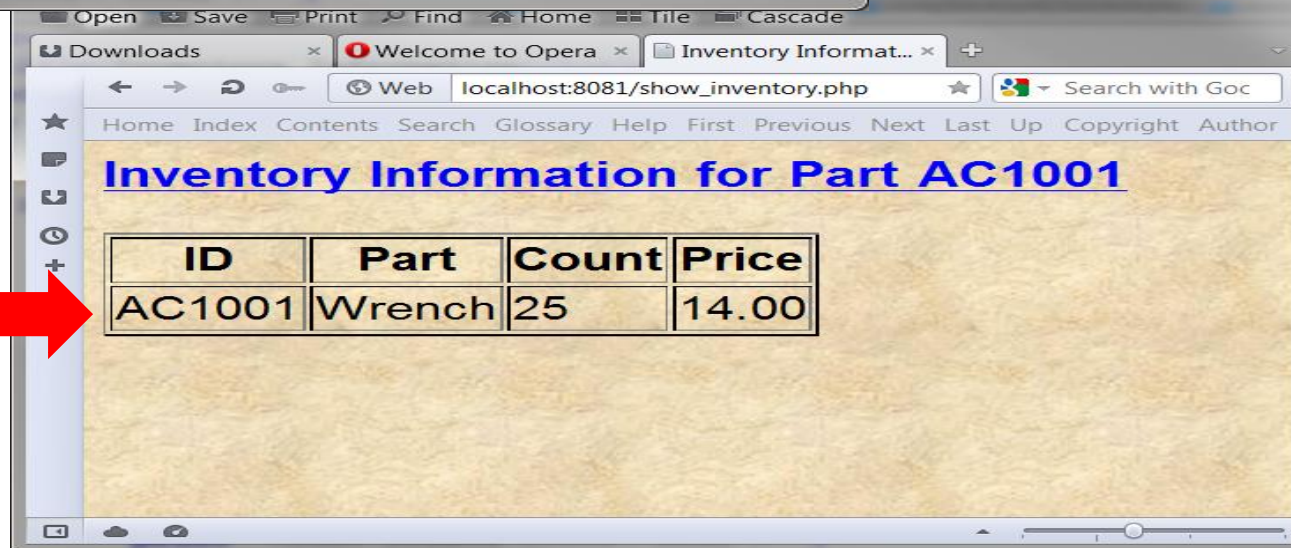
```

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.





1. User selects a part number



2. PHP script displays part details.

