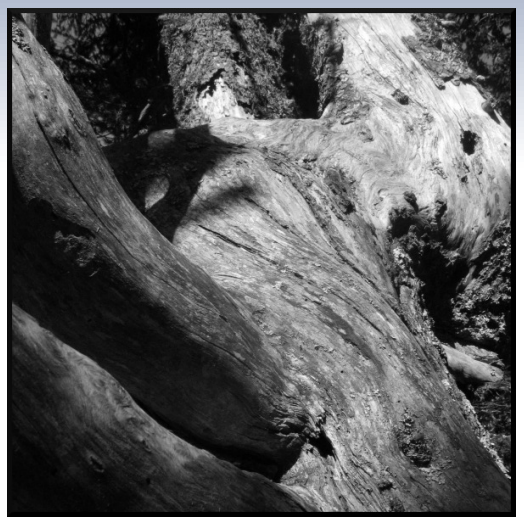


PHP

Jing Hua Ye

Cork Institute of Technology

Sep 13, 2018

Overview

Indexed arrays

- An indexed array (or just array) is a collection of of storage locations, indexed by position (starting from 0)
- To create an indexed array in PHP

Example

```
$rainfalls = array(3, 2, 2, 1, 3);
```

- Another way to create an indexed array

Example

```
$rainfalls[0] = 3;
$rainfalls[1] = 2;
$rainfalls[2] = 2;
$rainfalls[3] = 1;
$rainfalls[4] = 3;
```

Indexed arrays

- You can also create an empty array

Example

```
$toppings = array();
```

- To insert more values into the end of an existing indexed array

Example

```
$toppings[] = 'mushrooms';
$toppings[] = 'anchovies';
$toppings[] = 'chocolate';
$toppings[] = 'baked beans';
```

Indexed arrays

- To overwrite a value in an existing array

Example

```
$toppings[1] = 'fried egg';
```

- To access a value in an array

Example

```
$my_favourite = $toppings[0];
echo "My favourite is {$my_favourite}. ";
echo "My second favourite is {$toppings[3]}.";
```

Non-consecutive indexes

- In PHP, indexed arrays may even have non-consecutive indexes

Example

In a class test, two people scored 4 out of 10; three people scored 8 out of 10:
$frequencies[4] = 2;
$frequencies[8] = 3;

- What happens if you access one of the 'missing' elements?
- Answer: The element is NULL, just like any uninitialized variable
- **Class exercise:** So what will happen here?

```
echo "The number of students who scored 5 out of 10 is: {$frequencies[5]}";
```

Doing things to each value in an indexed array

foreach statement

```
foreach ($a as $v)
{ zero, one or more statements
}
```

- It visits each element in array $a in turn
- For each element, it executes the statements in the curly braces
- Within these statements, it refers to the element as $v

Class exercise: what is the output of the following?

```
echo "<ul>";
foreach ($toppings as $topping)
 echo "<li>{$topping}</li>";
echo "</ul>";
```

Observations

- Array size
  - In most programming languages, arrays are fixed length
  - But in PHP, arrays can grow (and shrink) as we need
- Element type
  - In many programming languages, the elements must all be of the same type, e.g. all integers, all strings
  - In PHP, the elements can be a mixture of types . . . but we'll make only limited use of this flexibility
- Indexes
  - In most programming languages, the elements are indexed by consecutive integers, starting from 0
  - In PHP, there is a lot more flexibility
- However, arrays in other languages are much more efficient than in PHP

Operators

- The compound assignment operators: +=, −=, ∗=, /=, %=, and .=
- Post-increment vs Pre-increment and Post-decrement vs Pre-decrement:

Operator	Name	Effect on $x	Value returned
$x++	Post-increment	Incremented	Original
++$x	Pre-increment	Incremented	New
$x−−	Post-decrement	Decremented	Original
−−$x	Pre-decrement	Decremented	New

- **Class Exercise:** Say what $x contains after each statement in this sequence, and give the output:

```
$x = 5;
$x++;
++$x;
echo $x++;
echo ++$x;
echo $x;
```

Built-in functions

- count($a) returns the number of elements in array $a (sizeof is an alias for count)
- array_sum($a) returns the sum of every element in the array
- explode($delim,$str) splits a string into an array. It's handy for getting input from the user and creating an array from it.

Example

```
$string_of_planets = 'mercury venus earth';
$array_of_planets = explode( ' ', $string_of_planets );
```

Built-in Functions

- trim($str)removes spaces from the beginning and end of a string

Example

```
$string_of_planets = ' mars jupiter saturn ';
$trimmed_string_of_planets = trim( $string_of_planets );
```

- strlen($str) returns the length of (number of characters in) string $str

Example

```
$how_many = strlen( 'uranus' ) + strlen( '' ) + strlen( ' ' );
echo $how_many;
```

Built-in Functions

- isset($v)returns true if $v is non-null; false otherwise
- in_array($val, $array) checks if $val is in an array called $array
- min($array) returns the smallest number in the array
- max($array) returns the largest number in the array

Example

- The user types a sentence into this form:

```
<form action="word_count.php" method="get">
 <input type="text" name="sentence" />
 <input type="submit" />
</form>
```

- Here is word_count.php, which tells the user how many words s/he entered:

```
$sentence = trim( $_GET['sentence'] );
if ( $sentence != '' )
{ $array_of_words = explode( ' ', $sentence );
  echo '<p>You typed ' . count( $array_of_words ) .
  ' words.</p>';
}
```

Class Exercise

- The user types a sentence into this form:

```
<form action="no_space_count.php" method="get">
 <input type="text" name="sentence" />
 <input type="submit" />
</form>
```

- Complete non_space_count.php, which tells the user how many characters s/he entered, excluding the spaces between words.
- E.g. if s/he enters fish and chips, the program echoes 12

Associative arrays

- An associative array (sometimes called a map) is a collection of storage locations, indexed by keys (strings)
- To create an associative array in PHP

Example

```
$fruit_prices = array('Apples' => 1.59, 'Pears' => 2.34, 'Kumquats' => 4.05,
'Jujubes' => 2.34);
```

- Another way to create an associative array

Example

```
$fruit_prices['Apples'] = 1.59;
$fruit_prices['Pears'] = 2.34;
$fruit_prices['Kumquats'] = 4.05;
$fruit_prices['Jujubes'] = 2.34;
```

Accessing the elements in an associative array

- To insert a new key and value into an existing associative array

Example

```
$fruit_prices['Mangoes'] = 5.26;
```

- To modify the value for an existing key in an associative array

Example

```
$fruit_prices['Kumquats'] = 3.93;
```

- To access a value in an array

Example

```
$pear_price = $fruit_prices['Pears'];
echo "Pears cost {$pear_price} ";
echo "but mangoes cost {$fruit_prices['Mangoes']}";
```

Doing things to each value in an associative array

foreach statement

```
foreach ($a as $k ⇒ $v)
{   zero, one or more statements
}
```

- It visits each element in array $a in turn
- For each element, it executes the statements in the curly braces
- Within these statements, it refers to the key of each element as $k and its value as $v

Class exercise: what is the output of the following?

```
echo "<table>";
foreach ($module_lecturers as $module ⇒ $lecturer)
{ echo "<tr>";
  echo "<td>{$module}</td>";
  echo "<td>{$lecturer}</td>";
  echo "</tr>";
}
echo "</table>";
```

Write a PHP script that outputs an unordered list of months that have exactly 31 days

Doing things to each value in an indexed and associative arrays

- The following is mostly for indexed arrays:

```
foreach ($a as $v)
{ zero, one or more statements
}
```

  However, it can be used on associative arrays too

- The following is mostly for associative arrays:

```
foreach ($a as $k ⇒ $v)
{ zero, one or more statements
}
```

  However, it can be used on indexed arrays too

Class Exercise

Hence, what does this output?

```
$toppings = array('mushroom','anchovies','chocolate','beans');
echo "<ul>";
foreach ($toppings as $index => $topping)
 echo "<li>{$index}: {$topping}</li>";
echo "</ul>";
```

And this?

```
$month_length = array('Jan' => 31, 'Feb' => 28, 'Mar' => 31,
'Apr' => 30, 'May' => 31, 'Jun' => 30, 'Jul' => 31,
'Aug' => 31, 'Sep' => 30, 'Oct' => 31,'Nov' => 31);
echo "<ul>";
foreach ($month_length as $len)
 echo "<li>{$len}</li>";
echo "</ul>";
```

Multidimensional arrays

- In an indexed array,
  - elements are indexed by integers
  - elements may be integers, floats, strings, Booleans,... and even arrays
- In an associative array,
  - the keys are strings
  - the values may be integers, floats, strings, Booleans,... and even arrays
- An array that contains other arrays is called a multidimensional array

2-dimensional arrays

Hugh Jeegoh is studying Alchemy at the National University of Nerdland and uses a multidimensional array to record his coursework grades in modules AL1101, AL1102 and AL1103 respectively:

```
$hughs_grades = array(
    array(72, 75, 91, 60),
    array(85, 0, 0, 0),
    array(57, 57, 57, 57)
);
```

Ann O Domini stores her grades, like Hugh does, but she realises she gets a more self-explanatory representation if she uses an associative array of indexed arrays:

```
$anns_grades = array(
    'AL1101' ⇒ array(100, 95, 90, 85),
    'AL1102' ⇒ array(50, 51, 51, 52),
    'AL1103' ⇒ array(60, 55, 60, 65)
);
```

Built-in functions

- count($a) (and sizeof) can be used on associative arrays as well as indexed arrays
- array_key_exists($k, $a)returns true if $k is one of the keys in associative array $a, otherwise returns false
- isset($variable):
  Rather than if (array_key_exists('Jujubes', $fruit_prices))...
  PHP programmers are more likely to write if (isset($fruit_prices['Jujubes']))
  These are the same except
  - the former returns true when $fruit_prices contains a key/value pair where the key is 'Jujubes' and the value is NULL
  - the latter returns false in this case
- array_keys($a)returns an array containing just the keys in associative array $a (usually in the order in which they were inserted)
- array_values($a)returns an array containing just the values in associative array $a (usually in the order in which they were inserted)

Class exercise: what is the output of the following?

```
$month_length = array('Jan' ⇒ 31, 'Feb' ⇒ 28, 'Mar' ⇒ 31,
'Apr' ⇒ 30, 'May' ⇒ 31, 'Jun' ⇒ 30, 'Jul' ⇒ 31,
'Aug' ⇒ 31, 'Sep' ⇒ 30, 'Oct' ⇒ 31,'Nov' ⇒ 31,
'Dec' ⇒ 31);
if (array_key_exists('Mar', $month_lengths))
 echo '<p>Beware the Ides of March</p>';
if ( isset($month_lengths['Mar']) )
 echo '<p>The Ides of March are come</p>';
if ( array_key_exists( 'April', $month_lengths ) )
 echo '<p>April is the cruelest month</p>';
if ( isset($month_lengths['April']) )
 echo '<p>Drip, drip, drop, little April showers</p>';
```

Class exercise: what is the output of the following?

```
$months = array_keys($month_lengths);
foreach ($months as $month)
 echo "<p>{$month}</p>";
foreach (array_values($month_lengths) as $length)
 echo "<p>{$length}</p>";
```

Class exercise

Write PHP that outputs the array as a table, but with one column per key/value

AL1101	WC1101
Hugh Jeegoh	Ann O Domini

$_GET is an associative array

- Suppose form.html contains this form:

```
<form action="process.php" method="get">
 <input type="text" name="firstname" />
 <input type="text" name="surname" />
 <input type="submit" />
</form>
```

- The PHP script, process.php, contains something like this to access the user's data:

```
$firstname = $_GET['firstname'];
$surname = $_GET['surname'];
```

- PHP automatically creates for us an associative array called $_GET

PHP's superglobals

- PHP creates 6 associative arrays for us, referred to as superglobal variables
- (No need to memorize) They contain the EGPCS information (environment, GET, POST, cookies and server), and information about files

$_ENV:	the names and values of any environment variables
$_GET:	information submitted as a GET request
$_POST:	information submitted as a POST request
$_COOKIE:	names and values of any cookies passed as part of a request
$_SERVER:	information about the web server. It contains many keys/values: `http://ie.php.net/manual/en/reserved.variables.server.php`
$_FILES:	information about any uploaded files

Class Exercise

- One is HTTP_USER_AGENT, which gives the string the browser used to identify itself
- Write a PHP script that tries to detect what browser the user is using ('Firefox', 'MSIE' or other) and outputs some messages of your choosing accordingly
- Pretend that there is a built-in function in_string($s1, $s2), which returns true if string s1 is contained within string s2, and false otherwise (Surprisingly, there isn't such a function that is exactly like this)
- Note this script will not be particularly reliable: browsers keep changing the way they identify themselves; some choose not to identify themselves; some firewalls block the browser identity

Defining functions versus invoking functions

- Before it can be used, a function must be defined (or declared)
- Once this has been done, the function can be invoked (or called, or executed, or run)
- Define your own function:
  - Use the keyword function
  - Then give the name of the function: choose a meaningful name that describes the purpose of your function
  - Next indicate the parameters
  - Finally, give the body: a block of code
- The body is not actually executed until you invoke the function

Class exercise: what is the output of the following?

```
<?php
 function square($num)
 {  return $num * $num;
 }
 echo '<table>';
 $result = square(1);
 echo '<tr><td>1</td><td>' . $result . '</td></tr>';
 echo '<tr><td>2</td><td>' . square(2) . '</td></tr>';
 echo '<tr><td>3</td><td>' . square(2+1) . '</td></tr>';
echo '<tr><td>4</td><td>'.square(square(2)).
 '</td></tr>';
 echo '</table>';
?>
```

Formal parameters and actual parameters

- On the previous slide, distinguish between

  formal parameters appear in the function definition (in this case just $x)
  actual parameters appear in function invocation (e.g. 2)

- The formal parameters are just variables
  - when the function is invoked, these variables are created and they are initialised using the actual parameters
  - these variables are used within the body of the function
  - they are destroyed when execution of the body of the function ends
- In the simplest case, a function invocation needs as many actual parameters as there are formal parameters, and in the right order
- The actual parameters should be of the right type (although PHP will do its usual type conversions if they aren't)
- Alternative terminology: formal arguments, actual arguments

The return statement

- When a function is invoked, the function body is executed
- Execution of the function body terminates
  - when the closing brace of the function body is reached, or
  - when a return statement is executed
- At this point, execution of the code that did the invoking is resumed

Advanced Topics

- When you invoke a function, if you supply fewer actual parameters than there are formal parameters, these parameters will remain unset (and a warning will be issued)
- But when you define a function, it is possible for you to supply a default value for a parameter, which will be used if the actual parameter is omitted when the function is invoked
- It is even possible to define functions that accept a variable number of parameters
- It is possible to store functions in variables
- It is possible to pass functions as parameters into other functions
- It is possible to define anonymous functions, especially when you want to pass these into other functions

Class Exercise

- Define a function that returns the volume of a cylinder
  - From geometry, if a cylinder has radius r and height h, then its volume V is given by: $V = \pi r^2 h$
  - Use 3.14 for π
- Write statements that invoke your function to compute and echo the volume of a cylinder whose radius is 10 and whose height is 20, and a cylinder whose radius is 20 and whose height is 10

Class Exercise

- Define a function called output_paragraph, which takes in a string of text and echoes it within an HTML paragraph element
- Modify your output_paragraph function: if the value passed to it is the empty string, then it shouldn't output anything
- Define a function called output_section, which echoes a section containing a heading and some content
- Your function should have the following prototype:
  void output_section(string $id, int $level, string $heading, string $content)
  where
  - $id is the value of the section's id attribute
  - $level is the heading level we want (1-6)
  - $heading is the content of the heading element
  - $content is the rest of the content of the section

Example of invoking output_section

- If you invoke it like this
  output_section('intro', 2, 'Introduction', ' Happy talk, keep talking happy talk. ');
- This is what it will echo:

```
<section id="intro">
 <h2>Introduction</h2>
 <p>Happy talk, keep talking happy talk.</p>
</section>
```

- But without the nice indentation!

Class Exercise

- Define a function called get_larger, which takes in two numbers and uses a two-armed conditional to return the larger of the two
- Define an alternative version that uses a one-armed conditional instead
- Define a third version that uses the ternary conditional operator instead of if

Variables in PHP

formal parameter:
- these are created when the function is invoked
- scope: the function's body

local variables
- variables which are created inside a function
- scope: from the statement in which they are created to the end of the function's body

global variables
- variables which are created outside a function
- scope: from the statement in which they are created to the end of the script but not inside functions

superglobal variables
- these are special PHP variables which you don't create
- scope: the whole script, both inside and outside functions

Using a variable outside its scope

- So what happens if you try to use a variable outside of its scope?
- In some programming languages, you get a fatal error message (either at compile-time or run-time, depending on the programming language)
- PHP gives no error/warning message: it simply assumes that this must be a different variable (although there may be a warning that this new variable is uninitialized)

Class exercise: what is the output of the following?

```
function func1()
{ echo $x;
}
$x = 10;
func1();
```

Class exercise: what is the output of the following?

```
function func2()
{ $x = $x + 1;
}
$x = 5;
func2();
echo $x;
```

Class exercise: what is the output of the following

```
function larger($x, $y)
{ $maximum = $y;
  if ($x > $y)
   $maximum = $x;
  return $maximum;
}
echo '<p>';
echo larger(3, 2);
echo ' ';
echo $maximum;
echo '</p>';
```

Advanced Topic

- Look up the keyword global
  - it turns a local variable into a global variable
  - generally avoid it: it makes it harder to reuse your functions in other scripts because they are tied into the current script
- Look up the keyword static
  - it enables a local variable to retain its value between invocations
  - generally avoid: it makes your code harder to understand (it means that functions can have side-effects and so multiple invocations even with the same actual parameters may give different results)

Common Error

Class Exercise: What is the output of the following?

```
function increase_price($product_prices, $increase)
{ foreach($product_prices as $product ⇒ $price)
   $product_prices[$product] = $price * (1 + $increase);
}

$fruit_prices = array('Apples' ⇒ 1.59, 'Pears' ⇒ 2.34,
'Kumquats' ⇒ 4.05, 'Jujubes' ⇒ 2.34);

increase_prices($fruit_prices, 0.1);

echo '<ul>';
foreach($fruit_prices as $fruit ⇒ $price)
 echo "<li>{$fruit}: {$price}</li>";
echo '</ul>';
```

Simpler example of the same error

Class Exercise: What is the output of the following?

```
function celebrate_birthday($age)
{ $age++;
}

$my_age = 20;
celebrate_birthday($my_age);
echo $my_age;
```

Pass-by-value

- As we know, when a function is invoked, temporary variables are created for each formal parameter
- By default, copies of the values of the actual parameters are placed into these temporary variables in order to initialise them
- This is known as pass-by-value (or sometimes 'call-by-value')
- In the lecture, we will work through this example to illustrate pass-by-value

```
function square($x)
{ $x * $x;
}
$v = 3;
echo '<p>' . square($v) . '</p>';
```

- It follows that in pass-by-value any changes a function makes to its formal parameters have no effect on the corresponding actual parameters (because the function is working on copies!)

Pass-by-reference

- In PHP, when actual parameters are variables it is possible for the programmer to request that their addresses, rather than their values, are used to initialise the formal parameters
- This is known as pass-by-reference (or sometimes 'call-by-reference')
- You request this in the function's parameter list by prefixing an ampersand (&) on the formal parameter

```
function celebrate_birthday(&$age)
{ $age++;
}
$my_age = 20;
celebrate_birthday($my_age);
echo $my_age;
```

- It follows that in pass-by-reference any changes a function makes to its formal parameters do effect the corresponding actual parameters (because the addresses 'point' to the originals)

When to use pass-by-reference

- Advice: use sparingly
- When your function needs direct access to a variable in order to modify its value
  - this is a classic case to use pass-by-reference
  - but we saw an alternative: return the modified value
- When your actual parameter is a very long string or very large array
  - using pass-by-reference saves time and memory
  - if we use pass-by-value, the whole string/array would need to be copied into the formal parameter
- In some languages, such as PHP, the programmer chooses between pass-by-value (default) and pass-by-reference (e.g. by using &)
- In a language such as Java, the decision is taken out of the programmers hands

Advancced Topics

- By default in PHP, return values are copied out of a function
- By prefixing the function name with &, the address of the value is returned
- This facility might be useful if the return value is a long string or large array
- But in fact PHP has certain optimisations ('copy-on-write') that mean that this facility is rarely necessary

Functions for headers and footers

- We could define functions to output the header and footer
- Put these function definitions in a separate file called, e.g., output_functions.php
- Each web page in our web site is changed as follows:
  - it is no longer a static web page (.html)
  - it is now a dynamic web page (.php)
  - it switches into interpret mode and uses the PHP command require_once to load the file of functions
  - at some point (early), it switches into interpret mode and invokes the output_header function
  - at some point (late), it switches into interpret mode and invokes the output_footer function

Definition of a simple output_header function

```
function output_header( $title, $stylesheet )
{ echo '<!DOCTYPE html>
        <html lang="en">
        <head>
        <meta charset="utf-8" />';
   echo "<title>{$title}</title>";
   echo "<link rel=\"stylesheet\" href=\"{$stylesheet}\" />";

   echo '</head><body>';
   echo "<h1>{$title}</h1>";
}
```

Example of a typical web page, e.g. alchemy.php

```
<?php
 require_once('output_functions.php');
 output_header('Department of Alchemy', 'nerdland.css' );
?>

<p> Welcome to the Department of Alchemy at the National
University of Nerdland! We're great, we are. </p>

<?php
 output_footer( 'National University of Nerdland' );
?>
```

Libraries

We could add lots more useful functions for outputting common HTML elements into the output_functions.php file to turn it into a useful 'library'

- E.g. we could include output_paragraph
- E.g. we could include output_submit_button to echo submit buttons for when the programmer is producing a form
- **Class exercise:** Define output_reset_button that will echo reset buttons
- **Class exercise:** Define output_textfield that will echo text fields. Your function could have the following prototype:
  void output_textfield($id, $label, $name, $size, $maxlength, $value, $is_disabled)
- Others, e.g.: output_unordered_list, ...

Libraries

```
function output_submit_button($button_label)
{ echo "<input type=\"submit\" name=\"submit\"
        value=\"{$button_label}\" />";
}
function output_paragraph($text)
{ echo "<p>{$text}</p>";
}
```

Advantages of functions and libraries

- Easier programming
  - coincides with the idea of breaking a problem down into more easily-solvable subproblems
- Greater readability
  - detail is encapsulated within well-named functions
  - the whole script may now be shorter (less repetition)
- Greater maintainability
  - due to greater readability
  - with less repetition, less risk of inconsistent changes

Advantages of functions and libraries

- Greater testability
  - each function can be independently tested and debugged
  - with less repetition, less code to test
- Greater re-usability
  - by using libraries, the above advantages are spread across many programs that you write
  - by using libraries, you have building blocks, so you can avoid 'reinventing the wheel'

Some built-in functions

- is_null
- empty
- is_int
- is_integer
- is_long
- is_float
- is_double
- is_real
- is_numeric

Checking firstname

```
<form action="process.php" method="get">
 <input type="text" name="firstname" maxlength="30" />
 <input type="text" name="surname" maxlength="30" />
 <input type="submit" />
</form>
<?php
 require_once('output_functions.php');
 output_header('Now I know all about you', 'stylesheet.css');

 $error = '';
 if (!isset($_GET['firstname']))
  $error = "Firstname is required";
 output_footer('CIT');
?>
```

Checking Firstname and Using Functions

- First we need to check whether the user has bypassed the form and left firstname out of the URL
- **Class Exercise:**Now check that the user hasn't left the field wholly blank or typed only spaces
- **Class Exercise:**Finally check that the user hasn't exceeded 30 characters
- After checking firstname, we need to check surname in a similar way
- Rather than repeat all this code, the obvious thing to do is define a function
- If there is an error, the function puts an error message into the $errors array
- If there are no errors, it returns the value that the user entered into the text field
- We can place this function into a library, validation_functions.php

A function that checks required text fields

```
function get_required_string(&$user_data, $name, $label, $maxlength, &$errors)
{ if (!isset($user_data[$name]))
   {   $errors[$name] = "{$label}is required";
       return NULL;
   }
   $value = trim($user_data[$name]);
   if ($value == '')
   {   $errors[$name] = "{$label}is required";
       return NULL;
   }
   if (strlen($value) > $maxlength)
   {   $errors[$name] = "{$label}must be {$maxlength}characters or less";
       return NULL;
   }
   return $value;
}
```

Rewritten version of the example PHP script from earlier

```
<?php
 require_once('output_functions.php');
 require_once('validation_functions.php');

 output_header('Now I know all about you', 'stylesheet.css');

 $error = array();
 $firstname = get_required_string($_GET, 'firstname', 'Firstname', 30, $errors);
 $firstname = get_required_string($_GET, 'surname', 'Surname', 30, $errors);

 if (count($errors) > 0)
 {   output_paragraph('You have errors!');
     output_unordered_list(array_values($errors));
 }
 else
  output_paragraph("Hello {$firstname}{$surname}");

 output_footer('CIT');
?>
```

A library for validating user data

- Take a look at the other functions in this library
- It contains functions to check all sorts of things, e.g. that the user has entered an integer in a certain range, or a float in a certain range, ...

```
$error = array();
$age = get_required_int($_GET, 'age', 'Age', 3, $errors, 0, 120);
$weight_in_kg = get_required_float($_GET, 'weight', 'Weight', $errors, 0,500);

if (count($errors) > 0)
{
}
else
{
}
```

For loop: Class Exercises

Same syntax as Java for loop.
The example is a for-loop where $i counts from 0 to 4 inclusive but stops when $i is 5 (hence the body is executed 5 times)

```
for ($i = 0; $i < 5; $i++)
{ echo "<p>{$i}</p>";
}
```

1. Write a loop in which $i counts from 0 to 99 inclusive. How many times is the loop body executed?
2. Write a loop in which $j counts from 0 to 100 inclusive. How many times is the loop body executed?
3. Suppose variable $num_exams contains the number of exams a student must sit. Write a loop that executes its loop body exactly $num_exams times

Class Exercise

- Suppose exercise.html contains this form:

```
<form action="exercise.php" method="get">
  <input type="text" name="number" />
  <input type="submit" />
</form>
```

  Write exercise.php, which outputs (as an unordered list) a multiplication table ('times table') from 1 up to 10 for the number that the user enters

- For example, if the user enters 5, the output is
  - 1 times 5 is 5
  - 2 times 5 is 10
  - ...
  - 10 times 5 is 50
- Using the same form as in the previous exercise (exercise.html), write exercise.php, which outputs the sum of all the integers from 1 up to the number the user entered (inclusive)
- For example, if the user enters 5, the output is 15 (i.e. $1 + 2 + 3 + 4 + 5$)

Off-by-one errors

The most common logic errors with for-loops are off-by-one errors, where the programmer writes a loop that executes one too many, or one too few, times

Class Exercise:A programmer wants a loop that executes exactly 5 times. Which of these are correct?

1. for ($k = 0; $k < 5; $k++)
2. for ($k = 0; $k <= 5; $k++)
3. for ($k = 1; $k < 5; $k++)
4. for ($k = 1; $k <= 5; $k++)

Class Exercise

Assuming $m and $n contain non-negative integers and that $n >= $m, how many iterations will there be of the following loops?

1. ```
   for ( $i = $m; $i < $n; $i++ )
     echo " {$i} ";
   ```
2. ```
   for ( $i = $m; $i <= $n; $i++ )
     echo "<p>{$i}</p>";
   ```

The version 2 is also correct but the version 1 is better than this version. Why?

1. ```
   $current_year = (int) date('Y');
   for ( $year = 1948; $year <= $current_year; $year += 4 )
     echo " {$year}was an Olympic year ";
   ```
2. ```
   for ( $year = 1948; $year <= (int)date('Y'); $year += 4 )
     echo "<p>{$year}was an Olympic year</p>";
   ```

Class exercises

1. Write a for-loop that echoes the numbers from 36 to 1 (inclusive) in that order
2. Rewrite the following so that as much code as possible is moved into the for-loop header

```
$number = (int)$_GET['number'];
$total = 0;
for ( $i = 1; $i <= $number; $i++ )
   $total += $i;
echo "<p>The sum of the numbers from 1 to {$number}is {$total}</p>";
```

Class exercises

1. Suppose mult.html contains this form:

```
<form action="mult.php" method="get">
 <input type="text" name="number" />
 <input type="submit" />
</form>
```

   Write mult.php, which outputs (as an HTML table) a set of multiplication tables ('times tables') for the number that the user enters. E.g. if the user enters 12, then your output runs from '1 times 1 is 1' up to '12 times 12 is 144'

2. Write a PHP script that outputs a 'diary' for the current year with one line per day

Other Types of Loop

The while loop, do-while loop, and for loops have exactly the same syntax and underneath operations as Java ones.

Class Exercise

- Suppose shift.html contains this form:

```
<form action="shift.php" method="get">
 <input type="text" name="duration" />
 <input type="submit" />
</form>
```

  This allows the user to enter the duration (secs) of his/her shift at the local factory

- And suppose we have an array which states, for each job the user must carry out, the time it will take (secs) e.g. $job_lengths = array(127, 195, 3012, ...);
- Using a while-loop, write shift.php which says how many of the jobs will get finished during the user's shift, assuming they are tackled in the order given

Class Exercise

Rewrite the following to use a for-loop:

```
$number = (int)$_GET['number'];
$root = 0;
$odd = 1;
while ( $number > 0 )
{  $number -= $odd;
   $root++;
   $odd += 2;
}
echo "<p>The square root is {$root}</p>";
```

Class Exercise

- Suppose interest1.html contains this form:

```
<form action="interest1.php" method="get">
 <input type="text" name="deposit" />
 <input type="text" name="interest_rate" />
 <input type="text" name="num_years" />
 <input type="submit" />
</form>
```

- Write interest1.php which outputs the amount in the user's account if s/he deposits deposit eurines for num_years years at an interest rate of interest_rate%

PHP and databases

- PHP has support for over 15 different database management systems, including MySQL, Oracle
- When you install PHP on your system, you must configure it so that it has the appropriate library of functions available for your database management system
- But the functions in each library have different names
  - the downside is that your script ends up being tied to one particular database management system

Connecting to the database server

- To communicate with a MySQL database server, you open a connection:

```
$dbconnection = mysqli_connect( $host, $user, $password, $dbname );
```

- In our labs,
  - $host will be 'localhost'
  - $user will be your usual id or probably 'root' at home
  - $password will be the password accessing database server
  - $dbname will be the name of your database

Checking that the connection is made

- The function mysqli_connect returns a 'link identifier' or false if a connection cannot be made
- We should check that a connection was made
- Here's how:

```
$dbconnection = mysqli_connect($host,$user,$password,$dbname);
if ( ! $dbconnection )
  die('Unable to connect!');
```

- The die function causes its parameter to be echoed and then causes execution of the script to terminate

Creating a query and executing it

- The query is written in SQL, and stored as a string in a PHP variable:

```
$sql = "SELECT * FROM birds;";
```

- Then it is executed using the mysqli_query function:

```
$dbresult = mysqli_query( $dbconnection, $sql );
```

- The function returns false if the query fails:

```
if ( ! $dbresult )
  die('Error in query ' . mysqli_error( $dbconnection ));
```

The function mysqli_error returns a string describing the most recent error

Processing the result set

- If the query was successful, the result set has been stored in $dbresult
- There are various things we can now do, e.g. find out how many rows in the result:

```
if ( ! mysqli_num_rows( $dbresult ) == 0 )
  echo '<p>No rows found</p>';
```

- Each time we call the mysqli_fetch_assoc function,
  - it returns a row from the result set and moves onto the next row, or
  - it returns NULL, when there are no more rows
- So, we'll usually want to write some kind of loop:

```
if ( mysqli_num_rows( $dbresult ) == 0 )
  echo '<p>No rows found</p>';
else
  while ($row = mysqli_fetch_assoc($dbresult))
    do something with $row
```

Processing the result set, continued

- The function mysqli_fetch_assoc returns each row as an associative array

Example

after the first time it is called in the example above, $row might contain: ('bnum' ⇒ 1, 'name' ⇒ 'Stevie', 'species' ⇒ 'swan')

- So we can use $row['bnum'], $row['name'] and $row['species']:

```
if ( mysqli_num_rows( $dbresult ) == 0 )
   echo '<p>No rows found</p>';
else
   while ($row = mysqli_fetch_assoc($dbresult))
      echo "<p>{$row['bnum']}{$row['name']}{$row['species']}</p>";
```

Freeing the space and closing the connection

- The result set ($dbresult) may occupy a lot of memory. Once you've finished with the results of the query, you can free the memory:

```
mysqli_free_result( $dbresult );
```

- If no further queries are to be run, you can close the connection:

```
mysqli_close( $dbconnection );
```

- You don't have to do either of these: the memory will be freed and the connection closed automatically when the script terminates

Inserting values

Imagine that $snum, $bnum and $donation contain values, e.g. from a form

```
// Try to connect to database
$dbconnection = mysqli_connect($host,$user,$password,$dbname);
if ( ! $dbconnection )
   die('Unable to connect!');

// Insert into the database
$sql = "INSERT INTO donations (snum, bnum, donation)
VALUES ({$snum}, {$bnum}, {$donation});";
$dbresult = mysqli_query( $dbconnection, $sql );
if ( ! $dbresult )
   die('Error in query ' . mysqli_error( $dbconnection ));

// Close the connection
mysqli_close( $dbconnection );
```

Big head

- A person's hat size is obtained by measuring the circumference of the head just above the ears and dividing by π
- Up to now, a program to take in circumferences and output hat sizes would consist of two files, e.g.:
  - head.html: an HTML web page that contains a form

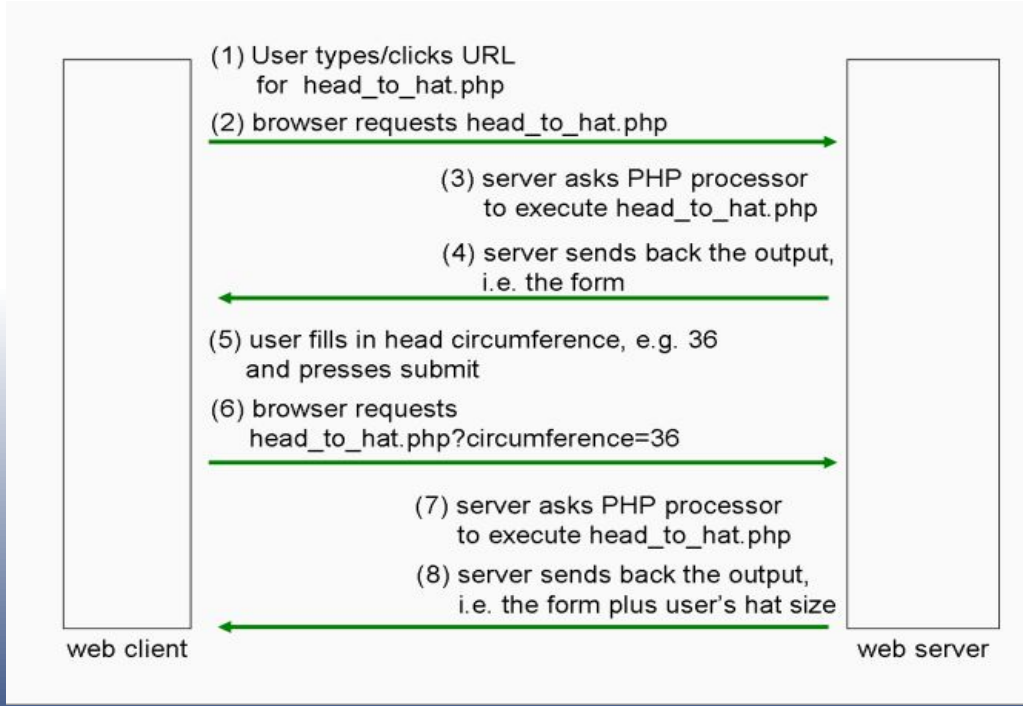
```
<form action="hat.php" method="get">
 <input type="text" name="circumference" />
 <input type="submit" />
</form>
```

  - hat.php: a PHP script that receives the data, carries out the calculation, and echoes the result into a new web page

Self-processing page

We can instead write this using a self-processing page:

- a single PHP script, e.g., head_to_hat.php
- when requested without user data, it outputs the form
- when requested with user data, it receives the data, carries out the calculation, and outputs the form again but with the result included



A useful function

- It is worth defining a function to output the form.
- Note the trick for putting the name of this file as the value for the action attribute

```
function output_form()
{   echo "<form action=\"{$_SERVER['PHP_SELF']}\" method=\"get\">";
    output_textfield('circumference', 'Circumference: ', 'circumference', 4, 4, '',
false);
    output_submit_button('Submit');
    echo "</form>";
}
```

Another useful function

- We need to be able to distinguish whether the user is requesting the form for the first time (hence, not submitting any data) from the case where s/he has seen the form, filled it in, pressed submit, and sent us data
- Here, as a function, is one of the many ways of checking for this (how it works will be explained in the lecture):

```
function is_initial_request()
  return ! isset($_GET['submit']);
```