SWINBURNE UNIVERSITY OF TECHNOLOGY

# COS10026
# Computing Technology Inquiry Project

**Lecture 8**

**PHP 2 – Functions and Form Data Processing**

## Topics

- Functions and Scope
- Control Flow
- Form Data Processing

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
- Syntax for defining a function is:

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

**Functions**
http://php.net/manual/en/language.functions.php

SWINBURNE UNIVERSITY OF TECHNOLOGY

# 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

SWINBURNE UNIVERSITY OF TECHNOLOGY

# 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

Formal Parameter

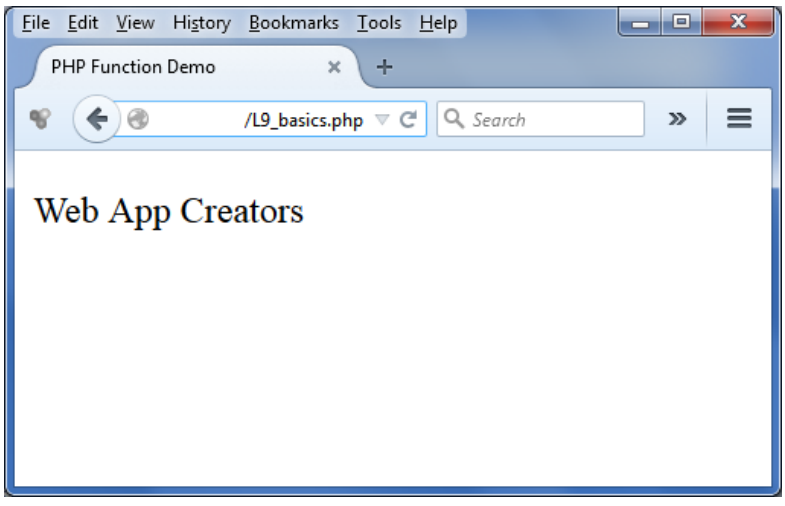
Function definition

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

Actual Parameter

Function invocation

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




**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;
}
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

# 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

SWINBURNE UNIVERSITY OF TECHNOLOGY

# 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

**Variable Scope**
http://php.net/manual/en/language.variables.scope.php


# Understanding Variable Scope (Cont.)

```php
<?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.**

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Topics

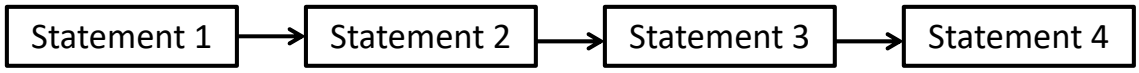
- Functions and Scope
- Control Flow
- Form Data Processing

**PHP Control Flow**
http://php.net/manual/en/language.control-structures.php

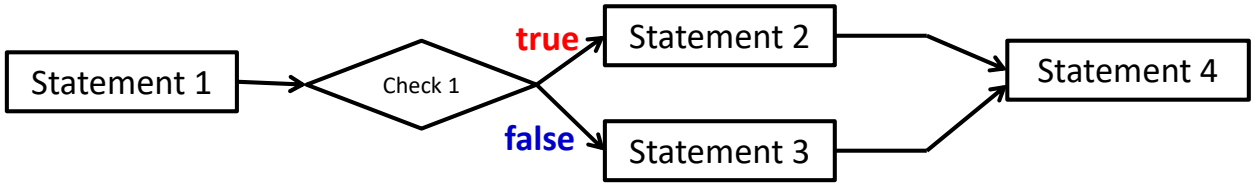
SWINBURNE UNIVERSITY OF TECHNOLOGY

# Three Models in Programming

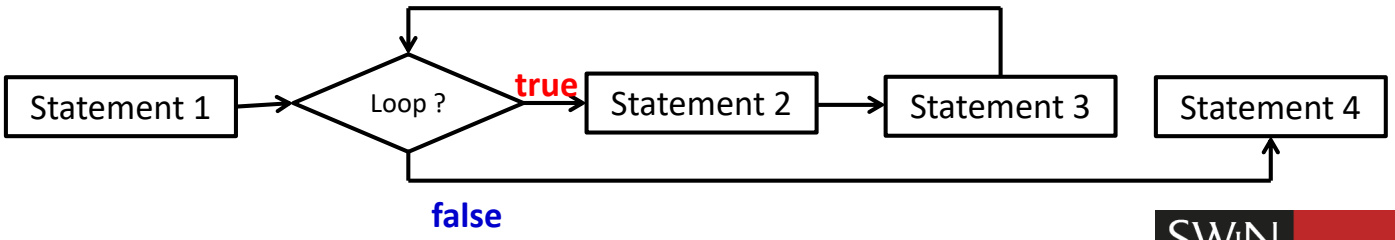
## Sequence




## Selection




## Repetition




# Selection

- **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;
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

# `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.***

SWINBURNE UNIVERSITY OF TECHNOLOGY

# 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>";
    }
}
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

# `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

SWINBURNE UNIVERSITY OF TECHNOLOGY

# `switch` Statement (continued)

- Syntax for the `switch` statement is:

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

SWINBURNE UNIVERSITY OF TECHNOLOGY

# `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!";
}
?>
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

# `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


# 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

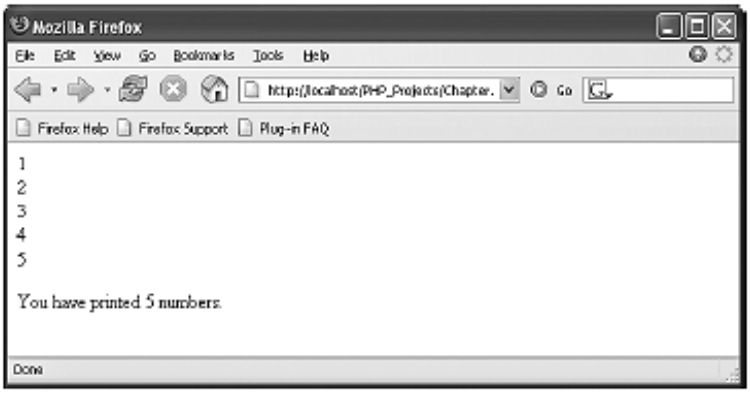
# `while` Statement (continued)

- 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

SWINBURNE UNIVERSITY OF TECHNOLOGY

# `while` Statement (continued)

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



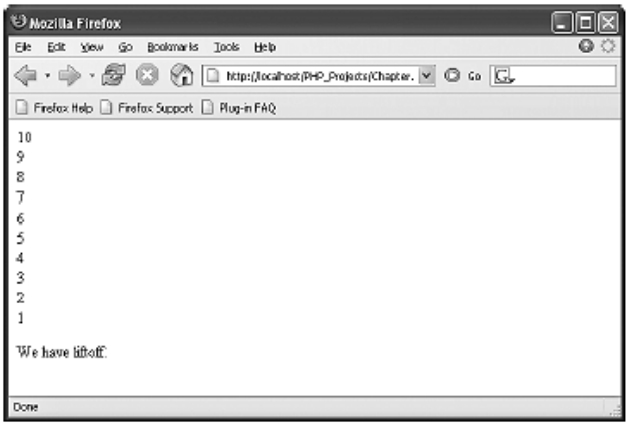

**Output of a `while` statement using an increment operator**

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

SWINBURNE UNIVERSITY OF TECHNOLOGY

# `while` Statement (continued)

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



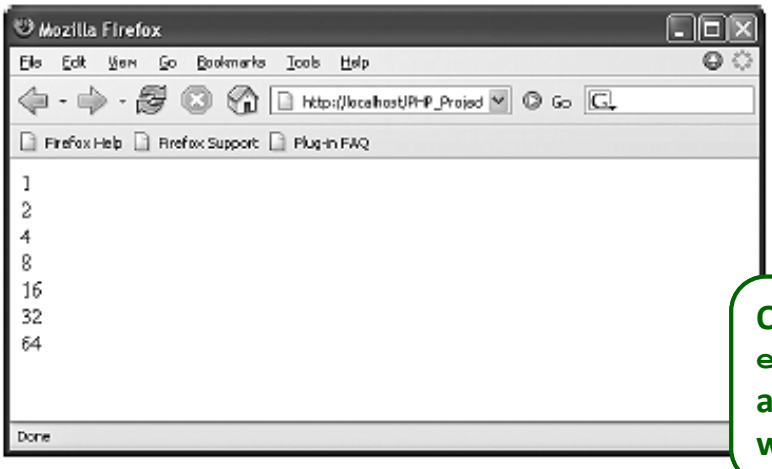

**Output of a `while` statement using a decrement operator**

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



# `while` Statement (continued)

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




**Output of a `while` statement using the assignment operator *=**

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

# `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);
```

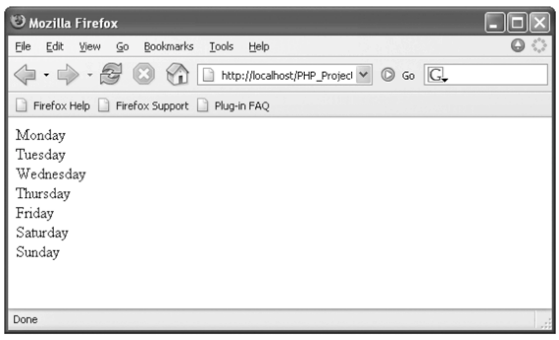
# do...while Statement (continued)

- **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`**

SWINBURNE UNIVERSITY OF TECHNOLOGY

# `for` Statement (continued)

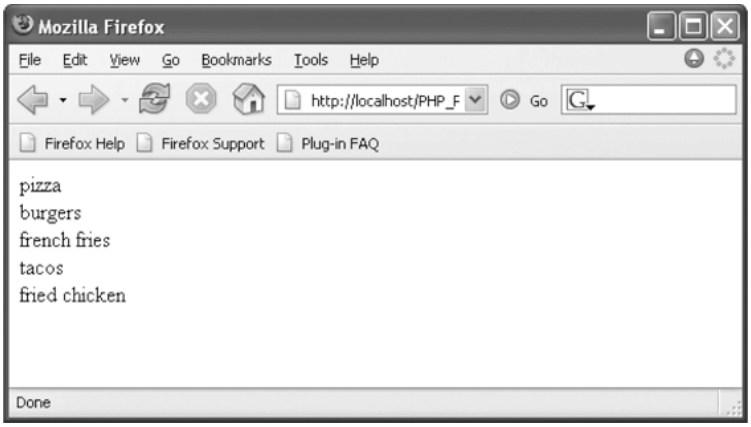
- Can also include code that initialises a counter and changes its value with each iteration
- Syntax of the **`for`** statement is:

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

SWINBURNE UNIVERSITY OF TECHNOLOGY

# 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;
}
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Topics

- Functions and Scope
- Control Flow

- Form Data Processing

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Form data extraction


must match

myfirst_phpform.html

```
....
<form method="post" action="php_process1.php">
    <p>Enter First Name:
    <input type="text" name="fname" />
    ...
    <input type="submit" value="Process" />
    </p>
</form>
...
```

php_process1.php

```
<?php
    ...
//Transfer form data to variables
    $fname = $_POST["fname"];
    $lname = $_POST["lname"];
    $favnum = $_POST["favnum"];

    ...
    echo "<h1>Welcome
            $fname!</h1>";

    ...
?>
```

superglobals

# Form Data Extraction Using Superglobals

- $_GET and $_POST **superglobals** (or autoglobals) read an array of *name-value* pairs submitted to the PHP script
- Superglobals are **associative arrays** – arrays whose elements are referred to with an alphanumeric key instead of an index number

  e.g. `$favnum = $_POST["favnum"];`

  Alphanumeric "Key" instead of an index number

- Are always accessible, regardless of scope

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

# Using Superglobals (continued)

- **$_GET** is the default method for submitting a form
- **$_GET** and **$_POST** allow you to access the values sent by forms that are submitted to a PHP script
- **GET method** appends form data as one long string to the URL specified by the **action** attribute
  - typically used for ***get*** information from a resource
    e.g. getting a record from a database
- **POST method** sends form data in the body of the HTTP request, not visible in the URL
  - typically used for ***creating*** a resource
    e.g. creating a new record in a database



# More Superglobals

- Superglobals contain client, server, and environment information that you can use in your scripts

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

| 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 the 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 that are defined with global scope |

# Using Superglobals (continued)

```
echo "<p>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"], "</p>";
```

Associative array of pre-defined elements (in capitals)



# Using Superglobals (Example 2)

- Given the following registration form

```
<body>
<h1>Log In Form</h1>
<form method="post" action="storeName.php">
  <p><label for="uname">Name</label>
  <input id="username" type="text" name="username"></p>
  <p><label for="uemail">Email</label>
  <input id="uemail" type="email" name="useremail"></p>
  <p><input type="submit" value="Log In" /></p>
</form>
</body>
```

form control *name values* will become *key index* for the superglobal associative array

**Log In Form**

Name

Email

Log In

# Using Superglobals (Example 2)

- In the file **storeName.php**, data is extracted via superglobal **$_POST**, because form **method="post"**

Any preferred variable name

Name from the input form

```
...
$u_name = $_POST['username'];
$u_email = $_POST['useremail'];
echo "<p>User name: $u_name<br/>";
echo "Email: $u_email</p>";
...
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Checking Form Data at the Server

- Always check/validate data at the server:
  - Maintain integrity of the server data
  - Help prevent malicious attack – e.g. SQL injection
- Check that GET or POST has been entered
- Validate data formats
- Cleanse entered data

See also http://www.w3schools.com/php/php_form_validation.asp

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Checking GET or POST data exists

- Use the **`isset()`** function to ensure that a variable is set before you attempt to use it

```
<?php
    if (isset ($_POST["fname"]))
        $fname = $_POST["fname"];
    else
        echo "<p>Error: Please enter data in the
        <a href=\"php_form1.php\">form</a><p>";
?>
```

Assign data to local variable if it is set

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Validating data formats – e.g. strlen

```
<$php
    if (isset ($_POST["fname"])) {
        $fname = $_POST["fname"];
        $err_msg = "";   // set the message to have no value
        if (strlen ($fname) == 0 ) {        // Look for data that is wrong
            $err_msg .= "<p>Error: enter first name.</p>";
        }
        if ($err_msg == "") {               // Proceed if nothing is wrong
            echo "<h1>Welcome $fname!</h1>";
        } else {        // Display error message, if data validation fails
            echo $err_msg;                  }
    } else
        echo "<p>Error: Please enter data in the form</p>";
?>
```

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Validating data formats – RegExp

```
<$php
    if (isset ($_POST["fname"])) {
        $fname = $_POST["fname"];
        $err_msg = "";
        if (!preg_match("/^[a-zA-Z ]*$/",$fname)) {
            $err_msg .=
              "<p>Error: Only letters and spaces allowed.</p>";
        }
        …
    }
    } else
        echo "Error: Please enter data";
?>
```

PHP function

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Regular expressions in PHP

int preg_match ( string $pattern , string $subject)

- Performs regular expression match
- Returns 1 if the pattern matches given subject, 0 if it does not, or FALSE if an error occurred.

- For more complex forms of the function see http://php.net/manual/en/function.preg-match.php

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Validating using the filter_var function

- filter_var()  filters a variable, predefined filters
- Returns the filtered data, or FALSE if the filter fails, e.g.

```
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        $err_msg .=  "Invalid email format";
}
```

- Predefined filters for validating
  - email, types, ip addresses, URLS, …
- Filters also available for sanitising data

http://php.net/manual/en/function.filter-var.php

Pre-defined filter


# Sanitising  data

- Because code can be mixed with HTML, form data can be vulnerable to ‘code injection’.
- Help prevent this by making sure there are no control characters in the data sent to a PHP script.
- Use a small function  like:

```
function sanitise_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
```

Remove leading or trailing spaces

Remove backslashes in front of quotes

Converts HTML control characters like < to the HTML code **<**

# Ex: Sanitising data before processing

```
<$php
    function sanitise_input($data) {
        $data = trim($data);
        $data = stripslashes($data);
        $data = htmlspecialchars($data);
        return $data;
    }
    if (isset ($_POST["fname"])) {
        $fname = $_POST["fname"];
        $fname = sanitise_input($fname);
        if (!preg_match("/^[a-zA-Z ]*$/",$fname)) {
... } ...?>
```

Call the function



# 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.