SOFT7008
Server Side Web Development

Functions and Control Structures

Méabh O'Connor
meabh.oconnor@cit.ie

User Defined 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 *your own function* is:

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

Defining Functions

- Functions, like all PHP code, must be contained within `<?php ... ?>` tags
- A **parameter** is a variable that is declared in the function declaration
- 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

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

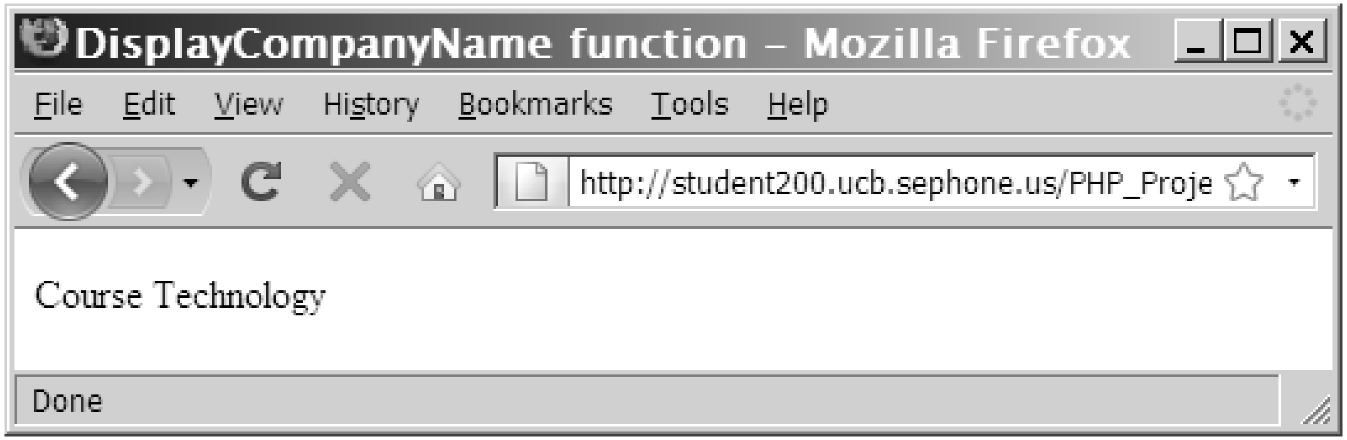
```
function displayCompanyName($Company1, $Company2,
  $Company3)
{
   echo "<p>$Company1</p>";
   echo "<p>$Company2</p>";
   echo "<p>$Company3</p>";
}
```

Calling Functions

- A function definition does not execute automatically.
- Creating a function definition only names the function, specifies its parameters and organizes the statements it will execute.
- A function must be **called** to execute it.
- When you pass arguments to a function – the value of each argument is assigned to the value of the corresponding formal parameter in the function definition.

Calling Functions

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


DisplayCompanyName function – Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://student200.ucb.sephone.us/PHP_Proje
Done

Course Technology

Figure 2-1 Output of a call to a user defined function

Returning Values

- A **return statement** returns a value to the statement that called the function
- Not all functions return values

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

Returning Values

- You can pass a function parameter by **value** or by **reference**
- A function parameter that is passed by value is a local copy of the variable.
- **By default** – variables are passed **by value** in PHP.
- A function parameter that is passed by reference is a reference to the original variable.
- To pass by reference – insert an ampersand (&) before the dollar sign of the parameter name in the function declaration.
- Do not use the ampersand when specifying the arguments at the function call.

Returning Values

Sample code:

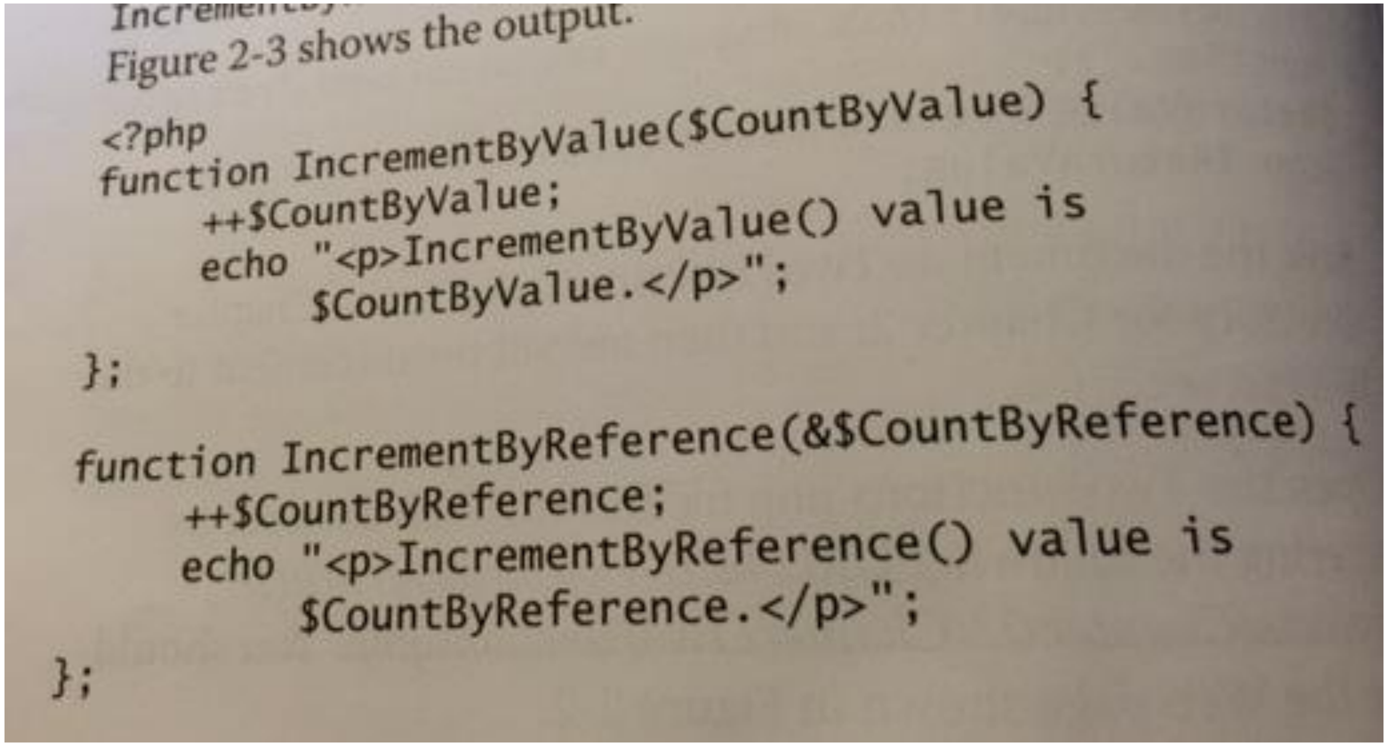


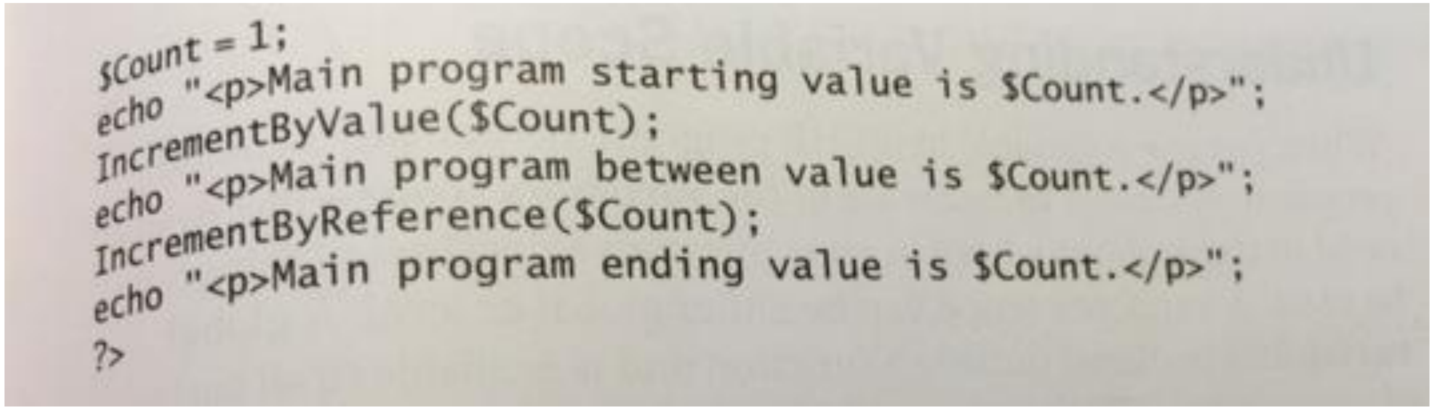
Figure 2-3 shows the output.

```
<?php
function IncrementByValue($CountByValue) {
      ++$CountByValue;
      echo "<p>IncrementByValue() value is
            $CountByValue.</p>";
};

function IncrementByReference(&$CountByReference) {
      ++$CountByReference;
      echo "<p>IncrementByReference() value is
            $CountByReference.</p>";
};
```

Returning Values

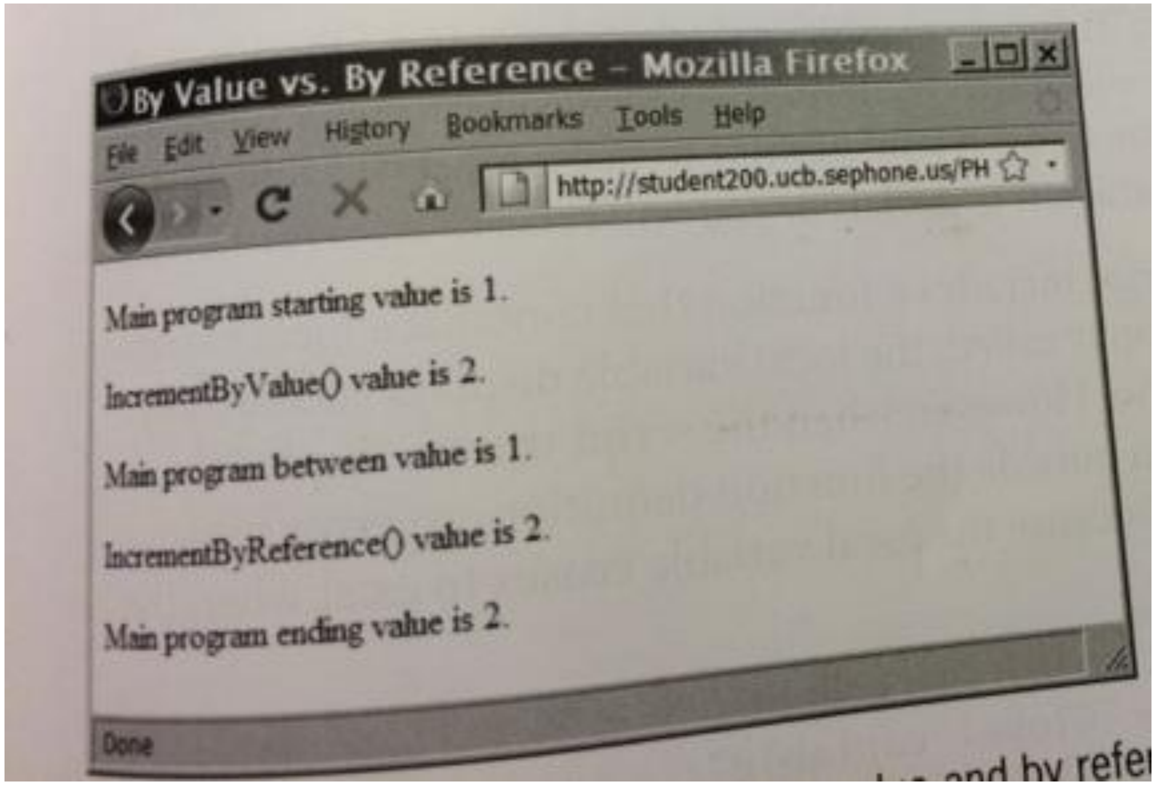
Sample code:



```
$Count = 1;
echo "<p>Main program starting value is $Count.</p>";
IncrementByValue($Count);
echo "<p>Main program between value is $Count.</p>";
IncrementByReference($Count);
echo "<p>Main program ending value is $Count.</p>";
?>
```

Returning Values

<u>Sample code:</u>


By Value vs. By Reference - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://student200.ucb.sephone.us/PH
Done

```
Main program starting value is 1.

IncrementByValue() value is 2.

Main program between value is 1.

IncrementByReference() value is 2.

Main program ending value is 2.
```

Understanding Variable Scope

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

The `global` Keyword

- In many programming languages – *global variables are automatically availabl*e to all parts of your program – including functions.

- **BUT this is *not* the case in PHP**

- In PHP, you **must** declare a global variable with the `global` keyword *inside* a function definition to make the variable available *within the scope of that function*.

The `global` Keyword

```
<?php
$globalVariable = "Global variable";

function scopeExample()
{
  $localVariable =“<p>$Local Variable</p>”;
  echo "<p>$localVariable</p>";
  // generates an error message
  echo "<p>$globalVariable</p>";

}
scopeExample();// call function
echo “<p>$globalVariable</p>”;
// generates error message
echo "<p>$localVariable</p>";
?>
```

The `global` Keyword

```
<?php
$globalVariable = "Global variable";
function scopeExample()
{
   global globalVariable; // no need to intialise
   $localVariable =“<p>$Local Variable</p>”
   echo "<p>$localVariable</p>";
   // does not generate an error message
   echo "<p>$globalVariable</p>";
}
scopeExample();
echo “<p>$globalVariable</p>”;
// generates error message
echo "<p>$localVariable</p>";
?>
```

Making Decisions

- When you write a computer program – regardless of the programming language – you often need to *execute different sets of statements depending on some predetermined criteria*.
  - e.g. – execute one set of code in the morning and another set at night.
- **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

- Contains three parts:
  - the keyword `if`
  - a conditional expression enclosed within parentheses
  - the executable statement(s)

- 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

```
…
$ExampleVar = 5;

if ($ExampleVar == 5)
{
     // condition evaluates to 'TRUE'
     echo " <p>The condition evaluates to true.</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 ($SalesTotal >= 50)
    if ($SalesTotal <= 100)
        echo " <p>The sales total is between
              50 and 100, inclusive.</p> ";
```

`switch` Statements

- Control program flow by executing a specific set of statements depending on the **value** of an expression
- Compare 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

- The syntax for the `switch` statement is:

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

`switch` Statements

- Consist of the following components:
  - The `switch` keyword
  - An expression
  - An opening brace
  - One or more `case` labels
  - The executable statements
  - The `break` keyword
  - A `default` label
  - A closing brace

`switch` Statements

- 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
- Different data types possible for each label.
- Multiple statements for a `case` label do not need to be enclosed within a command block

`switch` Statements

- 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 (:)

switch Statements

```
…
function city_location($AmericanCity)
{
     switch ($AmericanCity)
     {
       case “Boston”:
             return “Massechusetts”;
             break;
       case “Chicago”:
             return “Illinois”;
             break;
       case “Los Angeles”:
             return “California”;
             break;
```

switch Statements

```
        case “Miami”:
              return “Florida”;
              break;
        case “New York”:
              return “New York”;
               break;
        default:
              return “United States”;
              break;
      }
}
//call function
echo “<p>”, city_location(”Boston”), “</p>”;
```

- ...

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

- Tests the condition *prior* to executing the series of statements at each iteration of the loop

- The syntax for the `while` statement is:

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

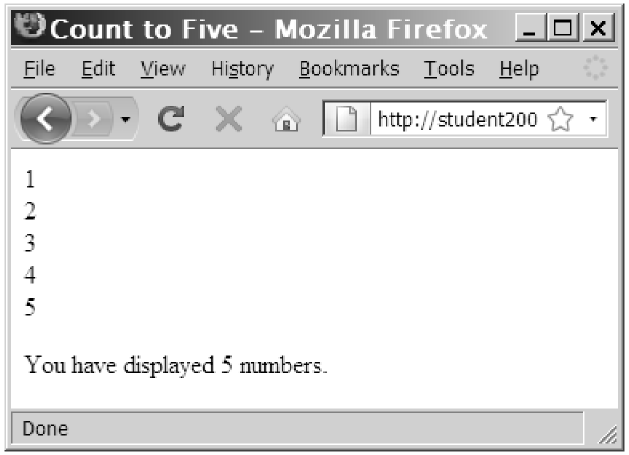
- As long as the conditional expression evaluates to `TRUE`, the statement or command block that follows executes repeatedly

`while` Statements

- 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

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


Count to Five – Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://student200
Done

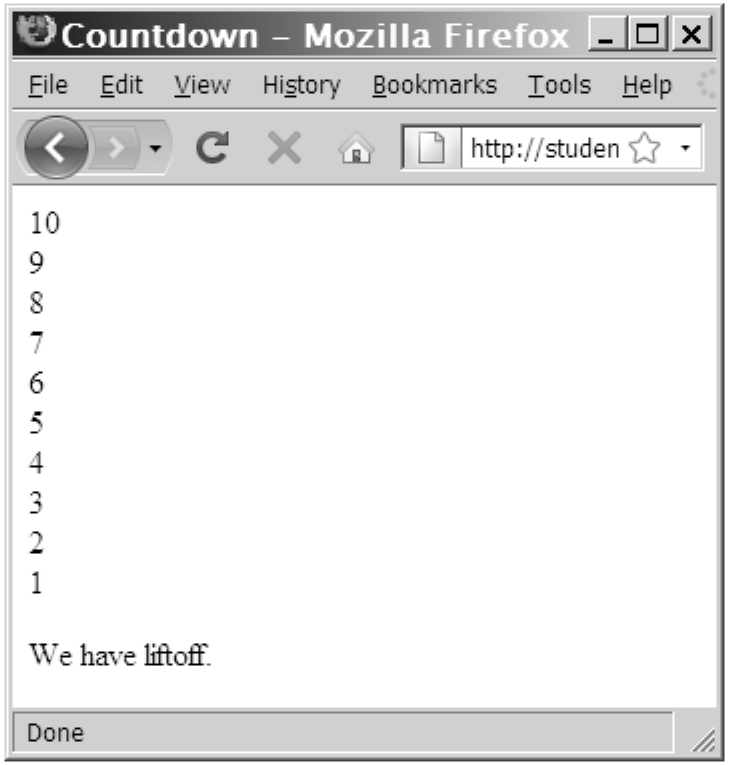
```
1
2
3
4
5

You have displayed 5 numbers.
```

Figure 2-5 Output of a `while` statement using an increment operator

`while` Statements

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


Countdown – Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://studen
Done

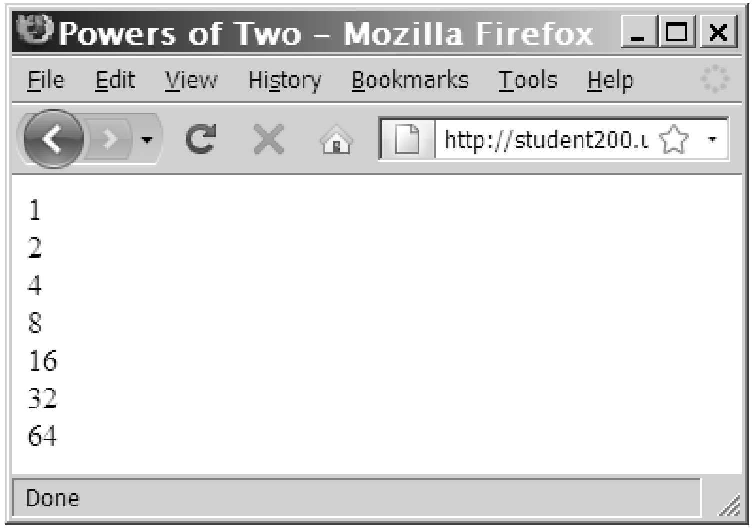
```
10
9
8
7
6
5
4
3
2
1

We have liftoff.
```

Figure 2-6 Output of a `while` statement using a decrement operator

`while` Statements (continued)

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


Powers of Two – Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://student200.u
Done

```
1
2
4
8
16
32
64
```

Figure 2-7 Output of a `while` statement using the assignment operator *=

`do...while` Statements

- Test the condition ***after*** executing a series of statements 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);
```

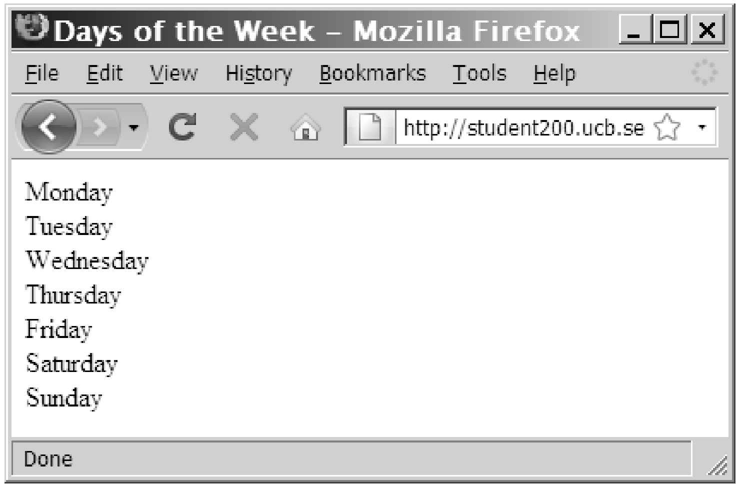
do...while Statements (continued)

- `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

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


Days of the Week – Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://student200.ucb.se
Done

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

Figure 2-9 Output of days of week script in Web browser

`for` Statements

- Combine the initialize, conditional evaluation, and update portions of a loop into a single statement
- Repeat a statement or a series of statements as long as a given conditional expression evaluates to `TRUE`
- If the conditional expression evaluates to `TRUE`, the `for` statement executes and continues to execute repeatedly until the conditional expression evaluates to `FALSE`

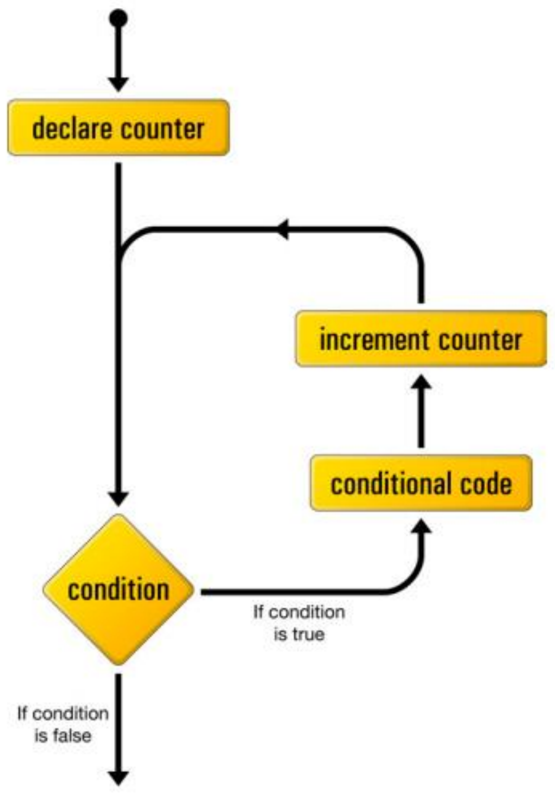
`for` Statements

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

- the **declare counter** statement is executed ***once*** at the start of the loop
- the **condition** statement is checked **each time** ***before*** the statements in the loop body are executed
- the **increment counter** statement is executed **each time** through the loop ***after*** the statements in the body.

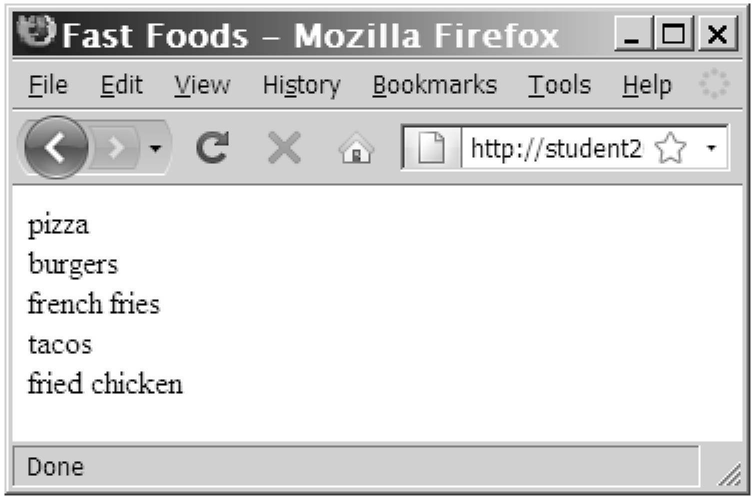
for Statements

- The logical path of execution for a **for** loop is as follows:



for Statements

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


Fast Foods – Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://student2
Done

```
pizza
burgers
french fries
tacos
fried chicken
```

Figure 2-10 Output of fast foods script

`foreach` Statements

- Used to iterate or loop through the elements in an **array**
- Do not require a counter; instead, you specify an array expression within a set of parentheses following the `foreach` keyword
- With each loop – a `foreach` statement moves to the next element in an array.
- During each iteration – a `foreach` statement assigns the value of the current array element to the *`$variable_name`* argument specified in the array expression.

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

foreach Statements

```
$DaysOfWeek = array("Monday", "Tuesday",
  "Wednesday", "Thursday", "Friday",
  "Saturday", "Sunday");

foreach ($DaysOfWeek as $Day)
{
      echo "<p>$Day</p>";
}
```

`foreach` Statements

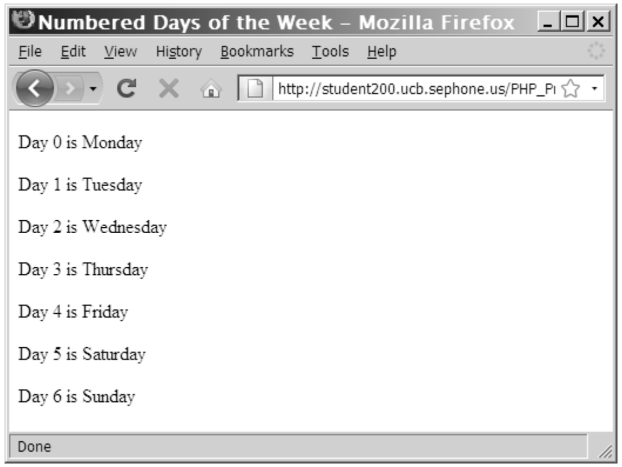
- A more advanced form of the `foreach` statement allows you to retrieve both the index (or key) and the value of each array element.

```
foreach($array_name as $index_name => $variable_name)
{
    statement(s);
}
```

- In this case – for each iteration the index of the current array element is stored in the `$index_name` variable.

foreach Statements

```
$DaysofWeek = array("Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday",
"Sunday");
foreach ($DaysOfWeek as $DayNumber => $Day) {
  echo "<p>Day $DayNumber is $Day</p>";
}
```


Numbered Days of the Week – Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://student200.ucb.sephone.us/PHP_P
Done

```
Day 0 is Monday

Day 1 is Tuesday

Day 2 is Wednesday

Day 3 is Thursday

Day 4 is Friday

Day 5 is Saturday

Day 6 is Sunday
```

Figure 2-11 Output of the foreach script with index values

Summary

- The lines that make up a function are called the **function definition**
- A function parameter that is passed by **value** is a local copy of the variable
- A function parameter that is passed by **reference** is a reference to the original variable
- A **global variable** is declared outside a function and is available to all parts of your program

Summary (continued)

- A **local variable** is declared inside a function and is only available within the function in which it is declared
- The process of determining the order in which statements execute in a program is called **decision making** or **flow control**
- The `if` statement is used to execute specific programming code if the evaluation of a conditional expression returns a value of `TRUE`

Summary (continued)

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

Summary (continued)

- The **switch statement** controls program flow by executing a specific set of statements, depending on the value of an expression
- 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`
- A `while` statement tests the condition prior to executing the series of statements at each iteration of the loop

Summary (continued)

- The `do...while` statement tests the condition after executing a series of statements
- The `for` statement combines the initialize, conditional evaluation, and update portions of a loop into a single statement
- The `foreach` statement is used to iterate or loop through the elements in an array

Summary (continued)

- The `include`, `require`, `include_once`, and `require_once` statements insert the contents of an external file at the location of the statement