PHP – Personal Home Page PHP: Hypertext Preprocessor (Lecture 35-37)

A Server-side Scripting Programming Language

An Introduction

What is PHP?

PHP stands for "**P**HP: **H**ypertext **P**reprocessor". It is a server-side scripting language. PHP scripts are executed on the server and it is especially suited for Web development and can be embedded into HTML. The main goal of the language is to allow web developers to write dynamically generated web pages quickly. Its syntax is very similar to C/C++ and it has support for almost all major OS, web-servers, and databases. PHP is an open source software (OSS) and is free to download and use.

What is a PHP File?

PHP files may contain text, HTML tags and scripts. PHP files are returned to the browser as plain HTML. PHP files have a file extension of ".php", ".php3", or ".phtml". As mentioned earlier, a PHP script is run on the web server, not on the user's browser. Therefore, client-side compatibility issues are not of concern of the programmer as simply return an HTML document for the browser to process. You need three things to make this work: the PHP parser (CGI or server module), a web server and a web browser.

PHP Hello World

Following is the PHP Hello World program.

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
        <?php // start of php code
        echo '<p>Hello World</p>';
        ?>
  </body>
</html>
```

The code written within "php" tags is parsed by the PHP processor and the rest is ignored. Hence, a PHP parser would convert the above code into the following HTML document and send it to the browser.

```
<html>
```

```
<head>
 <title>PHP Test </title>
</head>
<body>
        <p>Hello World</p>
</body>
</html>
```

It may be noted that PHP supports C++ type of comments and requires a semicolon ";" at the end of each statement.

PHP Opening and Closing Tags

PHP scripts are always enclosed in between two PHP tags. This tells your server to parse the information between them as PHP. The three different forms are as follows:

1.
```
<?php
        echo "this is the recommended style";
?>
```

2.
```
<script language="php">
        echo "this style is also available";
</script>
```

3.
```
<?
        echo "this is short tags style; it needs to be configured";
?>
```

There is also another form which may or may not be supported on some browsers. It is shown as below:

4.
```
<%
        echo 'this is ASP-style tags; it may not be available';
%>
```

Data types

The type of a variable is decided at runtime by PHP depending on the context in which that variable is used. PHP supports the following data types:

- Boolean
- integer
- float or double
- string
- array
- object
- resource
- NULL

Integers

The size of an integer is platform-dependent, although a maximum value of about two billion is the usual value (that's 32 bits signed). 2147483647 = $2^{31} – 1$. PHP does not support unsigned integers. It is important to note that, unlike C++ or Java, if you specify a number beyond the bounds of the integer type, it will be interpreted as a float instead. If an operation results in a number beyond the bounds of the **integer** type, a **float** will be returned instead. Also, there is no integer division operator in PHP. *1/2* yields the **float** *0.5*.

Strings

PHP strings are created using single quote or double quote. They can also be created by <<< which is called heredoc. One should provide an identifier after <<<, then the string, and then the same identifier to close the quotation. The closing identifier *must* begin in the first column of the line.

Variable

Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is case-sensitive. A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'.

Note: For our purposes here, a letter is a-z, A-Z, and any ASCII characters from 127 through 255 (0x7f-0xff). Therefore, $Inzi½ is a legal variable name. It is not necessary to initialize variables in PHP. Un-initialized variables have a default value of their type - FALSE, zero, empty string or an empty array.

String conversion to numbers

When a string is evaluated as a numeric value, the resulting value and type are determined as follows.

- The string will evaluate as a **float** if it contains any of the characters '.', 'e', or 'E'. Otherwise, it will evaluate as an integer.
- The value is given by the initial portion of the string. If the string starts with valid numeric data, this will be the value used. Otherwise, the value will be 0 (zero).

Here is an example:

```
<?php
        $foo = 2 + "10.5";                  // $foo is float (12.5)
        $foo = 1 + "-1.1e3";            // $foo is float (-1099)
        $foo = 1 + “Ali-1.3e3";             // $foo is integer (1)
        $foo = 1 + "10 Small Cats";   // $foo is integer (11)
        $foo = "10.0 cats " + 1;        // $foo is float (11)
?>
```

Using strings in numeric expressions provides flexibility but it should be obvious that it also is a source to numerous programming errors and mistakes. This kind of error is also very hard to debug and detect.

Arrays

An array in PHP is an ordered map. An **array** can be created by the **array()** construct. It takes a certain number of comma-separated *key => value* pairs. *key* may be an **integer** or **string.** The following example illustrates this concept:

```
<?php
        $arr = array("foo" => "bar", 12 => true);
        echo $arr["foo"];               // bar
        echo $arr[12];          // 1
?>
```

If no key is specified in the assignment, then the maximum of the existing integer indices is taken, and the new key will be that maximum value + 1. If the current maximum is negative then the next key created will be zero. If no integer indices exist yet, the key will be *0* (zero). Note that the maximum integer key used for this *need not currently exist in the array*. It simply must have existed in the array at some time since the last time the array was re-indexed. If the specified key already has a value assigned to it, that value will be overwritten. These things are demonstrated in the following example:

```
<?php
        $arr = array(5 => 1, 12 => 2);
```

```
        $arr[] = 56;     // This is the same as $arr[13] = 56;
                                // at this point of the script
        $arr["x"] = 42;          // This adds a new element to
                        // the array with key "x“
        unset($arr[13]);         // This removes the element from the array
        $arr[] = 56;     // This is the same as $arr[14] = 56;
                                // at this point of the script
        unset($arr);     // This deletes the whole array
?>
```

class

A class is defined as shown below:

```
<?php
        class Cart {
                var $items; // Items in our shopping cart
                                // Add $num articles of $artnr to the cart

                 function add_item($artnr, $num) {
                        $this->items[$artnr] += $num;
                }
                function remove_item($artnr, $num) {
                        if ($this->items[$artnr] > $num) {
                                $this->items[$artnr] -= $num;
                                return true;
                        }
                         elseif ($this->items[$artnr] == $num) {
                                unset($this->items[$artnr]);
                                return true;
                        }
                        else { return false; }
                }
        }
?>
```

Aliasing

Aliasing is used to assign by reference. To assign by reference, simply prepend an ampersand (&) to the beginning of the variable which is being assigned (the source variable) as shown below:

```
<?php
        $foo = ‘Fakhar';
        $bar = &$foo;                    // Reference $foo via $bar.
        $bar = "My name is $bar";     // Alter $bar...
        echo $bar;
```

```
    echo $foo;                          // $foo is altered too.
?>
```

Variable variable

A variable variable is like pointers that is, it maintains the address of a variable in it. This is shown in the example below:

```
<?php
        $a = ‘hello’;
        $$a = 'world';
        echo "$a ${$a}“;              // ‘hello world”
?>
```

Constants

A constant is an identifier (name) for a simple value that cannot change during the execution of the script. A constant is case-sensitive by default. By convention, constant identifiers are always uppercase. The name of a constant follows the same rules as any label in PHP. A valid constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thusly: *[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]** .

Logical operators

PHP supports the following logical operators.

and, or, xor, !, &&, ||

The reason for the two different variations of "and" and "or" operators is that they operate at different precedence. This is demonstrated with the help of the following example:

```
$a and $b or $c     //      ($a and $b) or $c
$a && $b || $c
$a and $b || $c     //      $a and ($b || $c)
```

String operators

There are two **string** operators. The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side.

```
<?php
        $a = "Hello ";
        $b = $a . "World!"; // now $b contains "Hello World!"
```

```
        $a = "Hello ";
        $a .= "World!";          // now $a contains "Hello World!"
?>
```

Array Operators

PHP provides a number of operators to manipulate arrays. These are:

- $a + $b Union – Union of $a and $b.
- $a == $b Equality – **TRUE** if $a and $b have the same key/value pairs.
- $a === $b Identity - **TRUE** if $a and $b have the same key/value pairs in the same order and of the same types.
- $a != $b Inequality - **TRUE** if $a is not equal to $b.
- $a <> $b Inequality - **TRUE** if $a is not equal to $b.
- $a !== $b Non-identity - **TRUE** if $a is not identical to $b.

Note the difference between == and ===. The former checks if the same key-value pairs are present in the two arrays and the order does not matter whereas the latter requires that in order to be identical the pairs have to be in the same order.

The + operator appends the right handed array to the left handed, whereas duplicated keys are NOT overwritten. This concept is elaborated as below:

```
$a = array("a" => "apple", "b" => "banana");
$b = array("a" => "pear", "b" => “date", "c" => “mango");
$c = $a + $b; // Union of $a and $b
```

This will result in an array c with the following values:

```
["a"]=> "apple" ["b"]=> "banana" ["c"]=> “mango"
```

Control Statements

PHP supports the following control statements:

- if
- while
- do while
- for
- switch
- foreach
- break
- continue

Most control structures work in a manner similar to C. The differences are highlighted in the following paragraphs.

If statements

The if statement in PHP has the following shape and form:

```
<?php
        if ($a > $b) { echo "a is bigger than b"; }
        elseif ($a == $b) { echo "a is equal to b"; }
        else { echo "a is smaller than b"; }
?>
```

There may be several *elseif*s within the same *if* statement. The first *elseif* expression (if any) that evaluates to **TRUE** would be executed. In PHP, you can also write 'else if' (in two words) and the behavior would be identical to the one of 'elseif' (in a single word).

foreach

foreach statement works only on arrays, and will issue an error when you try to use it on a variable with a different data type or an uninitialized variable. It gives an easy way to iterate over arrays. There are two syntaxes; the second is a minor but useful extension of the first:

- foreach (array_expression as $value) statement
- foreach (array_expression as $key => $value) statement

The first form loops over the array given by *array_expression*. On each loop, the value of the current element is assigned to *$value* and the internal array pointer is advanced by one. The second form does the same thing, except that the current element's key will also be assigned to the variable *$key* on each loop. As of PHP 5, it is possible to iterate objects too.

Note: Unless the array is referenced, *foreach* operates on a copy of the specified array and not the array itself. Therefore changes to the array element returned are not reflected in the original array. As of PHP 5, array's elements can be modified by preceding *$value* with &. This will assign reference instead of copying the value.

```
<?php
        $arr = array(1, 2, 3, 4);
        foreach ($arr as &$value)
                { $value = $value * 2; }
                        // $arr is now array(2, 4, 6, 8)
?>
```

This is possible only if iterated array can be referenced (i.e. is variable).

each

each return the current key and value pair from an array and advance the array cursor. It is used in the following manner:

array **each** (array &array)

Returns the current key and value pair from the array *array* and advances 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. If the internal pointer for the array points past the end of the array contents, **each()** returns **FALSE**.

break

break ends execution of the current *for*, *foreach*, *while*, *do-while* or *switch* structure. In PHP, *break* accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of. The following example illustrates this concept:

```
<?php
        $i = 0;
        while (++$i) {
                switch ($i) {
                        case 5:
                                echo "At 5<br />\n";
                                break 1; // Exit only the switch.
                        case 10:
                                echo "At 10; quitting<br />\n";
                                break 2; // Exit the switch and the while.
                        default:
                                break;    // Exit only the switch.
                }
        }
?>
```

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. **Note:** Note that in PHP the switch statement is considered a looping structure for the purposes of *continue*. Like break, *continue* accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

Omitting the semicolon after *continue* can lead to confusion as shown below:

```
<?php
        for ($i = 0; $i < 5; ++$i) {
                if ($i == 2) continue print "$i\n";
        }
?>
```

One can expect the result to be : 0 1 3 4 but this script will output : 2 because the return value of the **print()** call is *int(1)*, and it will look like the optional numeric argument mentioned above.

Alternative syntax for control statements

PHP offers an alternative syntax for some of its control structures; namely, *if*, *while*, *for*, *foreach*, and *switch*. In each case, the basic form of the alternate syntax is to change the opening brace to a colon (:) and the closing brace to *endif;*, *endwhile;*, *endfor;*, *endforeach;*, or *endswitch;*, respectively. An example is shown below:

```
<?php
        if ($a == 5):
                echo "a equals 5";
                echo "...";
        elseif ($a == 6):
                echo "a equals 6";
                echo "!!!";
        else: echo "a is neither 5 nor 6";
        endif;
?>
```

User defined functions

Just like all programming languages, one can define functions in PHP as shown in the following example:

```
<?php
        function foo($arg_1, $arg_2, /* ..., */ $arg_n)
        {
                echo "Example function.\n";
                return $retval;
        }
?>
```

PHP also allows functions to be defined inside functions. It may be noted that C and its descendents do not support this feature but descendants of Algol generally support it. The following example shows this concept:

```
<?php
	function foo() {
		function bar() {
		echo "I don't exist until foo() is called.\n";
		}
	}

	/* We can't call bar() yet since it doesn't exist. */

	foo();

	/* Now we can call bar(), foo()'s processesing has made it
	  accessible. */

	bar();
?>
```

PHP also supports recursive functions as shown below:

```
	<?php
		function recursion($a) {
			if ($a < 20) {
				echo "$a\n"; recursion($a + 1);
			}
		}
	?>
```

Function arguments

PHP supports passing arguments by value (the default), passing by reference, and default argument values. If you want an argument to a function to always be passed by reference, you can prepend an ampersand (&) to the argument name in the function definition:

```
	<?php
		function add_some_extra(&$string) {
			$string .= 'and something extra.';
		}
		$str = 'This is a string, ';
		add_some_extra($str);
		echo $str; // outputs 'This is a string, and
				// something extra.‘
	?>
```

You can't return multiple values from a function, but similar results can be obtained by returning a list as shown below:

```
	<?php
```

```
        function small_numbers() {
                return array (0, 1, 2);
        }
        list ($zero, $one, $two) = small_numbers();
?>
```

To return a reference from a function, you have to use the reference operator & in both the function declaration and when assigning the returned value to a variable:

```
<?php
        function &returns_reference() {
                return $someref;
        }
        $newref =& returns_reference();
?>
```

Classes and Objects

Classes and objects are similar to Java. A variable of the desired type is created with the *new* operator. It supports Single inheritance only and uses the keyword extends to inherit from a super class. The inherited methods and members can be overridden, unless the parent class has defined a method as final.

Databases

One of the strongest features of PHP is its support for providing web pages access to database. The following example shows database manipulation through ODBC in PHP:

```
<html>
<body>
<?php
        $conn=odbc_connect('northwind',",");
        if (!$conn)  {
                exit("Connection Failed: " . $conn);
        }
        $sql="SELECT * FROM customers";
        $rs=odbc_exec($conn,$sql);
        if (!$rs)  {exit("Error in SQL");}
        echo "<table><tr>";
        echo "<th>Companyname</th>";
        echo "<th>Contactname</th></tr>“;
        while (odbc_fetch_row($rs)){
                $compname=odbc_result($rs,"CompanyName");
                $conname=odbc_result($rs,"ContactName");
                echo "<tr><td>$compname</td>";
                echo "<td>$conname</td></tr>";
        }
        odbc_close($conn);
        echo "</table>";
?>
</body>
</html>
```