SWINBURNE UNIVERSITY OF TECHNOLOGY

Advanced Web Development: Functions and Control Structures

Week 3

Outline

Functions and Control Structures

- Study how to use functions to organise your PHP code
- Learn about variable scope and autoglobal variables
- Use if statements, `if...else` statements, and switch statements
- Use nested control structures
- Use `while` statements, `do...while` statements, for, and `foreach` statements to repeatedly execute code

- Reading: Textbook Chapter 2

SWINBURNE UNIVERSITY OF TECHNOLOGY

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");
```

Course Technology

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) {
    $sum = $a + $b + $c;
    $result = $sum / 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

Understanding Variable Scope (Cont.)

```
<?php
  // all functions usually grouped together
  // in one location
  function testScope() {
        $localVariable = "<p>Local variable</p>";
        echo "<p>$localVariable</p>";
                    // prints successfully
  }
  $globalVariable = "Global variable";
  testScope();
  echo "<p>$globalVariable</p>";
  echo "<p>$localVariable</p>"; // error message
?>
```

The `global` Keyword

- With many programming languages, global variables are automatically available to all parts of your program including functions.
- In PHP however, we need to use the `global` keyword to declare a global variable in a function where you would like to use it.

```
<?php
  function testScope() {
        global $name;
        echo "<p>$globalVariable</p>";
  }
  $name = "Global variable";
  testScope();
```

- If no “global” above, an error message will be printed out.

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

See ***Predefined Variables, Superglobals and examples:***
http://php.net/manual/en/reserved.variables.php

Using Autoglobals (continued)

PHP autoglobals

Array	Description
$_COOKIE	An array of values passed to the current script as HTTP cookies
$_ENV	An array of environment information
$_FILES	An array of information about uploaded files
$_GET	An array of values from a form submitted with the GET method
$_POST	An array of values from a form submitted with the POST method
$_REQUEST	An array of all elements found in the $_COOKIE, $_GET, and $_POST arrays
$_SERVER	An array of information about the Web server that served the current script
$_SESSION	An array of session variables that are available to the current script
$_GLOBALS	An array of references to all variables available in the 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
- `$_GET` appends form data as one long string to the URL specified by the action attribute
- `$_POST` sends form data as a transmission separate from the URL specified by the action attribute

Using Autoglobals (continued)

```
echo "This script was executed with the
  following server software: ",
  $_SERVER["SERVER_SOFTWARE"], "<br>";

echo "This script was executed with the
  following server protocol: ",
  $_SERVER["SERVER_PROTOCOL"], "<br>";

<form method=”POST”>

<input type=“text” name=“address”>


</form>

echo $_POST["name"];

echo $_POST["address"];
```

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` Statement

- 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;
```

- Contains three parts:
  - the keyword `if`
  - a conditional expression enclosed within parentheses
  - the executable statements

`if` Statement (continued)

- 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 **}**

```
$exampleVar = 5;
if ($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 if.</p>";
```

`if...else` Statement

- 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` Statement (continued)

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

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

Note: Single statement within `if` ... `else`, therefore no braces needed

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` Statement

- 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` Statement (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` Statement (continued)

- The syntax for the `switch` statement is:

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

```php
<?php
$fruit = "orange";

switch ($fruit) {
    case "orange":
        echo "$10 please";
        break;
    case "banana":
        echo "$5 please";
        break;
    case "apple":
    case "grape":
        echo "$7 please";
        break;
    default:
        echo "$0 to be hungry";
}
?>
```

`switch` Statement (continued)

- A `case` label consists of:
  - The keyword `case`
  - A literal value or variable name (e.g. “Boston”, 75, $Var)
  - 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
- 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` Statement

- 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);
}
```

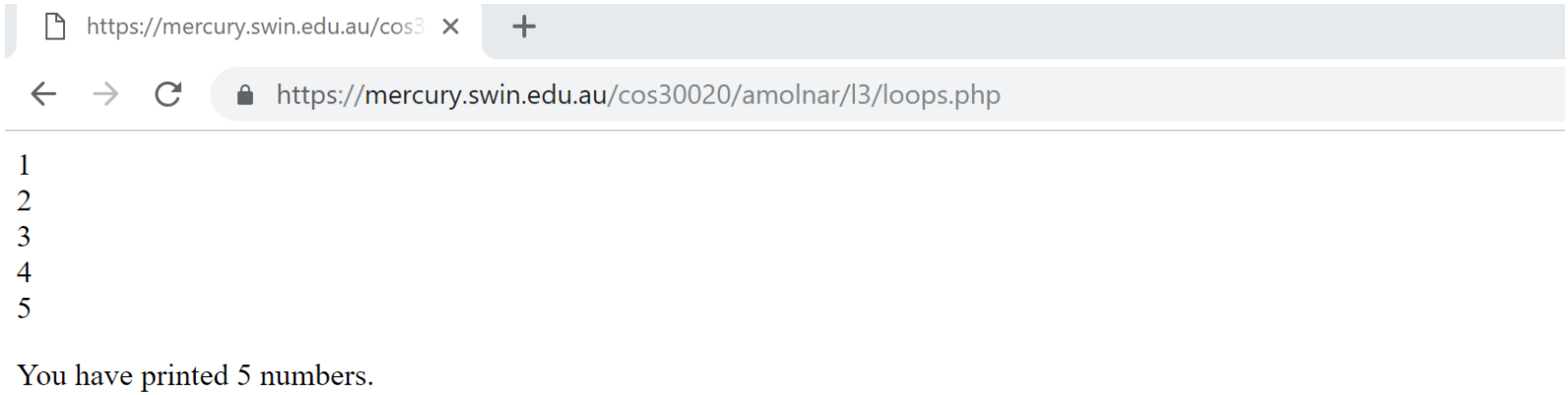
- As long as the conditional expression evaluates to true, the statement or command block that follows executes repeatedly

`while` Statement (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` Statement (continued)

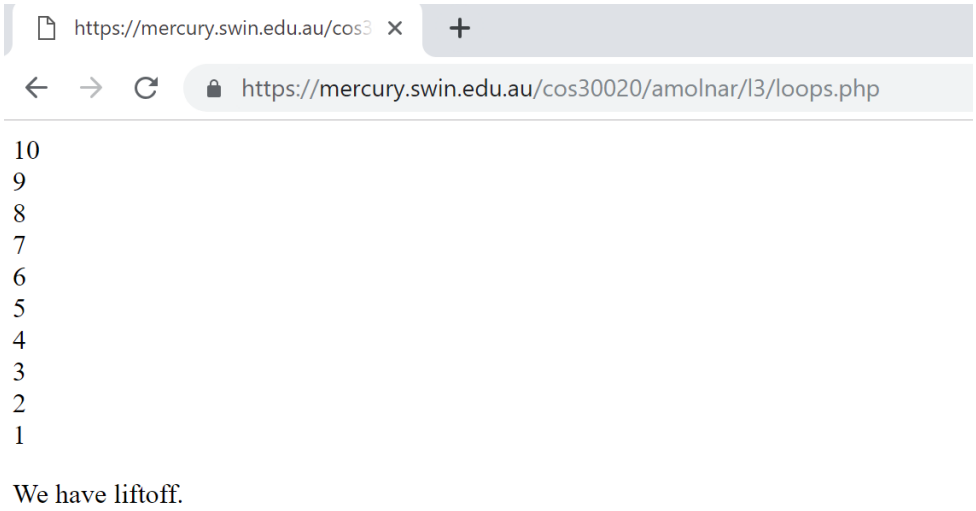
```
$count = 1;
while ($count <= 5) {
      echo "$count<br>";
      $count++;
}
echo "<p>You have printed 5 numbers.</p>";
```



Output of a `while` statement using an increment operator

`while` Statement (continued)

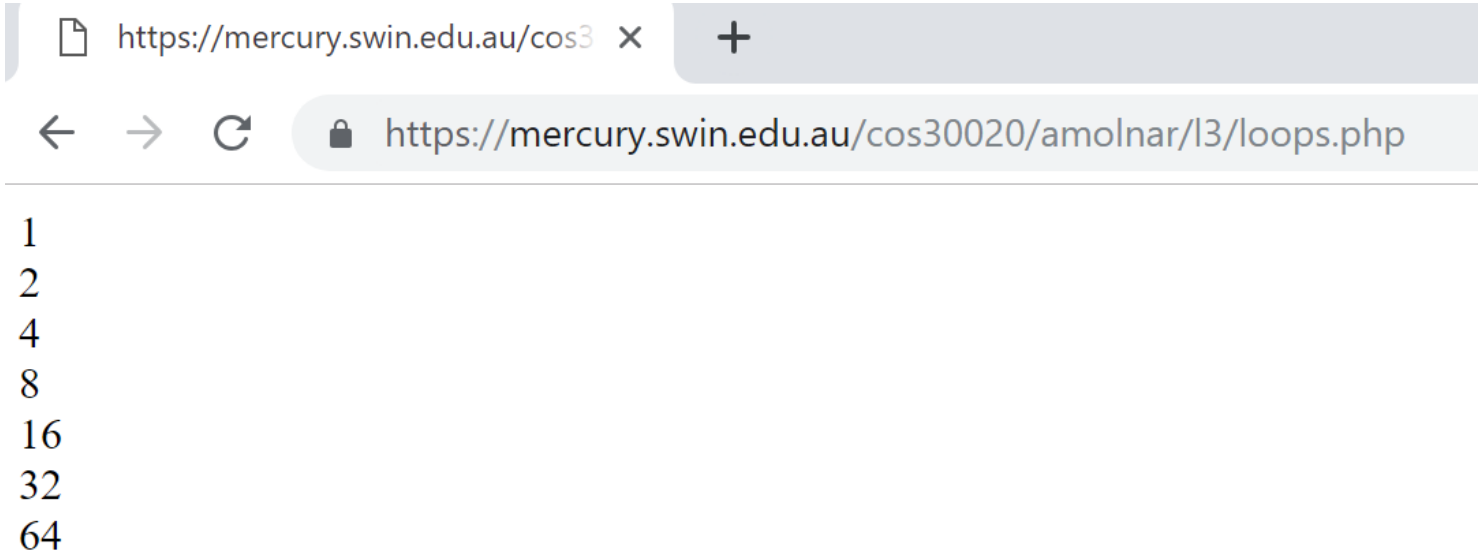
```
$count = 10;
while ($count > 0) {
      echo "$count<br>";
      $count--;
}
echo "<p>We have liftoff.</p>";
```



Output of a `while` statement using a decrement operator

`while` Statement (continued)

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



the assignment operator *=

`while` Statement (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";
}
```

- The `continue` statement

`do...while` Statement

- 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);
```

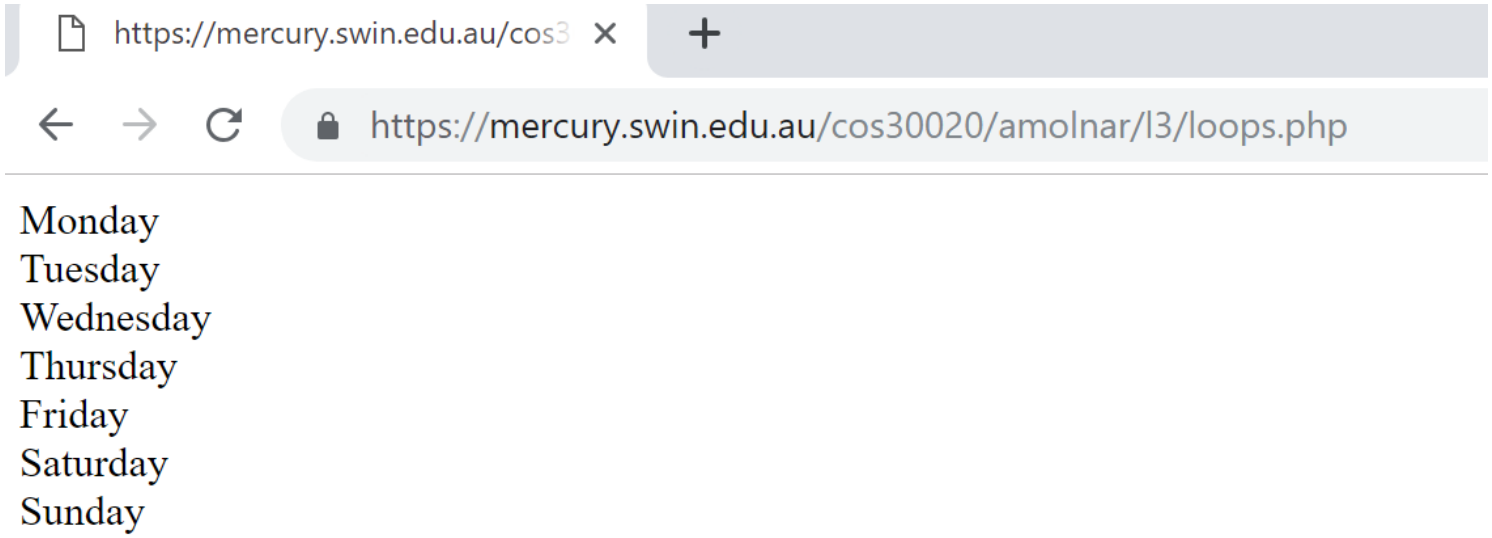
`do...while` Statement (continued)

- `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` Statement (continued)

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



Output of days of week script in Web browser

`for` Statement

- 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` Statement (continued)

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

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

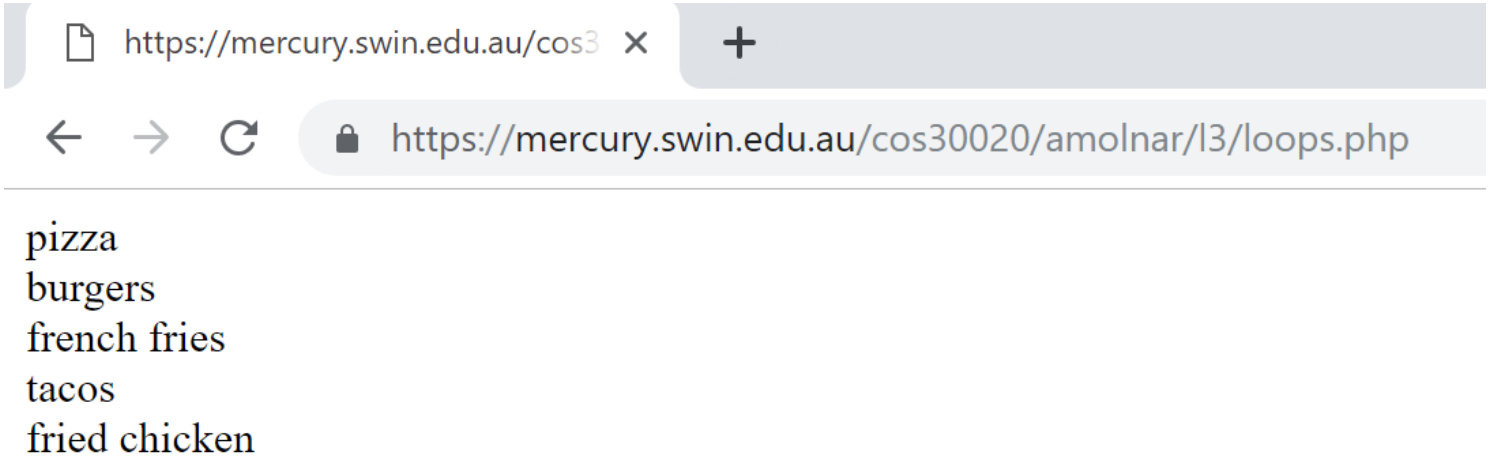
`for` Statement (continued)

```
$fastFoods = array("pizza", "burgers", "french fries",
   "tacos", "fried chicken");

for ($count = 0; $count < 5; $count++) {

      echo $fastFoods[$count], "<br>";

}
```



Output of fast-foods script

`foreach` Statement

- 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;
}
```

`foreach` Statement (continued)

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

 foreach ($daysOfWeek as $day) {

      echo "<p>$day</p>";

 }
```

Summary

- Functions are groups of statements that you can execute as a single unit
- Autoglobals contain client, server, and environment information that you can use in your scripts
- Decision making or flow control is the process of determining the order in which statements execute in a program

Summary (continued)

- The `if` statement is used to execute specific programming code if the evaluation of a conditional expression returns a value of true
- 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 `switch` statement controls program flow by executing a specific set of statements, depending on the value of an expression

Summary (continued)

- A `while` statement repeats a statement or a series of statements as long as a given conditional expression evaluates to true
- The `do...while` statement executes a statement or statements once, then repeats the execution as long as a given conditional expression evaluates to true
- The `for` statement is used for repeating a statement or a series of statements as long as a given conditional expression evaluates to true
- The `foreach` statement is used to iterate or loop through the elements in an array