

Functions and Control Structures

MIS 4530 - Dr. Garrett

A decorative graphic consisting of a solid teal horizontal bar at the top, followed by a white horizontal bar, and then three thin, parallel teal horizontal lines on the right side of the white bar.

Defining Functions

- **Functions** are groups of statements that you can execute as a single unit
- **Function definitions** are the lines of code that make up a function
- The syntax for defining a function is:

```
<?php
function name_of_function(parameters) {
    statements;
}
?>
```

Defining Functions (continued)

- Functions, like all PHP code, must be contained within `<?php ... ?>` tags
- A **parameter** is a variable that is used within a function
- Parameters are placed within the parentheses that follow the function name
- Functions do not have to contain parameters
- The set of curly braces (called **function braces**) contain the function statements

Defining Functions (continued)

- **Function statements** do the actual work of the function and must be contained within the function braces

```
function printCompanyName ($Company1, $Company2, $Company3)
{
    echo "<p>$Company1</p>";
    echo "<p>$Company2</p>";
    echo "<p>$Company3</p>";
}
```

Calling Functions

```
function printCompanyName ($CompanyName) {  
    echo "<p>$CompanyName</p>";  
}  
printCompanyName("Course Technology");
```

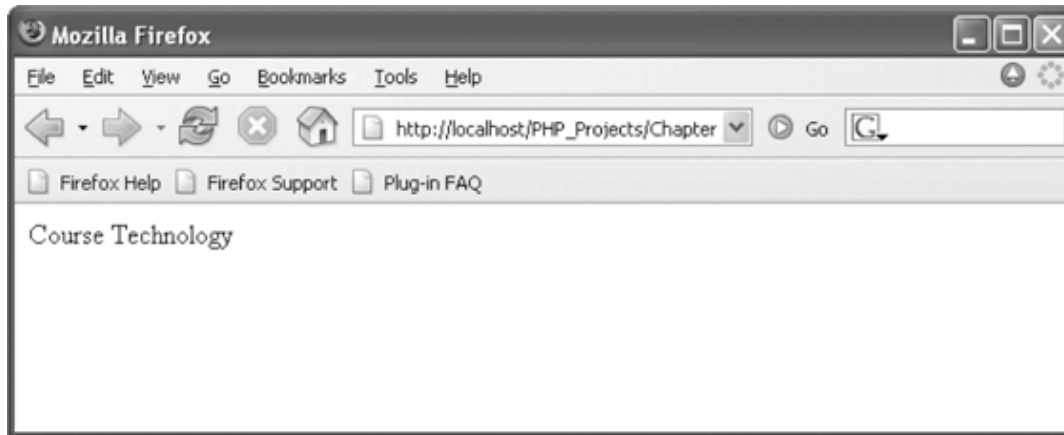


Figure 4-1 Output of a call to a custom function

Returning Values

- A **return statement** is a statement that returns a value to the statement that called the function
- A function does not necessarily have to return a value

```
function averageNumbers($a, $b, $c) {  
    $SumOfNumbers = $a + $b + $c;  
    $Result = $SumOfNumbers / 3;  
    Return $Result;  
}
```

Understanding Variable Scope

- **Variable scope** is where in your program a declared variable can be used
- A variable's scope can be either global or local
- A **global variable** is one that is declared outside a function and is available to all parts of your program
- A **local variable** is declared inside a function and is only available within the function in which it is declared

Using Autoglobals

- PHP includes various predefined global arrays, called **autoglobals** or **superglobals**
- Autoglobals contain client, server, and environment information that you can use in your scripts
- Autoglobals are **associative arrays** – arrays whose elements are referred to with an alphanumeric key instead of an index number

Using Autoglobals (continued)

| Array | Description |
|-------------------------|--|
| <code>\$_COOKIE</code> | An array of values passed to the current script as HTTP cookies |
| <code>\$_ENV</code> | An array of environment information |
| <code>\$_FILES</code> | An array of information about uploaded files |
| <code>\$_GET</code> | An array of values from a form submitted with the GET method |
| <code>\$_POST</code> | An array of values from a form submitted with the POST method |
| <code>\$_REQUEST</code> | An array of all the elements found in the <code>\$_COOKIE</code> , <code>\$_GET</code> , and <code>\$_POST</code> arrays |
| <code>\$_SERVER</code> | An array of information about the Web server that served the current script |
| <code>\$_SESSION</code> | An array of session variables that are available to the current script |
| <code>\$GLOBALS</code> | An array of references to all variables that are defined with global scope |

Using Autoglobals (continued)

- Use the `global` keyword to declare a global variable within the scope of a function
- Use the `$GLOBALS` autoglobal to refer to the global version of a variable from inside a function
- `$_GET` is the default method for submitting a form
- `$_GET` and `$_POST` allow you to access the values of forms that are submitted to a PHP script

Using Autoglobals (continued)

- `$_GET` appends form data as one long string to the URL specified by the action attribute
- `$_POST` sends form data as a transmission separate from the URL specified by the action attribute

Making Decisions

- **Decision making** or **flow control** is the process of determining the order in which statements execute in a program
- The special types of PHP statements used for making decisions are called **decision-making statements** or **decision-making structures**

`if` Statements

- Used to execute specific programming code if the evaluation of a conditional expression returns a value of **true**
- The syntax for a simple `if` statement is:

```
if (conditional expression)  
    statement;
```

`if` Statements (continued)

- Contains three parts:
 - the keyword `if`
 - a conditional expression enclosed within parentheses
 - the executable statements
- A **command block** is a group of statements contained within a set of braces
- Each command block must have an opening brace (`{`) and a closing brace (`}`)

if Statements (continued)

```
$ExampleVar = 5;
if ($ExampleVar == 5) { // CONDITION EVALUATES TO 'TRUE'
    echo "<p>The condition evaluates to true.</p>";
    echo '<p>$ExampleVar is equal to ', "$ExampleVar.</p>";
    echo "<p>Each of these lines will be printed.</p>";
}
echo "<p>This statement always executes after the if
statement.</p>";
```

`if...else` Statements

- An `if` statement that includes an `else` clause is called an **`if...else statement`**
- An `else` clause executes when the condition in an `if...else` statement evaluates to **`false`**
- The syntax for an `if...else` statement is:

```
if (conditional expression)
    statement;
else
    statement;
```

`if...else` Statements (continued)

- An `if` statement can be constructed without the `else` clause
- The `else` clause can only be used with an `if` statement

```
$Today = "Tuesday";  
if ($Today == "Monday")  
    echo "<p>Today is Monday</p>";  
else  
    echo "<p>Today is not Monday</p>";
```

Nested `if` and `if...else` Statements

- When one decision-making statement is contained within another decision-making statement, they are referred to as nested **decision-making structures**

```
if ($_GET["SalesTotal"] > 50)
    if ($_GET["SalesTotal"] < 100)
        echo "<p>The sales total is between 50 and 100.</p>";
```

switch Statements

- Controls program flow by executing a specific set of statements depending on the value of an expression
- Compares the value of an expression to a value contained within a special statement called a **case label**
- A **case label** is a specific value that contains one or more statements that execute if the value of the case label matches the value of the switch statement's expression

switch Statements (continued)

- Consists of the following components:
 - The `switch` keyword
 - An expression
 - An opening brace
 - A case label
 - The executable statements
 - The `break` keyword
 - A default label
 - A closing brace

switch Statements (continued)

- The syntax for the `switch` statement is:

```
Switch (expression) {  
    case label:  
        statement (s);  
        break;  
    case label:  
        statement (s);  
        break;  
    ...  
    default:  
        statement (s);  
}
```

switch Statements (continued)

- A `case` label consists of:
 - The keyword `case`
 - A literal value or variable name
 - A colon
- A `case` label can be followed by a single statement or multiple statements
- Multiple statements for a `case` label do not need to be enclosed within a command block

`switch` Statements (continued)

- The **default label** contains statements that execute when the value returned by the `switch` statement expression does not match a `case` label
- A `default` label consists of the keyword `default` followed by a colon

Repeating Code

- A **loop statement** is a control structure that repeatedly executes a statement or a series of statements while a specific condition is true or until a specific condition becomes true
- There are four types of loop statements:
 - `while` statements
 - `do...while` statements
 - `for` statements
 - `foreach` statements

while Statements

- Repeats a statement or a series of statements as long as a given conditional expression evaluates to true
- The syntax for the `while` statement is:

```
while (conditional expression) {  
    statement(s);  
}
```
- As long as the conditional expression evaluates to true, the statement or command block that follows executes repeatedly

while Statements (continued)

- Each repetition of a looping statement is called an **iteration**
- A while statement keeps repeating until its conditional expression evaluates to false
- A **counter** is a variable that increments or decrements with each iteration of a loop statement

while Statements (continued)

```
$Count = 1;
while ($Count <= 5) {
    echo "$Count<br />";
    ++$Count;
}
echo "<p>You have printed 5 numbers.</p>";
```

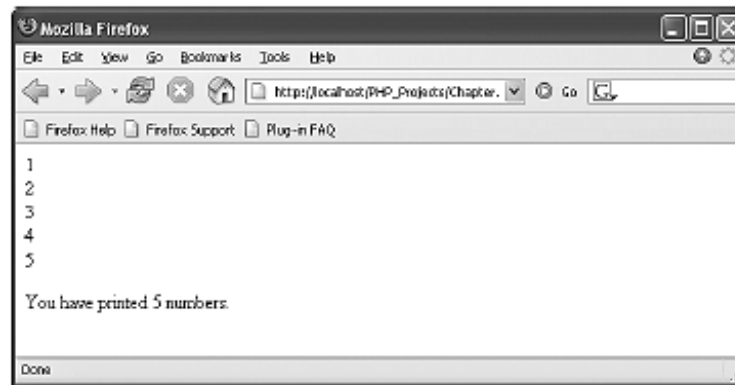


Figure 4-7 Output of a while statement using an increment operator

while Statements (continued)

```
$Count = 10;  
while ($Count > 0) {  
    echo "$Count<br />";  
    --$Count;  
}  
echo "<p>We have liftoff.</p>";
```

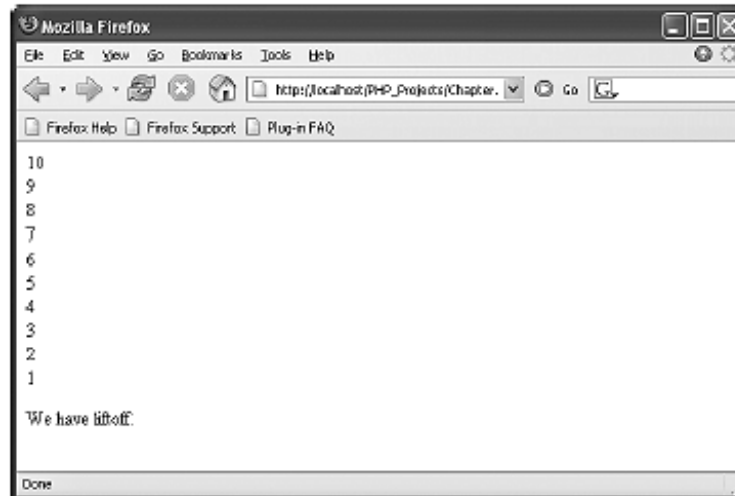


Figure 4-8 Output of a `while` statement using a decrement operator

while Statements (continued)

```
$Count = 1;
while ($Count <= 100) {
    echo "$Count<br />";
    $Count *= 2;
}
```

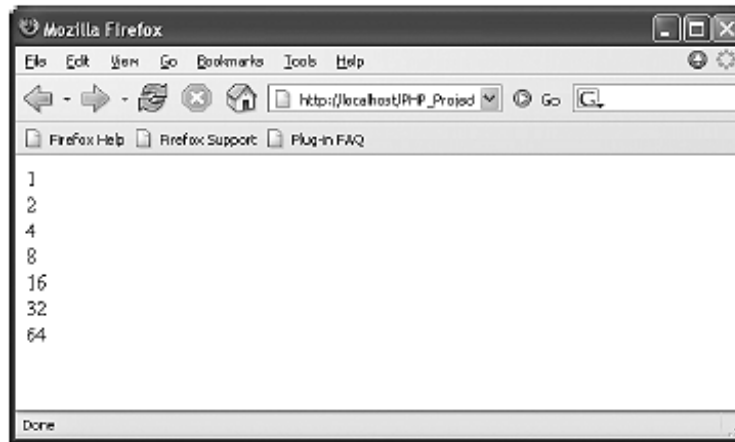


Figure 4-9 Output of a `while` statement using the assignment operator `*=`

while Statements (continued)

- In an **infinite loop**, a loop statement never ends because its conditional expression is never false

```
$Count = 1;
while ($Count <= 10) {
    echo "The number is $Count";
}
```

do...while Statements

- Executes a statement or statements once, then repeats the execution as long as a given conditional expression evaluates to true
- The syntax for the do...while statement is:

```
do {  
    statement(s);  
} while (conditional expression);
```

do...while Statements (continued)

- do...while statements always execute once, before a conditional expression is evaluated

```
$Count = 2;  
do {  
    echo "<p>The count is equal to $Count</p>";  
    ++$Count;  
} while ($Count < 2);
```

do...while Statements (continued)

```
$DaysOfWeek = array("Monday", "Tuesday", "Wednesday", "Thursday",  
"Friday", "Saturday", "Sunday");  
$Count = 0;  
do {  
    echo $DaysOfWeek[$Count], "<br />";  
    ++$Count;  
} while ($Count < 7);
```

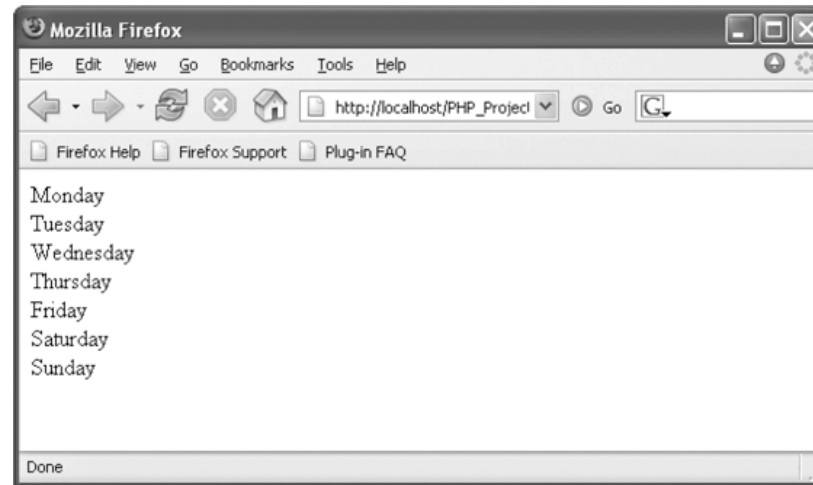


Figure 4-11 Output of days of week script in Web browser

for Statements

- Used for repeating a statement or a series of statements as long as a given conditional expression evaluates to true
- If a conditional expression within the `for` statement evaluates to true, the `for` statement executes and continues to execute repeatedly until the conditional expression evaluates to false

for Statements (continued)

- Can also include code that initializes a counter and changes its value with each iteration
- The syntax of the `for` statement is:

```
for (counter declaration and initialization; condition;  
      update statement) {  
    statement(s);  
}
```

for Statements (continued)

```
$FastFoods = array("pizza", "burgers", "french fries", "tacos",  
"fried chicken");  
for ($Count = 0; $Count < 5; ++$Count) {  
    echo $FastFoods[$Count], "<br />";  
}
```

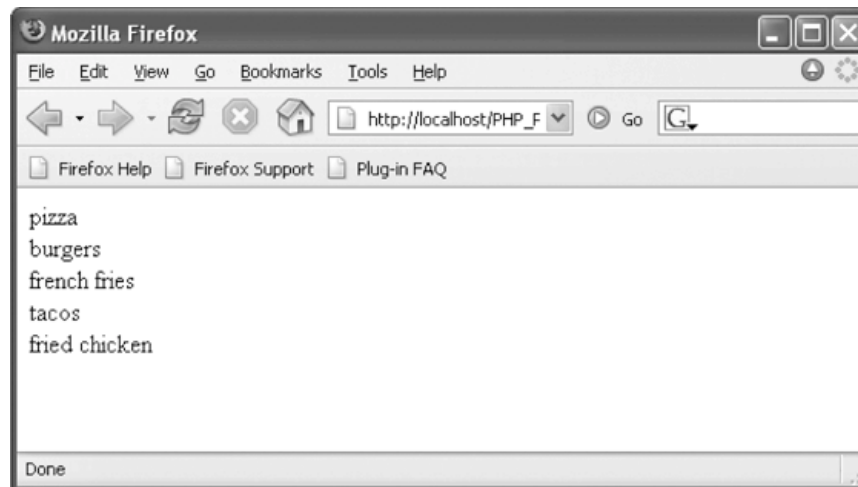


Figure 4-12 Output of fast-foods script

foreach Statements

- Used to iterate or loop through the elements in an array
- Does not require a counter; instead, you specify an array expression within a set of parentheses following the `foreach` keyword
- The syntax for the `foreach` statement is:

```
foreach ($array_name as $variable_name) {  
statements;  
}
```