Systems Programming & Scripting

Lecture 18: Control Structures

PHP Control Structures

- If statements:

if (expression)
  statement
else
  statement

- Blocks of statements are grouped by { … }
- Alternatively, use : after else and use a final *endif*
- The latter is useful for mixing PHP code with HTML data

Example: If statement

```
<?php
if($x > $y){
    echo ‘x is greater than y’;}
else if($x  < $y) {
    echo ‘x is less than y’;}
else{
    echo ‘x is equal to y’;}
?>
```

Example: If statement

```
<?php
if($x > $y) :
      echo "x is greater than y";
else :
 if ($x  < $y) :
     echo "x is less than y";
 else :
      echo "x is equal to y";
 endif;
endif;
?>
```

Switch Statement

```
switch (expression) {
  case value:
     statement
     break;
  …
  default:
statement
break
}
```

While Loop

while (expression)

statement

- *Or*

while (expression) :

statement

endwhile;

- *break* and *continue* work as usual

Example: sum-over-array

```
$arr = array(1,2,3,4,5);
$s = 0;
$i = 0;
while ($i < count($arr)) {
   $s += $arr[$i];
   $i++;
}
echo "<p>Sum over array 1..5: $s";
$x = array_sum($arr);
if ($x==$s) {
   echo "<p>The result is correct";
} else {
   echo "<p>The result is wrong";
}
```

do/while loop

do

statement

while (expression)

- The loop is executed at least once

For loop

for (start; condition; increment)

statement

- start, condition, increment can be arbitrary expressions
- Semantics is the same as in C#
- Alternative syntax with : and endfor can be used

Foreach loop

foreach ($array as $value)

statement

- statement is executed for each value in $array in turn being assigned to $value

foreach ($array as $key => $value)

statement

- *statement* is executed for each $key to $value mapping.

Example: sum-over-array

```
$arr = array(1,2,3,4,5);
$s = 0;
for ($i = 0; $i < count($arr); $i++)
  $s += $arr[$i];

echo "<p>Sum over array 1..5: $s";
```

Iterator functions

- Iterator functions generalise loops over arrays:
- current() … returns the current array element
- key() … returns the key of the current element
- reset()/end() … moves the iterator to the start/end of the array
- next()/prev() … moves the iterator to the next/previous element
- each() … return the next key-value pair in the array

Example: sum-over-array

```
$arr = array(1,2,3,4,5);
$s = 0;
while (list($key, $value) =
        each($arr))
  $s += $value;
```

Iterators over classes

- Iterators can also be used over arbitrary classes.
- Iteration is over all properties of the class.
- Eg. the following code, prints the values of all properties in a newly allocated cart:

```
$cart = new Cart();
foreach($cart as $key => $value) {
    print "$key => $value\n";
}
```

Serialisation

- PHP provides functions to `serialise()` and `unserialise()` arbitrary PHP datatypes.
- This can be used to store data-structures in a file, or to transfer them between machines.
- To unserialise, a definition of the class must be available.

for/foreach

- Foreach is a convenient control structure to iterate through the items of an array.

```
<?php
$fruits = array (“apple” => “My favourite”,
  “banana” => “don’t like”, “pineapple” =>
  “can eat”);
foreach ($fruits as $key => $value) {
      echo "Key: $key; Value: $value<br />\
  n";
  }
?>
```

Useful Operations on arrays

- Extracting elements from an array:

```
list($x,$y,$z) = $array;
```

- Extracting a slice from an array:

```
$slice=array_slice($arr,$offset,$length);
```

- Splitting an array into fixed size chunks:

```
$chunks=array_chunk($arr, $size);
```

- Extracting only keys/values:
  - `array_keys($arr);`
  - `array_values($arr);`
- Checking whether an element exists:

```
array_key_exists($key, $arr);
```

Converting between Arrays and Variables

```
$person = array('name' => 'Fred', 'age' =>
    35, 'wife' => 'Betty');
```

- `extract($person)` generates (local) variables from the keys of this array,ie:

```
$name = 'Fred';
$age = 35;
$wife = 'Betty';
```

- `compact('name', 'age', 'wife')` does the opposite

Higher-order array functions

```
array_walk($arr, $fct);
```

- Applies the function $fct to every array element.

```
array_reduce($arr, $fct, $default);
```

- Combines all elements in $arr, from left to right, by applying $fct, using the value $default for the empty array.

```
usort($arr, $fct);
```

- Sort $arr by a user-supplied $fct.

More array functions

- PHP provides families of functions to treat arrays as
  - Sets
  - Stacks
- For finding an element in an array:

```
in_array($elem, $arr);
```

Including code

include file_name

- Or

require file_name

- Loads the code in file_name; for require the file must exist
- Warnings from loading the code are supressed, if *@include* is used
- If permitted by the *php.in*i configuration file, code can be loaded from (remote) URLs

Example: header and footer

- The following code reads functions, generating a footer and a header for the web page from the library design.inc

```
<?php require 'design.inc';
      header(); ?>
```

content

```
<?php footer(); ?>
```

Functions

- Functions abstract over and parameterise code.
- They are like C# methods.
- As usual, you have to define:
  - The function name
  - The names of the arguments
  - NOT: the types of parameters and result

Function Definitions

function [&] function_name ([[&]param [, …]])

statement

- Only names but not types have to be specified.
- A function has 0 or more parameters;
  - *param*: arguments are *passed by value*
  - *&param:* arguments are *passed by reference*
- The body of the code can include HTML text
- An optional & indicates that a reference to data, rather than the data itself, is returned

Example: String concatenation

```
function strcat ($left, $right) {
  $combined = $left . $right;
  return $combined;
}

echo strcat("<p>", strcat("Hello ",
                          "World"));
```

Scope of Variables

- Variables in functions are *local*, i.e. different from variables with the same name defined outside the function.
- The keyword *global* is used to access the variables in the outer scope
- The keyword *static* is used to create a variable that remembers its value after leaving the function

More on Parameter Passing

- Specifying default parameters:

```
function get_preferences($what = “all”){...}
```

- If no argument is provided when calling the function, $what takes the value “all”
- Variable number of parameters:

```
function get_preferences ()
```

- And use these functions in the body:

```
$arr = func_get_args();
$count = func_num_args();
$value = func_get_arg(arg_number);
```

Anonymous functions

- To define a nameless function use:

```
create_function(args_string,code_string);
```

- This function can be passed as argument to other functions
- It is possible to use function variables:

```
$myfun();
```

- First evaluates the variable $myfun, which must be a function, and then calls it

Example: Customised sorting

```
// create an anonymous function
$lambda = create_function('$a,$b',
        'return(strlen($a)-strlen($b));');
// array to be sorted
$arr = array('some string', 'yet another
string', 'a string');
// sort array by str-length
usort($arr, $lambda);
// print it
print_r($arr);
```

Objects

- PHP is a full-blown object-oriented language
- The syntax for defining classes, class-hierarchies and object is similar to C#
- A class contains
  - Properties (data)
  - Methods (code)
- The new construct creates an object
- Infix → is used to refer to a property/method

Class Declaration

class class_name [extends base_class] {

[var $property [= value]; …]

[[static] function fct (args) { … }] }

- A class can inherit properties and methods from a base_class
- Properties may be initialised
- Methods can be static, ie once per-class
- Or refer to the object as $this

The usual stuff

- Access modifiers `public, protected, private` can be used to control access to properties
- Properties can be static, ie once per class
- To refer to a static property/method write: *class_name*`::`*property/method* must be used
- To access a property/method from the parent class:

`parent::`*property/method*

- Constants can be declared like this:

`const` *name = value;*

Interfaces

interface interface_name {

[function function_name (); …] }

- Defines the functions that should be supplied through the interface.
- The keyword `implements` is used on class declaration

Abstract Classes/Methods

abstract class class_name {

[abstract function function_name (); …] }

- To collect methods, common to several classes, abstract classes with abstract methods are used. Concrete classes inherit from them as usual

Constructors/Destructors

- The constructor of a class must have the name `__constructor`
- It is executed whenever new is called on the class
- The destructor of a class must have the name `__destructor`
- It is executed automatically when an object is not used any more
- PHP performs automatic garbage collection

Example: Shopping Cart

```
// interface for a general collection
interface MyCollection {
  function add_item($art_nr, $num);
  function remove_item($artnr, $num);
  function show ();
}
```

Example: Shopping Cart

```
class Cart implements MyCollection {
  var $items;
  var $debug = true;

  function add_item($art_nr, $num) {
    $this->items[$art_nr] += $num;
  }

  function show () {
    echo "<table>\n";
    echo "<td>Item</td><td>Quantity</td><tr>\n";
    foreach ($this->items as $key => $value) {
      echo "<td> $key</td> <td> $value </td> <tr>\
n";
    }
    echo "</table>\n";
  }
```

Example: Shopping Cart

```
function remove_item($artnr, $num) {
   if (!array_key_exists($artnr, $this->items)) {
    if ($this->debug) {
     echo "<P>Warning: Trying to remove non
               existing item $artnr<br>\n";
    }
    return false;
   }
   if ($this->items[$artnr] > $num) {
     $this->items[$artnr] -= $num;
     return true;
   } else if ($this->items[$artnr] == $num) {
     unset($this->items[$artnr]); // remove entry
     return true;
   } else {
     return false;
   }
  }
}
```

Example: Shopping Cart

```
$another_cart = new Cart();
$another_cart->add_item("0815", 3);
$another_cart->add_item("13", 1);
$another_cart->add_item("99", 9);
if ($another_cart != null) {
  echo "You have a second cart, with this
        contents:<br>";
  $another_cart->show();
}
$another_cart->remove_item("99", 8);
$another_cart->remove_item("0815", 3);
$another_cart->remove_item("55", 1);
if ($another_cart != null) {
  echo "After removing some items the contents
        is:<br>";
  $another_cart->show();
}
```

Exceptions

- PHP provides exceptions very similar to those in C#, eg.

```
function inverse($x) {
    if (!$x) {
        throw new Exception('Division by zero.');
    }
    else return 1/$x;
}

try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage();
}
```

Exceptions

- Try-catch blocks can be nested.
- Several catch-branches may be attached to a try, covering different exceptions.
- Tailored exceptions are defined by extending the pre-defined Exception class:

```
class MyException extends Exception { }
```

Type Hinting

- PHP can provide hints on type information.
- For example arguments to functions may be restricted to certain classes, or to be of an array type.
- This recovers some of the safety from statically typed languages.

```
public function test(OtherClass $otherclass) {
        echo $otherclass->var;    }
```

Namespaces

- PHP 5.3 onwards provides the concept of namespaces, to avoid name-clashes between modules.
- The definition of a namespace must be at the start of the file, eg.

```
namespace ThisModule { … }
```

- Namespaces can be nested, eg.

```
namespace ThisModule\SubModule { … }
```

- To refer to an object, use the notation

```
SubModule\SomeClass::SomeMethod()
```

- Or a fully qualified name

```
\ThisModule\SubModule\SomeClass::SomeMethod()
```

Exercise

- Implement the sum-over-array example using iterator constructs.
- Extend the shopping cart example to additionally store with each article a unit price and a currency. Compute the total costs of all articles in the shopping cart. Implement a function that converts all prices from one currency to another (GBP, EUR, USD).
- Write a PHP script, that reads a data file from the Big MACS store manager of coursework 1, and display the contents of the file in table. The supplier should be highlighted. The user should be able to pick the filename via a form.