

COS10011/60004 Creating Web Applications

Lecture 8 - Server-side Programming PHP: Part 1



Unit of Study Outline

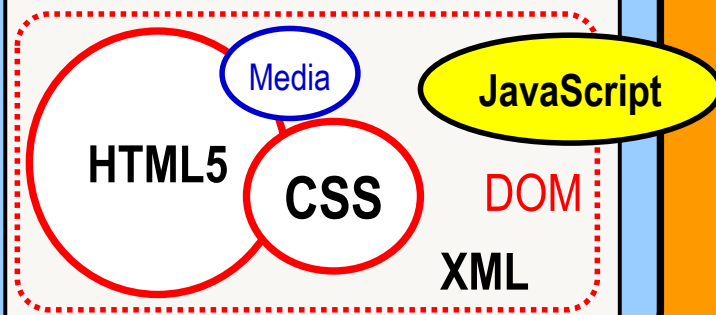
Internet Technologies: TCP/IP, URLs, URIs, DNS, MIME, SSL

Web Technologies: HTTP, HTTPS, Web Architectural Principles

Client Side Technologies:

Web Applications, Markup Languages

Web Documents



Server Side Technologies:

PHP, SSI, ...

Server-Side Data

MySQL

*Standards
Quality Assurance
Accessibility
Usability
Security*

Outline



- Client/Server Architecture
- PHP Scripting
- PHP Variables and Constants
- Data Types
- Arrays
- Expressions
- Functions and Scope
- Control Flow

3 - Creating Web Applications, © Swinburne



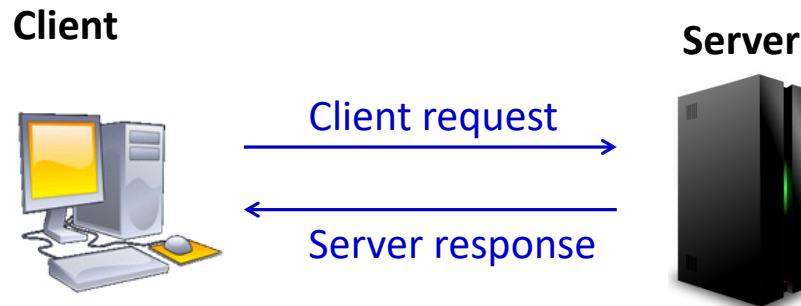
CLIENT/SERVER ARCHITECTURE



Client/Server Architecture



- A system consisting of a **Client** and a **Server** is known as a **two-tier** client/server system



The design of a two-tier client/server system

Client/Server Architecture (continued)



- **Client** (“front end”):
 - Presents an interface to the user
 - Gathers information from the user, submits it to a server, then receives, formats, and presents the results returned from the server
- **Server** (“back end”):
 - A computer from which a client requests information
 - Fulfills a request for information by managing the request or serving the requested information to the client
 - Responsible for data storage and management

Client/Server Architecture (continued)

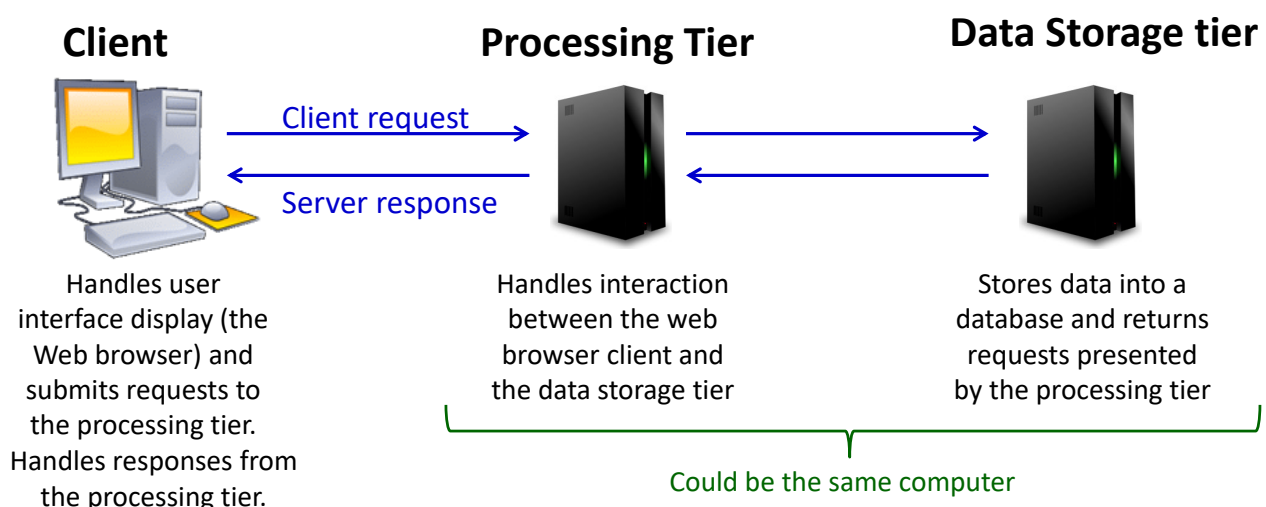


- A **three-tier**, or **multi-tier**, client/server system consists of three distinct pieces:
 - **Client tier**, or **user interface tier**, is the Web browser
 - **Processing tier**, or **middle tier**, handles the interaction between the Web browser client and the **data storage tier**
 - Performs necessary processing or calculations based on the request from the client tier
 - Handles the return of any information to the client tier

7 - Creating Web Applications, © Swinburne



Client/Server Architecture (continued)



The design of a three-tier client/server system

8 - Creating Web Applications, © Swinburne





PHP SCRIPTING

<http://php.net/manual/en/langref.php>



Server-Side Scripting and PHP



- **Server-side scripting** refers to a scripting language that is executed on a Web server
- **PHP** is a server-side ***embedded scripting language*** that is used to develop interactive web sites
 - Is easy to learn
 - Includes object-oriented programming capabilities
 - Supports many types of databases (MySQL, Oracle, Sybase, ODBC-compliant)



Server-Side Scripting and PHP (continued)



- **Embedded Scripts** are scripts that are **embedded** or linked into HTML documents, and stored on the server.
- In response to client requests, the **called pages are “parsed”** by the **server software**, the **embedded scripts are “processed”**, and the requested information or content is **returned as html**.
- Client requests often include parameters (key=value pairs) that are passed to the server, so the embedded scripts can query databases, or retrieve other dynamic information.
- As the server returns complete **“plain html”** web pages, the client response is **browser independent**.
- The embedded script **is not visible to the client**
- **the client only sees the completed html page.**

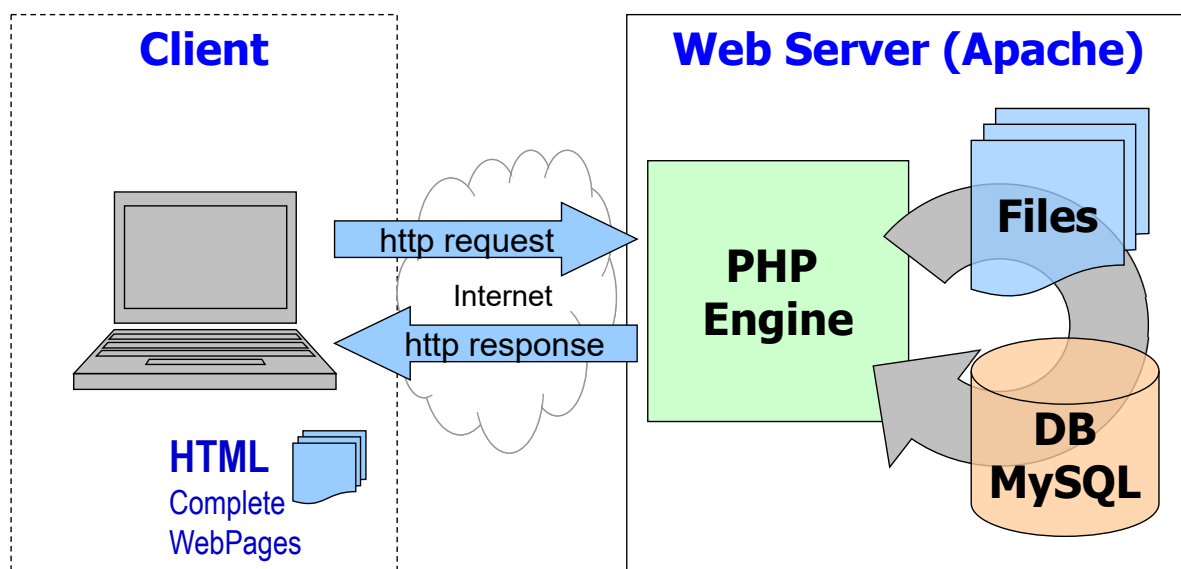
11 - Creating Web Applications, © Swinburne



Server-Side Scripting and PHP (continued)



Apache/PHP/MySQL example



12 - Creating Web Applications, © Swinburne



Server-Side Scripting and PHP (continued)



What is PHP?

<http://www.php.net>

- **It generates HTML**
- **PHP: Hypertext Preprocessor**
- a server-side scripting language,
- scripts are executed on the server
- supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- open source software (OSS)
- free to download and use
- PHP filename .php

13 - Creating Web Applications, © Swinburne

Source: w3schools



Server-Side Scripting and PHP (continued)

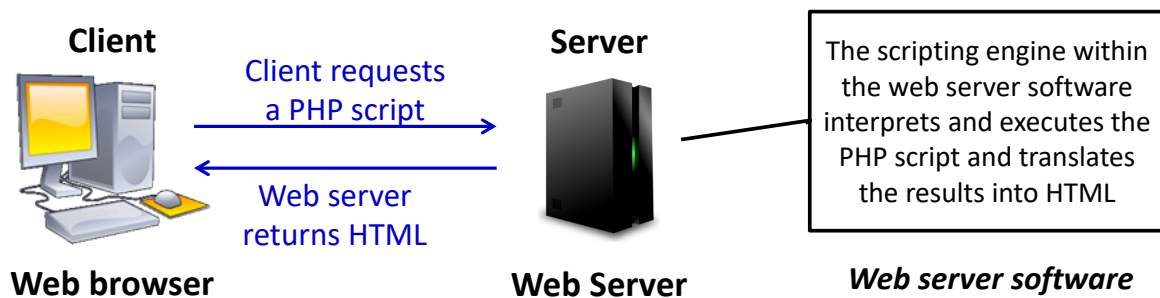


- PHP exists and executes solely on a web server, where it performs various types of processing or accesses databases
- PHP can't access or manipulate a client-side web browser (*c.f. JavaScript which can.*)

14 - Creating Web Applications, © Swinburne



Server-Side Scripting and PHP (continued)



How a Web server processes a PHP script

- **General rule:**

Use *client-side scripting* to handle user interface processing and light processing, such as form data validation;

Use *server-side scripting* for intensive calculations and data storage.

First PHP Example: myfirst_php.php



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <title>My Website</title>
    <!-- other meta here -->
</head>
<body>
<p>Hello World in unprocessed HTML</p>
<?php
    echo "<p>Hello World in HTML created by PHP</p>"
?>
</body>
</html>
```

Filename must have a PHP extension to be recognised by the pre-processor on the server

PHP code block

Output string in quotes as HTML



PHP Script Blocks

- **Code declaration blocks** are separate sections within a web page that are interpreted by the scripting engine
- There are four types of code declaration blocks:

- **Standard PHP script delimiters**

`<?php` statements; `?>`



Use this coding template

- (The `<script>` element)

`<script language =“php”>` statements; `</script>` ❌

- (Short PHP script delimiters)

`<?>` statements; `?>` ❌

- (ASP-style script delimiters)

`<%>` statements; `%>` ❌



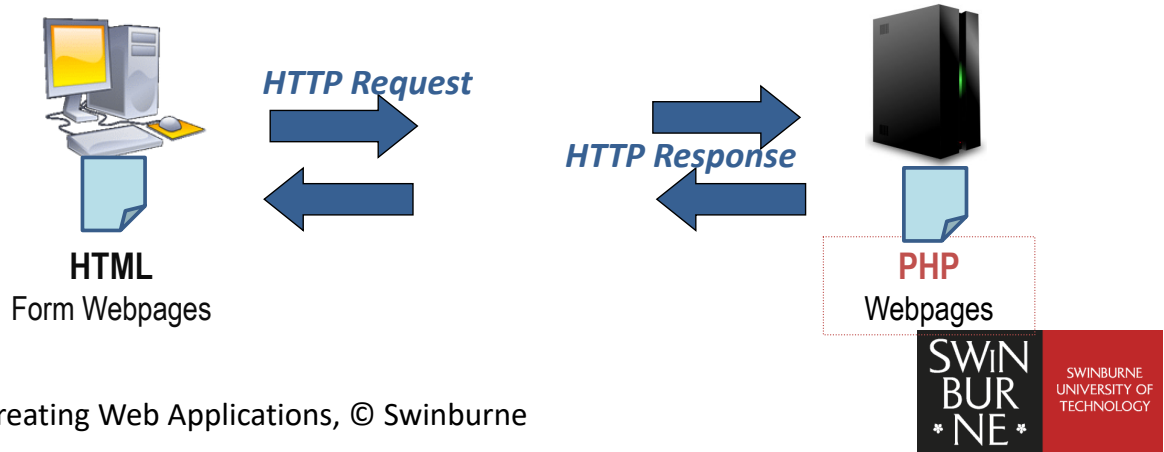
Generating HTML

- To return the results of any processing that occurs within a PHP code block, to the client, you must use an `echo` or `print` statement
- The **`echo`** and **`print` statements** create new text on a Web page that is returned as a response to a client
- `echo` and `print` statements are virtually identical except:
 - **`print`** statement accepts only a single argument and returns a value of 1
 - **`echo`** statement accepts multiple arguments and does not return any value

Generating HTML (continued)



- PHP scripts are executed.
Only HTML elements are sent back to the client
- Unless there are `echo` or `print` statements, (or plain HTML codes) in the PHP page, the returned web page will be blank,



Generating HTML (continued)



Example. Given the following PHP embedded script

```
...  
<body>  
<h1>Multiple Script Sections</h1>  
<h2>First Script Section</h2>  
<?php echo "<p>Output from the first ".  
    "script section.</p>"; ?>  
<h2>Second Script Section</h2>  
<?php echo "<p>Output from the second ".  
    "script section.</p>"; ?>  
</body>  
</html>
```

Note: PHP source files will not validate as HTML!

Generating HTML (continued)



Example.

The following HTML code is returned to the client

```
...
</head>
<body>
<h1>Multiple Script Sections</h1>
<h2>First Script Section</h2>
<p>Output from the first script section.</p>
<h2>Second Script Section</h2>
<p>Output from the second script section.</p>
</body>
</html>
```

Note: Validate the HTML in the served page.



Handling quotes



- How could we **echo** an image element `` ?

`echo "";` 

`echo "";` 

Use escape characters

`echo "";`

Use nested single quotes





PHP script

- uses round brackets () for operator precedence and argument lists
- uses square brackets [] for arrays and square bracket notation
- uses curly or brace brackets { } for blocks
- is embedded into an HTML file
- is never sent to a client's Web browser



PHP Script Syntax (continued)



A web page document

- that contains PHP code
must have an extension of .php
This is the default extension that most Web servers use to process PHP scripts
- that does not contain any PHP code
should have an .htm or .html extension
This reduces the server load on the Web server





PHP VARIABLES AND CONSTANTS

<http://php.net/manual/en/language.variables.php>

<http://php.net/manual/en/language.constants.php>



Example with variables



```
<html>
...
<body>
<h1>Hello World!</h1>
<?php
  echo "<p>";
  $i=1;
  while($i<=5) {
    echo "The number is " . $i . "<br />";
    $i++;
  }
  echo "</p>";
?>
</body>
</html>
```

All variables start with the symbol **\$**
Variables have **local scope** by default.
Unlike JavaScript, no 'var' needed.

String concatenation
operator in PHP





Would this work?

```
<?php
    for ($i = 1; $i < 7; $i++) {
        echo "<h$i>Heading $i</h$i>";
    }
?>
```

- Let's test it.
- YES! Variable output can be any HTML
 - not just text nodes

But we need to be careful
with our quotes. More later



Interleaving PHP with HTML

Would this work?

```
<?php
    for ($i = 1; $i < 7; $i++) { echo "<h$i>"
?>
```

Heading

HTML outside PHP block

```
<?php
    echo "$i</h$i>"; }
?>
```

- YES! PHP can be arbitrarily interleaved with HTML
(but don't break a string)



Variables and Constants

- Values stored in computer memory are called **variables** or **constants**
- Data contained in variables or constants are classified into categories known as **data types**
- The name you assign to a variable is called an **identifier** and in PHP it:
 - must begin with a dollar sign (\$)
 - can include letters (A to Z, a to z) and numbers (0 to 9) or an underscore (_) ... *but cannot start with a number*
 - cannot include spaces
 - is case sensitive



Variables - Naming

- Suggested naming style for variables
`$votingAge`
or
`$voting_age`
- Q: Do the two variable names below, refer to the same variable (identifier)?
 - `$firstName`
 - `$FirstName`

PHP is Case Sensitive

Variables - Declaring, initialising, modifying



- Specifying and creating a variable name is called ***declaring the variable***
- Assigning a first value to a variable is called ***initialising the variable***
- In PHP, you must ***declare*** and ***initialise*** a variable in the same statement:

```
$variable_name = value;
```

- You can later change the variable's value:

```
$variable_name = new_value;
```

Variables - Declaring, initialising, modifying



- The data type of a variable (identifiers) or constant depends on the **data type** of the value assigned to it

- `$unitName = "Creating Web Applications";`
- `$lectureHours = 2;`
- `$creditPoints = 12.5;`
- `$isCoreUnit = TRUE;`

Hint: Use meaningful names
Notice any naming pattern?

- Q: Are the following correct?
 - `$unitCode = COS10011;`
 - `$creditPoints = 12.5cp;`

Variables – Outputting the Values



- To output the contents of a variable, pass the variable name to the `echo` statement with/without enclosing it in double quotes:

```
$votingAge = 18;  
echo $votingAge;
```

Outputting variables in strings



- 3 different techniques

`<?php`

```
for ($i = 0; $i < 10; $i++) {  
    echo "<p>Number " , $i , "</p>";
```

Technique 1 - listing

```
    echo "<p>Number " . $i . "</p>";
```

Technique 2 - concatenation

```
    echo "<p>Number $i</p>";
```

Technique 3 – embedded (double quotes)

`?>`



What would happen here?

<?php

```
for ($i = 0; $i < 10; $i++) {  
    echo "<p>Number $i +1 </p>";  
}
```

?>

Suppose we wanted to add 1
to the variable in the loop

Would not work. Need:

```
echo "<p>Number ", $i +1, "</p>";
```

Technique 1 - listing

35 - Creating Web Applications, © Swinburne



Variables – Outputting the Values



- Note differences if surrounded by double or single quotes**

```
echo "<p>The legal voting age is  
$votingAge.</p>";
```

The value contained in \$votingAge will be printed

```
echo '<p>The legal voting age is  
$votingAge.</p>';
```

The text '\$votingAge' will be printed

If in doubt, separate with commas

```
echo "<p>The legal voting age is ",  
    $votingAge, ".</p>";
```

36 - Creating Web Applications, © Swinburne





Constants

- A **constant** contains a value that **does not change** during the course of program execution
- Constant names **do not** begin with a dollar sign (\$)
- Method 1
 - Use the **define("CONSTANT_NAME", value)** function to create a constant
e.g. `define("MAX_VAL", 5);`
 - The value can be a text string, number, or Boolean value
- Method 2
 - Use the **const** keyword (from PHP 5.3 on)
 - e.g. `const MAX_VAL = 5;`
 - Cannot be declared inside functions, loops, if statements or try/ catch blocks
- PHP includes numerous predefined constants that you can use in your scripts e.g. `PHP_INT_MAX`

37 - Creating Web Applications, © Swinburne



Example: use of Constants



do not forget the double quotes

<?php

```
define ("MAX_ELEMENT", 8);
echo "<ol>";
for ($i = 0; $i < MAX_ELEMENT; $i++) {
    echo "<li>item ", $i+1, " </li>";
}
echo "</ol>";
```

?>

For calculations use
Technique 1 – listing



38 - Creating Web Applications, © Swinburne



Constants – Naming



- Suggested naming style for constant

PASSING_MARK

- Q: Which one of the following is a constant?

\$MAX_ELEMENTS

MAX_ELEMENTS



PHP DATA TYPES

<http://php.net/manual/en/language.types.php>



PHP is a loosely typed programming language

- **Strongly typed programming languages** require you to declare the data types of variables
 - **Static** or **strong typing** refers to data types that ***do not*** change after they have been declared
 - *C is a strongly typed programming language*
- **Loosely typed programming languages** do not require you to declare the data types of variables
 - **Dynamic** or **loose typing** refers to data types that can change after they have been declared
 - *PHP is a loosely typed programming language.*



PHP Data Types (continued)



- A **data type** is the specific category of information that a variable contains
- Data types that can be assigned only a single value are called **primitive types**

Data Type	Description
Integer	Positive or negative numbers with no decimal places
Floating-point numbers	Positive or negative numbers with decimal places, or expressed in exponential notation
Boolean	Logical value represented by true or false
String	Any sequence of characters
NULL	An empty value



Numeric



PHP supports two *numeric* data types:

- An **integer** is a positive or negative number with no decimal places
e.g. -250, 2, 100, 10,000
- A **floating-point number** is a number that contains decimal places or that is written in exponential notation
e.g. -6.16, 3.17, 2.7541
 - **Exponential notation**, or **scientific notation**, is a short way for writing very large numbers or numbers with many decimal places eg. 2.0e11

Boolean



- A **Boolean** is a value of **true** or **false**
- It decides which part of a program should execute and which part should compare data

String



- **String** is a sequence of characters
- It is created directly by placing the series of characters between double or single quotes, for example
 - "This is a string"
 - 'This is also a string'

PHP Data Types (continued)



The PHP language supports:

- A **resource** data type – a special variable that holds a reference to an external resource, such as a database or an XML file
- **Reference** or **composite** data types, which contain multiple values or complex types of information
 - Two reference data types: **arrays** and **objects**



ARRAYS

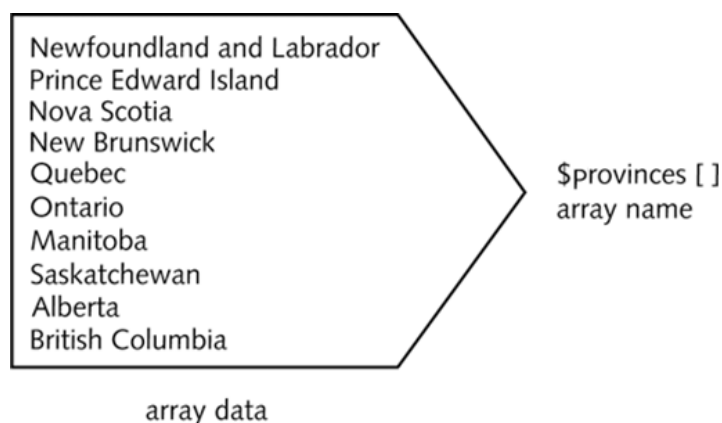
<http://php.net/manual/en/language.types.array.php>



Arrays



- An **array** contains a set of data represented by a single variable name



Conceptual example of an array
Canada's provinces



Declaring and Initialising Indexed Arrays



- An **element** refers to each piece of data that is stored within an array
 - By default, it starts with the number zero (0)
- An **index** is an element's numeric position within the array
 - Referenced by enclosing its index in brackets at the end of the array name:
 - **\$provinces[1]**

Creating an Array



- The `array()` construct syntax is:
`$array_name = array(values);`

```
$provinces = array(  
    "Newfoundland and Labrador",  
    "Prince Edward Island",  
    "Nova Scotia",  
    "New Brunswick",  
    "Quebec",  
    "Ontario",  
    "Manitoba",  
    "Saskatchewan",  
    "Alberta",  
    "British Columbia"  
);
```



Creating an Array (continued)

- Array name and brackets syntax is:
`$array_name[]`

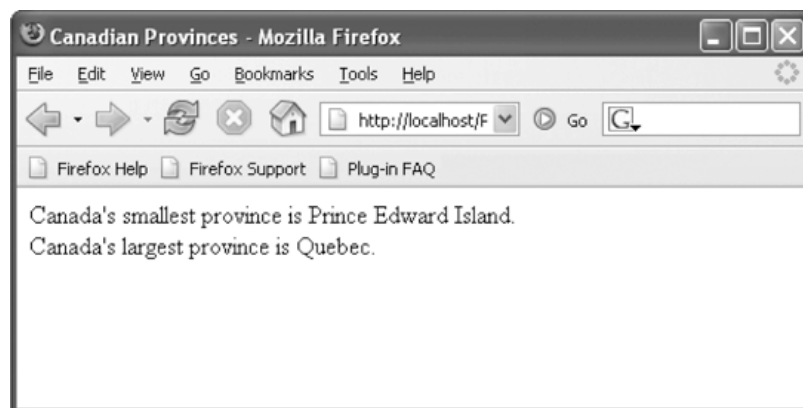
```
$provinces[] = "Newfoundland and Labrador";  
$provinces[] = "Prince Edward Island";  
$provinces[] = "Nova Scotia";  
$provinces[] = "New Brunswick";  
$provinces[] = "Quebec";  
$provinces[] = "Ontario";  
$provinces[] = "Manitoba";  
$provinces[] = "Saskatchewan";  
$provinces[] = "Alberta";  
$provinces[] = "British Columbia";
```

***Note:** In PHP, array elements can be of different data types*

Accessing Element Information



```
echo "<p>Canada's smallest province is  
$provinces[1].<br />";  
echo "Canada's largest province is  
$provinces[4].</p>";
```



Output of elements in the `$provinces[]` array



count () Function

- Use the **count ()** function to find the total number of elements in an array

```
$provinces = array("Newfoundland and Labrador",  
"Prince Edward Island", "Nova Scotia",  
"New Brunswick", "Quebec", "Ontario", "Manitoba",  
"Saskatchewan", "Alberta", "British Columbia");
```

```
$territories = array("Nunavut",  
"Northwest Territories", "Yukon Territory");
```

```
echo "<p>Canada has ",  
count($provinces), " provinces and ",  
count($territories), " territories.</p>";
```

Output:

Canada has 10 provinces and 3 territories.

print_r () Function



- Use to print or return information about variables
- Most useful with arrays because they print the index and value of each element



Output of the **\$provinces []** array with the **print_r ()** function



Modifying Array Elements

- To change an array value, use the index of an individual element of the array:

```
$hospitalDepts = array(  
    "Anesthesia",           // first element [0]  
    "Molecular Biology",    // second element [1]  
    "Neurology");           // third element [2]
```

To change the first array element in the **\$hospitalDepts[]** array from “Anesthesia” to “Anesthesiology” use:

```
$hospitalDepts[0] = "Anesthesiology"
```

55 - Creating Web Applications, © Swinburne



PHP EXPRESSIONS

<http://php.net/manual/en/language.expressions.php>



PHP Expressions



- An **expression** is a literal value or variable
 - that can be evaluated by the PHP scripting engine to produce a result
- **Operands** are variables and literals contained in an expression
- A **literal** is a value such as a literal string or a number
- **Operators** are symbols (e.g. +, *) that are used in expressions to manipulate operands

57 - Creating Web Applications, © Swinburne



PHP Expressions (continued)



PHP Operator Types

Operator Type	Description
Array	Performs operations on arrays
Arithmetic	Performs mathematical calculations
Assignment	Assigns values to variables
Comparison	Compares and returns a Boolean value
Logical	Performs Boolean operations on Boolean operands
Special	Performs various tasks, these operators do not fit within other operator categories

- A **binary operator** requires an operand before and after the operator
- A **unary operator** requires a **single** operand either before or after the operator

58 - Creating Web Applications, © Swinburne





Arithmetic Operators

- **Arithmetic operators** are used in PHP to perform mathematical calculations

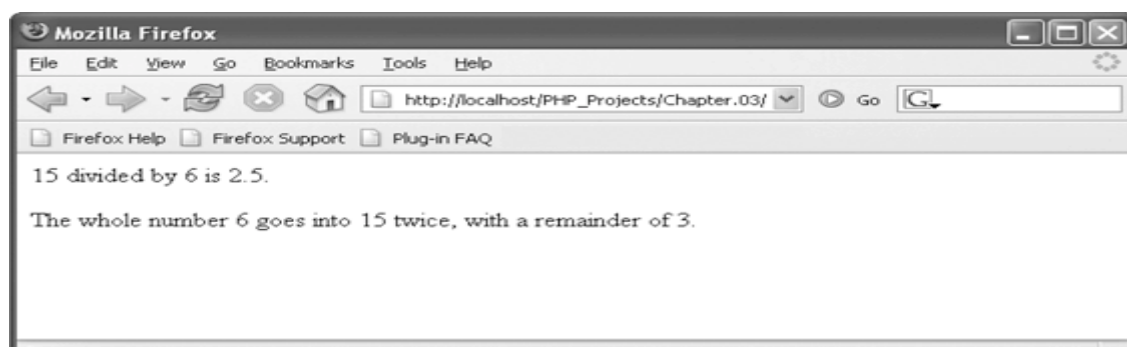
PHP arithmetic binary operators

Operator	Name	Description
+	Addition	Adds two operands
-	Subtraction	Subtracts one operand from another operand
*	Multiplication	Multiplies one operand by another operand
/	Division	Divides one operand by another operand
%	Modulus	Divides one operand by another operand and returns the remainder

Arithmetic Operators (continued)



```
$divisionResult = 15 / 6;
$modulusResult = 15 % 6;
echo "<p>15 divided by 6 is $divisionResult.</p>";
// result of '2.5'
echo "The whole number 6 goes into 15 twice, with a
remainder of $modulusResult.</p>";
// result of '3'
```





Arithmetic Unary Operators

- The increment (++) and decrement (--) unary operators can be used as prefix or postfix operators
- A **prefix operator** is placed before a variable
- A **postfix operator** is placed after a variable

PHP arithmetic unary operators

Operator	Name	Description
++	Increment	Increases an operand by a value of one
--	Decrement	decreases an operand by a value of one

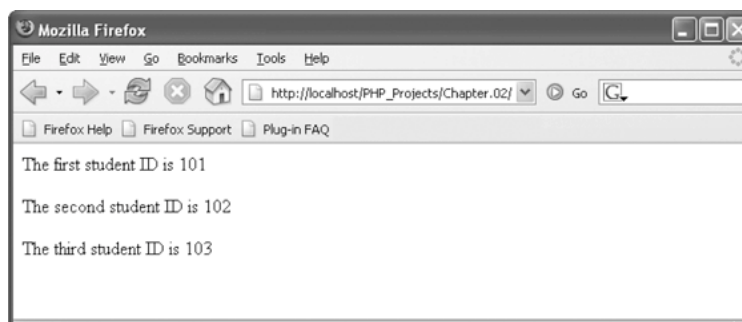
Arithmetic Unary Operators (continued)



```
$StudentID = 100;
$CurStudentID = ++$StudentID; // assigns '101'
echo "<p>The first student ID is ",
    $CurStudentID, "</p>";
$CurStudentID = ++$StudentID; // assigns '102'
echo "<p>The second student ID is ",
    $CurStudentID, "</p>";
$CurStudentID = ++$StudentID; // assigns '103'
echo "<p>The third student ID is ",
    $CurStudentID, "</p>";
```

prefix increment operator

Script that uses the prefix increment operator



Output of the prefix version of the student ID script

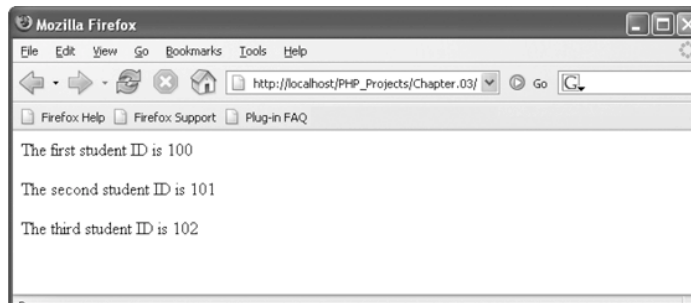
Arithmetic Unary Operators (continued)



```
$StudentID = 100;  
$CurStudentID = $StudentID++; // assigns '100'  
echo "<p>The first student ID is ",  
    $CurStudentID, "</p>";  
$CurStudentID = $StudentID++; // assigns '101'  
echo "<p>The second student ID is ",  
    $CurStudentID, "</p>";  
$CurStudentID = $StudentID++; // assigns '102'  
echo "<p>The third student ID is ",  
    $CurStudentID, "</p>";
```

postfix increment operator

Script that uses the postfix increment operator



Output of the postfix version of the student ID script

63 - Creating Web Applications, © Swinburne



Arithmetic Unary Operators (continued)



- Q: What is the difference between prefix increment operator and postfix increment operator ?

64 - Creating Web Applications, © Swinburne



Assignment Operators



- **Assignment operators**

are used for assigning a value to a variable:

```
$myFavoriteSuperHero = "Superman";
```

```
$myFavoriteSuperHero = "Batman";
```

- **Compound assignment operators**

perform mathematical calculations on variables and literal values in an expression, and then assign a new value to the left operand

Assignment Operators (continued)



PHP assignment operators

Operator	Name	description
=	Assignment	Assigns the value of the right operand to the left operand
+=	Compound addition assignment	Adds the value of the right operand to the value of the left operand and assigns the sum to the left operand
-=	Compound subtraction assignment	Subtracts the value of the right operand to the value of the left operand and assigns the difference to the left operand
*=	Compound multiplication assignment	Multiplies the value of the right operand to the value of the left operand and assigns the product to the left operand
/=	Compound division assignment	Divides the value of the right operand to the value of the left operand and assigns the quotient to the left operand
%=	Compound modulus assignment	Divides the value of the right operand to the value of the left operand and assigns the remainder (modulus) to the left operand

Assignment Operators (continued)



```
$x = 100;
```

```
$y = 200;
```

```
$x += $y;    same as    $x = $x + $y;  
(Answer: 300)
```

```
$x = 2;
```

```
$y = 6;
```

```
$x *= $y;    same as    $x = $x * $y;  
(Answer: 12)
```

Comparison and Conditional Operators



- **Comparison operators**
are used to compare two operands and determine how one operand compares to another.
- A Boolean value of **true** or **false** is returned after two operands are compared
- *The comparison operator compares values, whereas the assignment operator assigns values*
- Comparison operators are used with **conditional statements** and **looping statements**

Comparison and Conditional Operators

(continued)



PHP comparison operators

Operator	Name	Description
==	Equal	Returns true if the operands are equal
===	Strict equal	Returns true if the operands are equal and of the same type
!= or <>	Not equal	Returns true if the operands are not equal
!==	Strict not equal	Returns true if the operands are not equal or not of the same type
>	Greater than	Returns true if the left operand is greater than the right operand
<	Less than	Returns true if the left operand is less than the right operand
>=	Greater than or equal	Returns true if the left operand is greater than or equal to the right operand
<=	Less than or equal	Returns true if the left operand is less than or equal to the right operand

69 - Creating Web Applications, © Swinburne



Comparison and Conditional Operators

(continued)



- The **conditional operator** executes one of two expressions, based on the results of a conditional expression
- The syntax for the conditional operator is:

```
conditional expression  
? expression1 : expression2;
```
- If the conditional expression evaluates to **true**, *expression1* executes
- If the conditional expression evaluates to **false**, *expression2* executes

70 - Creating Web Applications, © Swinburne

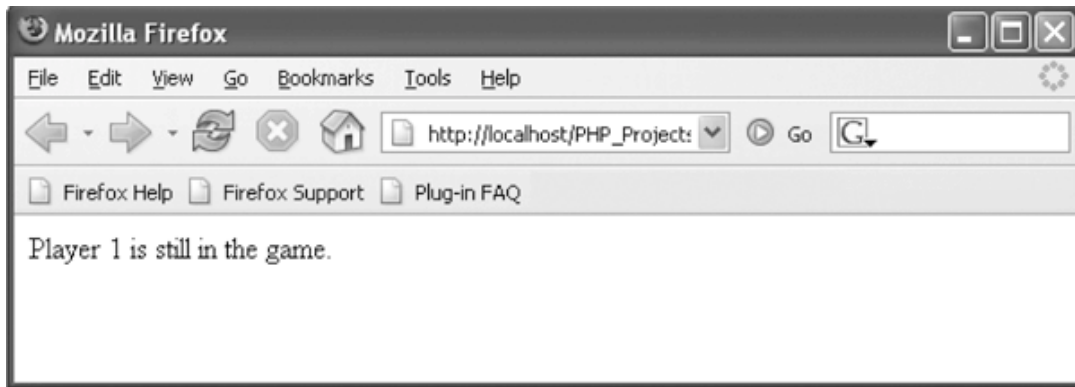


Comparison and Conditional Operators



(continued)

```
$blackjackPlayer1 = 20;  
($blackjackPlayer1 <= 21)  
    ? $result = "Player 1 is still in the game."  
    : $result = "Player 1 is out of the action.";  
echo "<p>", $result, "</p>";
```



Output of a script with a conditional operator

71 - Creating Web Applications, © Swinburne



Logical Operators



- **Logical operators** are used for comparing two Boolean operands for equality
- A Boolean value of true or false is returned after two operands are compared

PHP logical operators

Operator	Name	Description
&&, and	And	Returns true if both the left operand and right operand return a value of true; otherwise, it returns a value of false
, or	Or	Returns true if either the left operand or right operand returns a value of true; if neither operand returns a value of true, it returns a value of false
!	Not	Returns true if an expression is false and returns false if an expression is true

72 - Creating Web Applications, © Swinburne



Special Operators



PHP special operators

Operator	Description
new	Created a new instance of a user-defined or predefined object type
[]	Accesses an element of an array
=>	Specifies the index or key of an array element
,	Separates arguments in a list
?:	Executes one of two expressions based on the results of a conditional expression
instanceof	Returns true if an object is of a specified object type
@	Suppresses any error messages that might be generated by an expression to which it is prepended
(int), (integer), (bool), (boolean), (double), (string), (array), (object)	Casts or transform a variable of one data type into a variable of another data type

Note: These Special Operators are introduced throughout this unit as necessary

73 - Creating Web Applications, © Swinburne



Operator Precedence



- **Operator precedence** refers to the order in which operations in an expression are evaluated
- **Associativity** is the order in which operators of equal precedence execute
- *What to do if not certain when you write code? Add parentheses*



Operator Precedence (continued)



Operator precedence in PHP

Operator	Description	Associativity
new	New object	None
[]	Array elements	Right to left
!	Logical Not	Right to left
++	Increment	Right to left
--	Decrement	Right to left
(int) ...	Cast	Right to left
@	Suppress error message	Right to left
* / %	Multiplication/division	Left to right
+ - .	Addition/subtraction/string concatenation	Left to right
< <= > >=	Comparison	None
== != <> === !==	Equality	None
&&	Logical And	Left to right
	Logical Or	Left to right
?:	Conditional	Left to right
= += -= *= /= %=	Assignment	Right to left
and	Logical And	Left to right
or	Logical Or	Left to right
,	List separator	Left to right

75 - Creating Web Applications, © Swinburne



FUNCTIONS

<http://php.net/manual/en/language.functions.php>





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
- Syntax for defining a function is:

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

77 - Creating Web Applications, © Swinburne



Defining Functions (continued)

- PHP Functions 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

78 - Creating Web Applications, © Swinburne



Defining Functions (continued)



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

```
function printNames($name1, $name2, $name3)
{
    echo "<p>$name1</p>";
    echo "<p>$name2</p>";
    echo "<p>$name3</p>";
}
```

Calling Functions



Function
definition

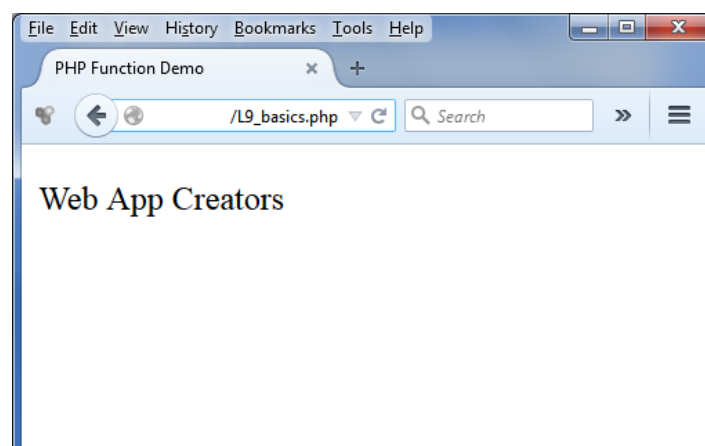
```
function printCompanyName($companyName) {
    echo "<p>$companyName</p>";
}
```

Formal
Parameter

Function
invocation

```
printCompanyName("Web App Creators");
```

Actual
Parameter



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;  
}
```

By default, variable in a PHP function has local scope. Unlike JavaScript no 'var' needed.

PHP inbuilt (internal) functions



- String functions - *Examples*

str_replace()

Replace all occurrences of the search string with the replacement string

htmlspecialchars()

Convert special characters to HTML entities

<http://php.net/manual/en/ref.strings.php>

- Variable Functions - *Examples*

is_int()

Find whether the type of a variable is integer

isset()

Determine if a variable is set and is not NULL

<http://php.net/manual/en/ref.var.php>



VARIABLE SCOPE

<http://php.net/manual/en/language.variables.scope.php>



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 one that 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, 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 $globalVariable;
    echo "<p>$globalVariable</p>";
}
$globalVariable = "Global variable";
testScope();
?>
```

If no **"global"** keyword, an error message would be printed out.

**Best Practice: Use local variables.
Pass parameters. Avoid global.**

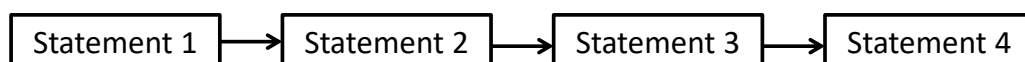
PHP CONTROL FLOW

<http://php.net/manual/en/language.control-structures.php>

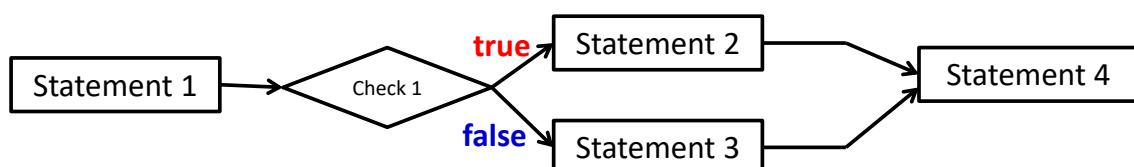
Three Models in Programming



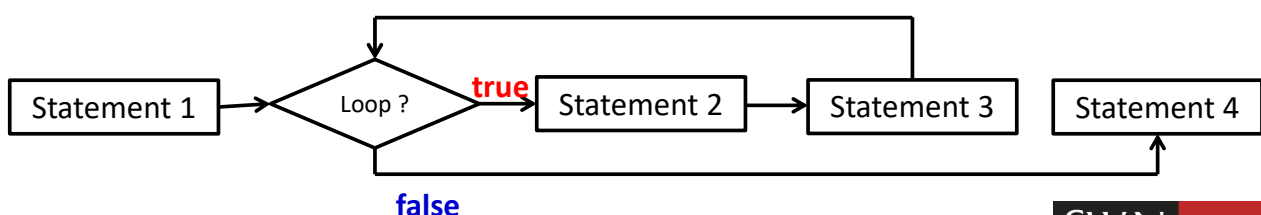
Sequence



Selection



Repetition





- **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**
- 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 will execute after the '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**
- 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

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

Note: Single statements within **if ... else** do not need braces. *But Best Practice to include them.*

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)



- Syntax for the `switch` statement is:

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


switch Statement (continued)



```
<?php
    $colour="red";
    switch ($colour) {
        case "red":
            echo "Red!";
            break;
        case "blue":
            echo "Blue!";
            break;
        case "green":
            echo "Green!";
            break;
        default:
            echo "Some other colour!";
    }
?>
```

97 - Creating Web Applications, © Swinburne



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

98 - Creating Web Applications, © Swinburne





Selection – Example with html

- We can mix php coding with html coding.
For example:

```
<?php if (conditional expression) {  
?>
```

```
A block of html code (eg. Show a form)
```

```
<?php  
} else {  
?>
```

```
Another block of html code (eg. Hide a form)
```

```
<?php  
} ?>
```

- It is simpler to use this model for multiple lines of static html, rather than 'echo' all the lines of html code inside the php.

Repetition



- 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**

- 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)

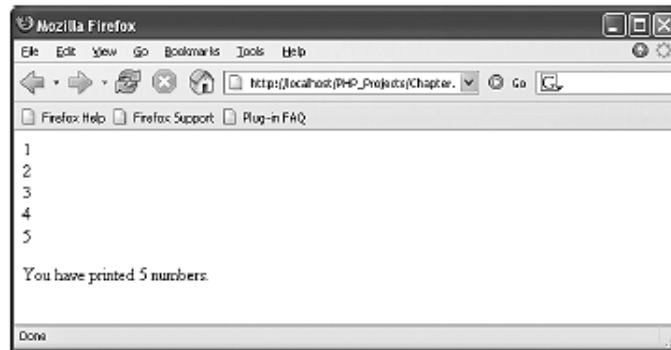


- Each repetition of a looping statement is called an **iteration**
- A **while** statement keeps repeating until its conditional expression evaluates to **false**
- A **counter** is a variable that *increments* or *decrements* with each iteration of a loop statement

while Statement (continued)



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



Q: If we added an `echo $count;` at the end, what would be the value?

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>";
```



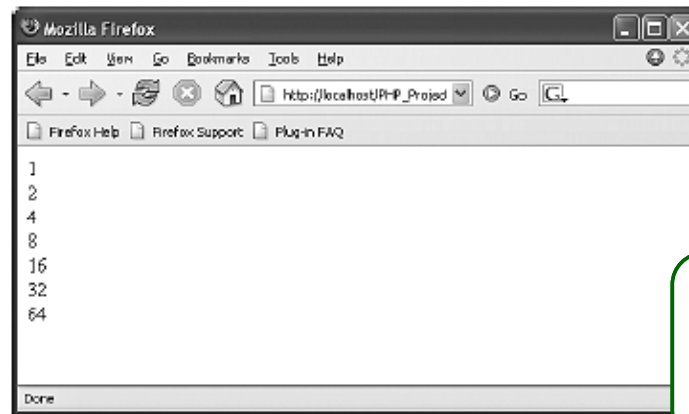
Q: If we added an `echo $count;` at the end, what would be the value?

Output of a while statement using a decrement operator

while Statement (continued)



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



Q: If we added an `echo $count;` at the end, what would be the value?

Output of a `while` statement using 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
<http://php.net/manual/en/control-structures.continue.php>

do...while Statement



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

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

do...while Statement (continued)



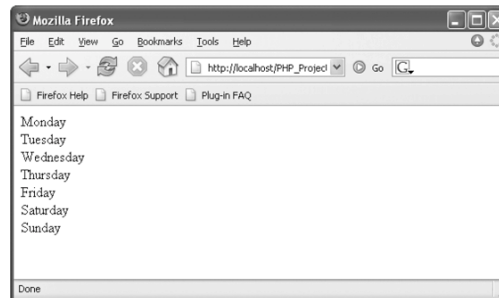
- **do...while** statements *always execute once*, before a conditional expression is evaluated

```
$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

Q: If we added an
`echo $count;`
at the end, what
would be the value?

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)



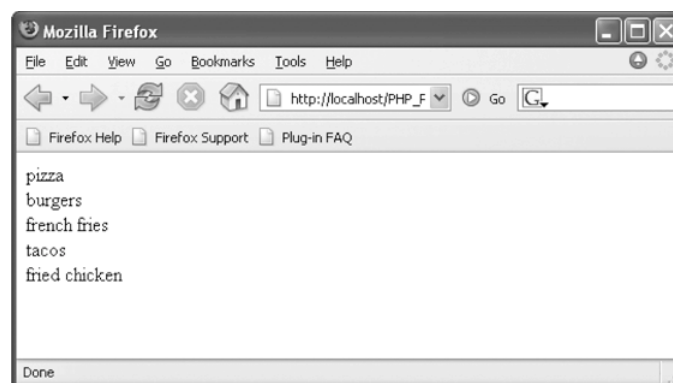
- Can also include code that initialises a counter and changes its value with each iteration
- Syntax of the **for** statement is:

```
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**

```
$daysOfWeek = array("Monday",  
    "Tuesday", "Wednesday", "Thursday",  
    "Friday", "Saturday", "Sunday");  
foreach ($daysOfWeek as $day) {  
    echo "<p>$day</p>"; }  
}
```

Note: Java Script Syntax is different:

```
for (variable in collectionOfObject) {  
    statements; }  
}
```

foreach Statement (continued)



- 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
- Syntax for the **foreach** statement is:

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

Appendix - Some Useful PHP Functions



strlen (*string*)

Return the length of a string

Example:

```
<?php
    echo strlen("CWA");
    // output: 3
    echo strlen("Hello world!");
    // output: 12
?>
```



Appendix - Some Useful PHP Functions



implode (*separator, array*)

Join array elements with a string

Example:

```
<?php
$arr = array('It', 'is', 'a', 'beautiful', 'day!');
echo implode(" ", $arr) . "<br>";
echo implode("---", $arr) . "<br>";
echo implode(" | ", $arr);
?>
```

Output

```
It is a beautiful day!
It---is---a---beautiful---day!
It | is | a | beautiful | day!
```

Appendix - Some Useful PHP Functions



explode (separator, string)

Break a string into an array based on the separator.

Example:

```
<?php
$colors = 'red,green,blue';
print_r(explode(',', $colors));
echo "<hr>";
$dob = '12/09/1998';
$dobArr = explode('/', $dob);
echo "The year of birth was " . $dobArr[2];
?>
```

Output

```
Array ( [0] => red [1] => green [2] => blue )
The year of birth was 1998
```

Appendix - Some Useful PHP Functions



strpos(string,find) (for more details, please refer to other references)

Find the position of the first occurrence of a string inside another string. It returns the position, or FALSE if the string is not found. **Note:** String positions start at 0, and not 1.

Example:

```
<?php
$str = "html,css,javascript,php";
echo strpos($str,"php") . "<br>";
if (strpos($str,"html") !==false)
    echo "Yes, we learn html.";
else
    echo "No, we don't learn html.";
?>
```

Output

```
20
Yes, we learn html.
```



What's Next? - PHP Part 2

119 - Creating Web Applications, © Swinburne

