Universitat d'Alacant
Universidad de Alicante

Department of Software and Computing Systems

Network Europe - Russia - Asia of
Masters in Informatics as a Second Competence
159025-TEMPUS-1-2009-1-FR-TEMPUS-JPCR

Web Programming with PHP

Sergio Luján Mora
Department of Software and Computing Systems
University of Alicante (Spain)

LANGUAGE SYNTAX

Contents

- Language Features
- Variables
- Arrays
- Text Strings
- Constants
- Operators
- Flow Control Sentences
- Functions

Language Features (I)

- Server code opening and closing tags:

  `Short:<? ... ?>` left angle bracket + question mark … question mark + right angle bracket

  `Normal: <?php ... ?>`

  `XML: <script language="php">...</script>`

  `ASP Style: <% ... %>`, not always available, depending on the interpreter configuration

Language Features (II)

- Comments:

```
/* Comment C-Style
    multiline */
// Comment C++-Style, just one line
# Comment Bash/Perl-Style, just one line
```

Language Features (III)

- End of instruction: semicolon (;)
- Printing strings on screen:

```
echo "text string";
<?="text string"?>
```

- Uppercase and lowercase:
  - Variable names are CASE SENSITIVE:
    `$MyNumber` is different than `$mynumber`
  - Function names and reserved words are CASE INSENSITIVE:
    `PRINT()` is the same function than `print()`

Variables (I)

- PHP is a soft-typed language
- It is not mandatory to declare variables. They are declared automatically first time they appear in our code.
- ALL VARIABLES HAVE A '$' SYMBOL IN FRONT OF THEIR NAME:

```
$var1 = 123;
$var2 = 'hello world';
```

Variables (II)

- All variables are 'mixed' typed. This generic type is similar than 'variant' VBScript type.
- Nevertheless, there are basic types:
  - int, integer → Numeric Integer. 0NNN 8-based or octal, 0xNN 16-based or hexadecimal.
  - float, double, real → Numeric Floating point
  - array, string, object

Variables (III)

- Type examples:
  - Integers: decimal, octal or hexadecimal:
    ```
    $Var = 123;
    ```
  - Floating point:
    ```
    $Var = 1.3e4;
    ```
  - Arrays or vectors.
    ```
    $Var[2] = 123;
    ```
  - Text Strings:
    ```
    $Var = "A Text String\n";
    ```
  - Objects:
    ```
    $Var = new oMyClass();
    ```

Universitat d'Alacant
Universidad de Alicante
lsi

Variables (IV)

- A variable can be of different type throughout the time
- In order to avoid errors and ambiguities, the PHP interpreter performs the necessary type conversions (*casts*) when working with variables and contents of different types:

```
$num = 123;
echo $num; //num transforms into String
```

- Performing an explicit conversion or cast:

```
$var = (string)123;
```

- We also can change the type of a variable with: `settype()`:

```
$var = 12;
settype($var, double);
```

Universitat d'Alacant
Universidad de Alicante
lsi

Variables (V)

- Variable scope → According to the place where it is declared:
  - Global to a entire file.
  - Local to a function
  - Local to a class or object (class variables or attributes). They can be accessed by means operator '->' (arrow: hyphen-right angle bracket)

Variables (VI)

- If we want to access to a global variable from a function, we use the reserved word 'global'

```
$myglobalvar = 3;

function myfunction() {
   global $myglobalvar; // refers to the
  global var
   echo $myglobalvar; //prints the value of
  the global var
}
```

- If we don't do this way (not to using 'global') we would be declaring and printing the value of a new local variable in our function.

Variables (VII)

- It is possible to define variable aliases: that is, two or more variables pointing to the same data (as if they were pointers)
- Operator '&' (ampersand) let us obtain references to a variable:

```
$myalias = &$myvariable;
```

- `unset()` is used to remove references:

```
unset ($myalias);
```

Variables (and VIII)

- It is possible to access to a variable content (v1) through another variable (v2) that stores the name of the first variable (v1) by means of '$$' (double dollar):

```
$a = 123;
$b = 'a';
echo $$b; // $($b) = $(a) = 123
```

Arrays (I)

- They are declared with square brackets ([]) at the end of the array name.
- Brackets are also used to access to their elements
- Fisrt element has is zero indexed.
- A PHP array can have different types in their elements
- An element can also be accessed by an associative index or key (hash tables)
- PHP allows multidimensional arrays
- There is a Constructor function: `array()`
  - Paramenters: pairs key/value: `key=>value`

Universitat d'Alacant
Universidad de Alicante
lsi

Arrays (II)

- Example, different element types:

```
$vector1[0] = 1;
$vector1[1] = 'hello';
$vector1["name"] = "John";
```

- Example: Using the array Constructor:

```
$vector2 = array (1, "John", 3);
$vector3 = array(
  0 => 1,
  1 => "John",
  "name" => "Peter",
  3 => 5 );
// index => value
```

Universitat d'Alacant
Universidad de Alicante
lsi

Arrays (III)

- Other ways of creating arrays:

```
$a[] = 'a';
$a[] = 'b';
$a[] = 'c';
// these ways are equivalent
$a = array('a', 'b', 'c');
```

Arrays (IV)

- An array index or key can be an integer or string
- An element designed by a string key has not a mapped integer index.
- If an index is omitted, then a new index is automatically generated, starting from 0.
- When generating a new index, if numeric, this one will be next intenger index + 1.
- If we create a new element with an existing index or key, last one will overwrite the previous.

Arrays (V)

- Example:

```
$firstquarter = array(1 => 'January',
   'February', 'March');

print_r($firstquarter);
```

- Output:

```
Array ( [1] => January [2] => February [3] =>
   March )
```

This is an array starting from 1, nor from 0.

Arrays (VI)

- Example:

```
$fruits = array (
  "fruits"  => array("a" => "orange", "b" => "banana", "c" =>
   "apple"),
  "numbers" => array(1, 2, 3, 4, 5, 6),
  "holes"   => array("first", 5 => "second", "third")
);

print_r($fruits);
```

- Output:

```
Array (
  [fruits] => Array ( [a] => orange [b] => banana [c] => apple )
  [numbers] => Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5
   [5] => 6 )
  [holes] => Array ( [0] => first [5] => second [6] => third ) )
```

This is a multidimensional array

Arrays (VII)

- Example:

```
$array = array(1, 1, 1, 1,  1, 8 => 1,  4 => 1,
   19, 3 => 13);

print_r($array);
```

- Output:

```
Array ( [0] => 1 [1] => 1 [2] => 1 [3] => 13 [4]
   => 1 [8] => 1 [9] => 19 )
```

Remarks:

- Element '3' overwrite its value from '1' to '13', because it is redefined.
- Value '19' is allocated in element '9', because it is the maximun index (8) plus 1

Arrays (VIII)

- `count($array)`: Count elements in the array '$array'
- `in_array($elem, $array[, $strict])`: Checks if value '$elem' exists in array '$array'. ReturnS TRUE if '$elem' is found in '$array', FALSE otherwise. If parameter '$strict' is set to TRUE, then the function will also check the type of the '$elem' in the array elements.
- Moving along array elements:
  - `reset($array)`: set the internal pointer ot an array to its first element.
  - `current($array)`: Returns the current element in an array. Current means the element the internal pointer points to.
  - `next($array)`: Advace the internal pointer of an array (to the next element)
  - `prev($array)`: Rewind the internal array pointer
  - `end($array)`: Set the internal pointer of an array to its last element.

Arrays (IX)

- `list($var1, ...)`: Assign variables as if they were an array in one operation.
- `each($array)`: Returns the current key and value pair from an array and advance the array cursor
  - This pair is returned in a four-element array, with the keys *0*, *1*, *key*, and *value*. Elements *0* and '*key*' contain the key name of the array element, and *1* and '*value*' contain the data.
- Another way moving along an array:

```
$myarray = array('one', 'two', 'three');
reset($myarray);
while (list($key, $value) = each($myarray))
    echo "$key => $value\n";
```

Universitat d'Alacant
Universidad de Alicante
lsi

Arrays (X)

- `sort($array, $flag)`: Sort and array. The optional second parameter '*$flag*' may be used to modify the sorting behavior using these values:
  - **SORT_REGULAR** - compare items normally (don't change types)
  - **SORT_NUMERIC** - compare items numerically
  - **SORT_STRING** - compare items as strings
  - …
- `explode ($limite, $cadena)`: Splits '$cadena' by a string '$limite' returning an array of string, each of which is an obtained substring.
- `implode($union, $array)`: Joins '$array' elements with a '$union' string. The string '$union' is used as glue among the '$array' elements.

Universitat d'Alacant
Universidad de Alicante
lsi

Arrays (and XI)

- Another functions:
  - `array_diff()`: Computes the difference of arrays
  - `array_fill()`: Fills an array with values
  - `array_reverse()`: Returns an array with elements in reverse order
  - `array_search()`: Searches the array for a given value and returns the corresponding key if successful
  - `array_sum()`: Calculate the sum of values in an array
  - `array_walk()`: Apply a user function to every member of an array

Constants (I)

- `define()` function allows us to declare a constant.
  - `define('myCons', value, noUppercase)`
  - If 'noUppercase' set to TRUE, the constant will be defined case-insensitive .
- It can never be changed or undefined.
- Unlike with variables, you must not prepend a constant with a '$'.

Constants (and II)

- They have global scope, so they can be accessed from everywhere in our PHP script.
- Only scalar data (types boolean, integer, float and string) can be contained in constants
- Example:

```
define ('kHi', 'Hello world!');
echo "Constant kHI is: " . kHi;
```

Operators

- Arithmetic: +, -, *, /, %
- Incrementing/decrementing:

```
$a++, ++$a, $a--, --$a
```

- Bitwise: &(AND), |(OR), ^(XOR), ~(NOT), >>, <<
- Logical: and, or, xor, !, &&, ||
- Comparison: ==, ===, !=, !==, <, >, <=, >=.
- String: . (concatenation), .=

True/False evaluation (I)

- PHP interpreter has defined 2 constants: TRUE and FALSE
  - TRUE is an integer with value 1
  - FALSE is an empty char string

True/False evaluation (and II)

- When working with numeric values, 0 is FALSE and any other value is TRUE.
- With strings, an empty string ("") evaluates to FALSE, and any non-empty string is TRUE.
  - There is an exception: a string with value "0"
- With arrays, FALSE is an array without elements and TRUE in other case.
- When working with objects: an object evaluates to FALSE if it is an empty object (that is, without members nor attributes in its class) and TRUE in other case.

Assignation, equality and identity (I)

- Assignation: =
  - $a = $b, This assigns to $a then value in $b.
- Equality: ==, !=
  - $a == $b, TRUE if $a is equal to $b.
  - $a != $b, TRUE if $a is not equal to $b.
- Identity: ===, !==
  - $a === $b, TRUE if $a is equal to $b, and they are of the same type.
  - $a !== $b, TRUE if $a is not equal to $b, or they are not of the same type.

Assignation, equality and identity (II)

```
if("0" == 0)
  echo "YES";
else
  echo "NO";

if("0" === 0)
  echo "YES";
else
  echo "NO";
```

Assignation, equality and identity (and III)

```
if("0" == 0)
  echo "YES";  ------------------------> YES!
else
  echo "NO";

if("0" === 0)
  echo "YES";
else
  echo "NO";  ------------------------> NO!
```

Universitat d'Alacant
Universidad de Alicante
lsi

More assignation operators

- +=, -=
- *=, /=, %=
- &=, ^=
- .=
- >>=, <<=

Associativity	Operator
Left	,
Left	or
Left	xor
Left	and
Right	print
Left	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
Left	?:
Left	||
Left	&&
Left	|
Left	^
Left	&
Non associative	== != === !==
Non associative	< <= > >=
Left	<< >>
Left	+ - .
Left	* / %
Right	! ~ ++ -- (int) (double) (string) (array) (object) @
Right	[
Non associative	new

Ternary Operator ?:

- It has the same functionality like in C, C++:
  - `(expr1) ? (expr2) : (expr3);`
- Evaluates to 'expr2' if 'expr1' evaluates to **TRUE**, and 'expr3' if 'expr1' evaluates to **FALSE**.
- Example:

```
$str = $a > $b ? "a is greater then b"
  : "a is not greater then b";
```

Error control operator @

- Error control operator: When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored, so the programmer will have to catch the exception and manage the error.
- Example:

```
$myFile = @file('afile.txt')
```

  - If 'file' function makes an error, `'$myfile'` will contain a `null`, but the interpreter will not show the default error message.

Flow control sentences

- Very similar to C, C++, Java and JavaScript sentences:
  - if...elseif...else
  - switch
  - while, do...while
    - break and continue
  - for
- Similar to JavaScript:
  - foreach

if...elseif...else

- It allows for conditional execution of code fragments :

```
if (expression) {
Instructions
}
```

- With Else:

```
if (expression) {
Instructions if true
}
else {
Instructions if false
}
```

if...elseif...else

- With 'elseif'

```
if (expression) {
  Instructions if true
}
elseif (expression 2) {
  Instruction set 2
}
...
else {
  Final Instruction set
}
```

switch...case...default

- Multiple conditional statement:

```
switch (variable) {
    case value 1:
            instructions 1
            break;
    case value 2:
            instructions 2
            break;
    ...
    case value N:
             instructions N
             break;
    default:
             default instructions set
}
```

while and do...while

- Loop sentences

```
while (expression) {
    instructions
}
--------
do {
    instructions
} while (expression);
```

break and continue

- `break;`
  - *break* ends execution of the current *for*, *foreach*, *while*, *do-while* or *switch* structure.
- `break n;`
  - 'n' is a numeric argument which tells it how many nested enclosing structures are to be broken out of.
- `continue;`
  - *continue* is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration.
- `Continue n;`
  - *continue* accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

for (...)

- Similar to C:

```
for (expression1; expression2;
  expression3) {
   instructions
}
```

- The first expression (expr1) is evaluated (executed) once unconditionally at the beginning of the loop.
- In the beginning of each iteration, expr2 is evaluated. If it evaluates to TRUE, the loop continues and the nested statement(s) are executed. If it evaluates to FALSE, the execution of the loop ends.
- At the end of each iteration, expr3 is evaluated (executed).

For: examples

- One:

```
$factorial5 = 1;
for ($i = 2; $i <= 5; $i++ ) {
    $factorial5 *= $i;
}
```

- Two:

```
for ($factorial5 = 1, $i = 2; $i <= 5; $i++ ) {
    $factorial5 *= $i;
}
```

- Three:

```
for ($factorial5=1, $i=2; $i<=5; $factorial5*=$i,
  $i++);
```

Foreach (I)

- New since PHP4!

```
foreach ($array as $variable) {
    instrucciones
}
```

- Loops over the array given by '*$array*'. On each loop, the value of the current element is assigned to *$variable* and the internal array pointer is advanced by one (so on the next loop, you'll be looking at the next element).

Foreach (and II)

- Another form or syntax:

```
foreach ($array as $key => $value) {
    instrucciones
}
```

- This form does the same thing, except that the current element's key will be assigned to the variable *$key* on each loop.
- As of PHP 5, it is possible to iterate objects too.

Functions (I)

```
function myFunction ($arg_1, $arg_2, ...,
  $arg_n) {
    instructions
    return $exitValue;
}
```

- All arguments are passed by value (making a local copy).
- `return` is optional and allows us to give a return value to the caller. This causes the function to end its execution immediately
- PHP does not support function overloading (define two or more functions with the same name)

Functions (II)

- We can declare default argument values, they are solved from right to left:

```
function myFunction ($arg1, $arg2="value")
  {...}
```

- If we want pass arguments by reference, prepend an ampersand (&) to the argument name:

```
function myFuncyion ($arg1, &$arg2) {...}
```

Universitat d'Alacant
Universidad de Alicante
lsi

Functions (and III)

- Variable functions: This means that if a variable name has parentheses appended to it, and PHP interpreter will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.

```
function sayHello_morning() {
   echo "Good Morning"; }
function sayHello_night() {
   echo "Good night"; }
$sTime = "night";
$sFunction = "sayHello_".$sTime;
echo $ sFunction();
```

- This can be used to implement callbacks, function tables, and so forth.