1ο ΣΧΟΛΕΙΟ ΚΩΔΙΚΑ

«Βασικά Θέματα Προγραμματισμού στην Ανάπτυξη Δυναμικών Διαδικτυακών Εφαρμογών»

(PHP Variables-Operators-Functions-Conditional-Loops-Objects)

Γιάννης Σαμωνάκης

Table of Contents

What is PHP

PHP is a server-side scripting [1] language designed for web development but also used as a general-purpose programming language [2]. As of January 2013, PHP was installed on more than 240 million websites (39% of those sampled) and 2.1 million web servers. While PHP originally stood for Personal Home Page, it now stands for PHP: Hypertext Preprocessor, which is a recursive backronym.

PHP code can be simply mixed with HTML code, or it can be used in combination with various templating engines and web frameworks. PHP code is usually processed by a PHP interpreter. An interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without previously compiling them into a machine language program. After the PHP code is interpreted and executed, the web server sends resulting output to its client, usually in form of a part of the generated web page – for example, PHP code can generate a web page's HTML code, an image, or some other data.

What can PHP do

There are three main areas where PHP scripts are used [5].

Server-side scripting. This is the most traditional and main target field for PHP. You need three things to make this work. The PHP parser (CGI or server module), a web server and a web browser. You need to run the web server, with a connected PHP installation. You can access the PHP program output with a web browser, viewing the PHP page through the server. All these can run on your home machine if you are just experimenting with PHP programming. See the installation instructions section for more information.

Command line scripting. You can make a PHP script to run it without any server or browser. You only need the PHP parser to use it this way. This type of usage is ideal for scripts regularly executed using cron (on *nix or Linux) or Task Scheduler (on Windows). These scripts can also be used for simple text processing tasks. See the section about Command line usage of PHP for more information.

Writing desktop applications. PHP is probably not the very best language to create a desktop application with a graphical user interface, but if you know PHP very well, and would like to use some advanced PHP features in your client-side applications you can also use PHP-GTK to write such programs. You also have the ability to write cross-platform applications this way. PHP-GTK is an extension to PHP, not available in the main distribution. If you are interested in PHP-GTK, visit » its own website.

Your first PHP-enabled page

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

The above script output "Hello World!"). Every piece of PHP code begins with “<?php” and is finished by adding “?>” at the end. Everything outside of a pair of opening and closing tags is ignored by the PHP parser which allows PHP files to have mixed content (HTML+PHP). This allows PHP to be embedded in HTML documents

Declaring PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

```
<!DOCTYPE html>
<html>
<body>

<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;

echo $txt;
echo "<br>";
echo $x;
echo "<br>";
echo $y;
?>

</body>
</html>
```

After the execution of the statements above, the variable $txt will hold the value Hello world!, the variable $x will hold the value 5, and the variable $y will hold the value 10.5.

When you assign a text value to a variable, put quotes around the value. Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

Output in PHP

In PHP there are two basic ways to get output: echo and print.

Display text

```
<!DOCTYPE html>
<html>
<body>

<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made
", "with multiple parameters.";
?>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>

</body>
</html>
```

Display variables

```
<!DOCTYPE html>
<html>
<body>

<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

print "<h2>" . $txt1 . "</h2>";
print "Study PHP at " . $txt2 . "<br>";
print $x + $y;
?>

</body>
</html>
```

Data Types

Variables can store data of different types, and different data types can do different things. PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

PHP String	
A string is a sequence of characters, like "Hello world!". A string can be any text inside quotes. You can use single or double quotes	<!DOCTYPE html> <html> <body> <?php $x = "Hello world!"; $y = 'Hello world!'; echo $x; echo " "; echo $y; ?> </body> </html>

PHP Integer	
An integer is a whole number (without decimals). It is a number between -2,147,483,648 and +2,147,483,647. Rules for integers: • An integer must have at least one digit (0-9) • An integer cannot contain comma or blanks • An integer must not have a decimal point • An integer can be either positive or negative • Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed	<!DOCTYPE html> <html> <body> <?php $x = 5985; var_dump($x); ?> </body> </html>

with 0x) or octal (8-based - prefixed with 0) In the following example $x is an integer. The PHP var_dump() function returns the data type and value	

PHP Float	
A float (floating point number) is a number with a decimal point or a number in exponential form. In the following example $x is a float. The PHP var_dump() function returns the data type and value	<!DOCTYPE html> <html> <body> <?php $x = 10.365; var_dump($x); ?> </body> </html>

PHP NULL Value	
Null is a special data type which can have only one value: NULL. A variable of data type NULL is a variable that has no value assigned to it. Tip: If a variable is created without a value, it is automatically assigned a value of NULL. Variables can also be emptied by setting the value to NULL	<!DOCTYPE html> <html> <body> <?php $x = "Hello world!"; $x = null; var_dump($x); ?> </body> </html>

PHP Arrays	
An array stores multiple values in one single variable. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this $cars1 = "Volvo"; $cars2 = "BMW"; $cars3 = "Toyota"; However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300? The solution is to create an array!	<!DOCTYPE html> <html> <body> <?php $cars = array("Volvo", "BMW", "Toyota"); echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";

An array can hold many values under a single name, and you can access the values by referring to an index number

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array

```
$age = array("Peter"=>"35", "Ben"=>"37",
"Joe"=>"43");
```

Or

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

```
?>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37",
"Joe"=>"43");
echo "Peter is " . $age['Peter'] . "
years old.";
?>

</body>
</html>
```

PHP Objects	

An object is a data type which stores data and information on how to process that data.
 	
In PHP, an object must be explicitly declared.
 	
First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods (see section *Functions as Class Methods*)
	<!DOCTYPE html>
	<html>
	<body>
	
	<?php
	
	// class definition
	
	class Bear {
	
	    // define properties
	    public $name;
	    public $weight;
	    public $age;
	    public $sex;
	
	}
	
	// my first bear
	$daddy = new Bear;
	$daddy->name = "Daddy Bear"; // give him a name
	$daddy->age = 8; // how old is he
	$daddy->sex = "male"; // what sex is he
	$daddy->colour ="black"; //what colour is his coat
	$daddy->weight = 300; // how much does he weigh
	
	// give daddy a wife
	$mommy = new Bear;
	$mommy->name = "Mommy Bear";
	$mommy->age = 7;
	$mommy->sex = "female";
	$mommy->colour = "black";
	$mommy->weight = 310;
	
	// and a baby to complete the family
	$baby = new Bear;
	$baby->name = "Baby Bear";
	$baby->age = 1;
	$baby->sex = "male";
	$baby->colour = "black";
	$baby->weight = 180;
	?>
	
	</body>
	</html>

PHP Resource
The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. A common example of using the resource data type is a database call

Constants

Constants are like variables except that once they are defined they cannot be changed or undefined. A constant is an identifier (name) for a simple value. The value cannot be changed during the script. A valid constant name starts with a letter or underscore (no $ sign before the constant name).

Syntax: *define(name, value, case-insensitive)*

Parameters:

- name: Specifies the name of the constant
- value: Specifies the value of the constant
- case-insensitive: Specifies whether the constant name should be case-insensitive. Default is false

```
<!DOCTYPE html>
<html>
<body>

<?php
// case-insensitive constant name
define("GREETING", "Welcome to W3Schools.com!", true);
echo greeting;
?>

</body>
</html>
```

User Defined Functions

Besides the built-in PHP functions, we can create our own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.
- A function name can start with a letter or underscore (not a number).
- Give the function a name that reflects what the function does!
- Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

```
<!DOCTYPE html>
<html>
<body>

<?php
function writeMsg() {
     echo "Hello world!";
}

writeMsg();
?>

</body>
</html>
```

Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma. Usually the functions use arguments to calculate a new value and return it to the main program.

The following example has a function with two arguments. When the sum() function is called, we pass along two integers which they are used inside the function to calculate their sum. Finally, the function returns the result.

```
<!DOCTYPE html>
<html>
<body>

<?php
function sum($x, $y) {
     $z = $x + $y;
     return $z;
}

echo "5 + 10 = " . sum(5,10) . "<br>";
echo "7 + 13 = " . sum(7,13) . "<br>";
echo "2 + 4 = " . sum(2,4);
?>

</body>
</html>
```

Variables Scope

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced/used. PHP has three different variable scopes:

- local
- global
- static

A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5; // global scope

function myTest() {
     // using x inside this function will generate an error
     echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>

</body>
</html>
```

A variable declared within a function has a **LOCAL SCOPE** and can only be accessed within that function:

```
<!DOCTYPE html>
<html>
<body>

<?php
function myTest() {
     $x = 5; // local scope
     echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>

</body>
</html>
```

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

The **global keyword** is used to access a global variable from within a function. To do this, use the global keyword before the variables (inside the function):

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5;
$y = 10;

function myTest() {
     global $x, $y;
     $y = $x + $y;
}

myTest(); // run function
echo $y; // output the new value for variable $y
?>

</body>
</html>
```

PHP also stores all global variables in an array called $GLOBALS[index]. The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5;
$y = 10;

function myTest() {
     $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y;
?>

</body>
</html>
```

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job. To do this, use the **static keyword** when you first declare the variable:

```
<!DOCTYPE html>
<html>
<body>

<?php
function myTest() {
     static $x = 0;
     echo $x;
     $x++;
}

myTest();
echo "<br>";
myTest();
echo "<br>";
myTest();
echo "<br>";
myTest();
echo "<br>";
myTest();
?>

</body>
</html>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called. The variable is still local to the function.

Functions as Class Methods

As mentioned in table "*PHP Objects*", a class is a structure that can contain properties (variables) and methods (functions). We can create and use class methods in the same way as the class properties. Below, we extend the above example in order to add some methods:

```
<!DOCTYPE html>
<html>
<body>
<?php
// class definition
class Bear {
    // define properties
    public $name;
    public $weight;
    public $age;
    public $sex;

    // define methods
    public function eat() {
        echo $this->name." is eating...";
    }

    public function run() {
        echo $this->name." is running...";
    }
```

```
    public function sleep() {
        echo $this->name." is sleeping...";
    }
}
// my first bear
$daddy = new Bear;
$daddy->name = "Daddy Bear"; // give him a name
$daddy->age = 8; // how old is he
$daddy->sex = "male"; // what sex is he
$daddy->colour ="black"; //what colour is his coat
$daddy->weight = 300; // how much does he weigh

// give daddy a wife
$mommy = new Bear;
$mommy->name = "Mommy Bear";
$mommy->age = 7;
$mommy->sex = "female";
$mommy->colour = "black";
$mommy->weight = 310;

// and a baby to complete the family
$baby = new Bear;
$baby->name = "Baby Bear";
$baby->age = 1;
$baby->sex = "male";
$baby->colour = "black";
$baby->weight = 180;

// a nice evening in the Bear family
// mommy eat
$mommy->eat();
// and so does baby
$baby->eat();
// mommy sleeps
$mommy->sleep();
// and so does daddy
$daddy->sleep();
// baby eats some more
$baby->eat();
?>

</body>
</html>
```

Operators

Operators are used to perform operations on variables and values. PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

Arithmetic Operators

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced in PHP 5.6)

Assignment Operators

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

Comparison Operators

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

Increment/Decrement Operators

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

Logical Operators

Operator	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

String Operators

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

Array Operators

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- **if statement** - executes some code only if a specified condition is true

Syntax:

```
if (condition) {
code to be executed if condition is true;
}
```

- **if...else statement** - executes some code if a condition is true and another code if the condition is false

Syntax:

```
if (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}
```

- **if...elseif....else statement** - specifies a new condition to test, if the first condition is false

Syntax:

```
if (condition) {
    code to be executed if condition is true;
} elseif (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
$t = date("H");

if ($t <  "10") {
     echo "Have a good morning!";
} elseif ($t <  "20") {
     echo "Have a good day!";
} else {
     echo "Have a good night!";
}
?>

</body>
</html>
```

- **switch statement** - selects one of many blocks of code to be executed

Syntax:

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
    ...
    default:
        code to be executed if n is different from all labels;
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
$favcolor = "red";

switch ($favcolor) {
     case "red":
         echo "Your favorite color is red!";
```

```
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, blue, or green!";
}
?>

</body>
</html>
```

Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true

Syntax:

```
while (condition is true) {
    code to be executed;
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 1;

while($x <= 5) {
   echo "The number is: $x <br>";
   $x++;
}
?>

</body>
</html>
```

- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true

Syntax:

```
do {
    code to be executed;
} while (condition is true);
```

- **for** - loops through a block of code a specified number of times

The for loop is used when you know in advance how many times the script should run.

Parameters:

- init counter: Initialize the loop counter value
- test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- increment counter: Increases the loop counter value

Syntax:

```
for (init counter; test counter; increment counter) {
    code to be executed;
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
for ($x = 0; $x <= 10; $x++) {
   echo "The number is: $x <br>";
}
?>

</body>
</html>
```

- **foreach** - loops through a block of code for each element in an array

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax:

```
foreach ($array as $value) {
    code to be executed;
}
```

Example:

```
<!DOCTYPE html>
<html>
<body>

<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
```

```
   echo "$value <br>";
}
?>

</body>
</html>
```

References

[1] http://en.wikipedia.org/wiki/Server-side_scripting

[2] http://en.wikipedia.org/wiki/General-purpose_programming_language

[3] http://php.net/downloads.php

[4] http://php.net/manual/en/install.php

[5] http://php.net/manual/en/intro-whatcando.php

[6] http://www.w3schools.com/php/default.asp

[7] http://devzone.zend.com/6/php-101-php-for-the-absolute-beginner/