02 – PHP Basics

CET218 – Advanced Web Programming

Dr. Ahmed Said

How things are going so far

- **Engagement**
  - Lecture (lots of good questions and participation)
  - Section (practical, people working together in breakouts)
- **Github**
  - GitHub Repository: https://github.com/dr-ahmed-said/advanced-web
  - Fork the course repository and submit your course work in your fork/branch
  - Your work will be monitored and graded

The Difference Between echo & print

- The two commands are quite similar, but
  - print is a function-like construct that takes a single parameter and has a return value (which is always 1),
  - whereas echo is purely a PHP language construct.
- Since both commands are constructs, neither requires parentheses.
- By and large, the echo command usually will be a tad faster than print, because it doesn't set a return value.
- On the other hand, because it isn't implemented like a function, echo cannot be used as part of a more complex expression, whereas print can.

```
$b ? print "TRUE" : print "FALSE";
```

Operators

- PHP offers a lot of powerful operators of different types
  - **Unary operators**: such as incrementing (`$a++`) or negation (`!$a`), take a single operand.
  - Binary operators: takes two operands
  - The one ternary operator: which takes the form `expr ? x : y`, requires 3 operands.

Operator	Description	Example
Arithmetic	Basic mathematics	`$a + $b`
Array	Array union	`$a + $b`
Assignment	Assign values	`$a = $b + 23`
Bitwise	Manipulate bits within bytes	`12 ^ 9`
Comparison	Compare two values	`$a < $b`
Execution	Execute contents of backticks	`` `ls -al` ``
Increment/decrement	Add or subtract 1	`$a++`
Logical	Boolean	`$a and $b`
String	Concatenation	`$a . $b`

Operator Precedence

- If all operators had the same precedence, they would be processed in the order in which they are encountered.

1 + 2 + 3 - 4 + 5
2 - 4 + 5 + 3 + 1
5 + 2 - 4 + 1 + 3

1 + 2 * 3 - 4 * 5
2 - 4 * 5 * 3 + 1 ?
5 + 2 - 4 + 1 * 3

1 + (2 * 3) - (4 * 5)
2 - (4 * 5 * 3) + 1
5 + 2 - 4 + (1 * 3)

The precedence of PHP operators (high to low)

Operator(s)	Type
()	Parentheses
++ --	Increment/decrement
!	Logical
* / %	Arithmetic
+ - .	Arithmetic and string
<< >>	Bitwise
< <= > >= <>	Comparison
== != === !==	Comparison
&	Bitwise (and references)
^	Bitwise
|	Bitwise
&&	Logical
||	Logical
? :	Ternary
= += -= *= /= .= %= &= != ^= <<= >>=	Assignment
and	Logical
xor	Logical
or	Logical

Associativity

- Processing expressions from left to right, except were operator precedence is in effect.
- But some operators require processing from right to left, and this direction of processing is called the operator's associativity.
- For some operators, there is no associativity.

```php
<?php
$level = $score = $time = 0;
?>
```

This multiple assignment is possible only if the rightmost part of the expression is evaluated first and then processing continues in a right-to-left direction.

Operator	Description	Associativity
< <= >= == != === !== <>	Comparison	None
!	Logical NOT	Right
~	Bitwise NOT	Right
++ --	Increment and decrement	Right
(int)	Cast to an integer	Right
(double) (float) (real)	Cast to a floating-point number	Right
(string)	Cast to a string	Right
(array)	Cast to an array	Right
(object)	Cast to an object	Right
@	Inhibit error reporting	Right
= += -= *= /=	Assignment	Right
.= %= &= |= ^= <<= >>=	Assignment	Right
+	Addition and unary plus	Left
-	Subtraction and negation	Left
*	Multiplication	Left
/	Division	Left
%	Modulus	Left
.	String concatenation	Left
<< >> & ^ |	Bitwise	Left
?:	Ternary	Left
|| && and or xor	Logical	Left
,	Separator	Left

Equality

- The equality operator is == (two equals signs).
- It is important not to confuse it with the = (single equals sign) assignment operator.
- As you see, by returning either TRUE or FALSE, the equality operator enables you to test for conditions

```php
<?php
$month = "March";
if ($month == "March") echo "It's springtime";
?>
```

Equality

```php
<?php
$a = "1000";
$b = "+1000";
if ($a == $b) echo "1";
?>
```

- But that's not the whole story, because PHP is a loosely typed language.
  - If the two operands of an equality expression are of different types, PHP will convert them to whatever type makes the best sense to it.
- A rarely used identity operator, which consists of three equals signs in a row, can be used to compare items without doing conversion.

```php
<?php
$a = "1000";
$b = "+1000";
if ($a == $b) echo "1";
if ($a === $b) echo "2";
?>
```

Variable incrementing & decrementing

- Adding or subtracting 1 is such a common operation that PHP provides special operators for it.

```php
++$x;
--$y;
```

```php
$x = 9
if (++$x == 10) echo $x;
```

So, what will the echo statement display?

```php
$y = 0
if ($y-- == 0) echo $y;
```

So, what will the echo statement display: 0 or -1?

String **types**

- PHP supports **two types of strings** that are denoted by the type of quotation mark that you use.
- If you wish to assign a literal string, preserving the exact contents, you should use single quotation marks (apostrophes), like this:

```php
$variable = 0;
$info = 'Preface variables with a $ like this: $variable';
echo $info;

$count = 5;
echo "This week $count people have viewed your profile";
```

- Every character within the single-quoted string is assigned to $info.
- If you had used double quotes; PHP would have attempted to evaluate $variable as a variable.

Interpreted Strings

```php
echo "Today is your $ageth birthday.\n"; # $ageth not found
echo "Today is your {$age}th birthday.\n";
```

PHP

- if necessary to avoid ambiguity, can enclose variable in {}

Interpreted Strings (cont.)

```php
$name = “Xenia";
$name = NULL;
if (isset($name)) {
    echo "This line is not going to be reached.\n";
}
```

PHP

- a variable is NULL if
  - it has not been set to any value (undefined variables)
  - it has been assigned the constant NULL
  - it has been deleted using the unset function
- can test if a variable is NULL using the isset function
- NULL prints as an empty string (no output)

String concatenation

- Concatenation is a term for putting something after another thing.
- So, string concatenation uses the period (.) to append one string of characters to another.

```php
echo "You have " . $msgs . " messages.";
```

PHP

```
You have 5 messages.
```

- you can append one string to another using .=, like this:

```php
$bulletin .= $newsflash;
```

PHP

String Type

```php
$favorite_food = "Ethiopian";
print $favorite_food[2];
$favorite_food = $favorite_food . " cuisine";
print $favorite_food;
```

PHP

- zero-based indexing using bracket notation
- there is no char type; each letter is itself a String
- string concatenation operator is . (period), not +
  - 5 + "2 turtle doves" === 7
  - 5 . "2 turtle doves" === "52 turtle doves"
- can be specified with "" or ''

String Functions

```php
# index 0123456789012345
$name = "Stefanie Hatcher";
$length = strlen($name);
$cmp = strcmp($name, "Brian Le");
$index = strpos($name, "e");
$first = substr($name, 9, 5);
$name = strtoupper($name);
```

PHP

String Functions (cont.)

Name	Function
strlen	length
strpos	indexOf
substr	substring
strtolower, strtoupper	toLowerCase, toUpperCase
trim	trim
explode, implode	split, join
strcmp	compareTo

bool (Boolean) type

```php
$feels_like_summer = FALSE;
$php_is_great = TRUE;
$student_count = 7;
$nonzero = (bool) $student_count; # TRUE
```

PHP

- The following values are considered to be FALSE (all others are TRUE):
  - 0 and 0.0 (but NOT 0.00 or 0.000)
  - "", "0", and NULL (includes unset variables)
  - arrays with 0 elements
- FALSE prints as an empty string (no output); TRUE prints as a 1

Math operations

```php
$a = 3;
$b = 4;
$c = sqrt(pow($a, 2) + pow($b, 2));
```

PHP

math functions

abs	ceil	cos	floor	log	log10	max
min	pow	rand	round	sin	sqrt	tan

math constants

M_PI	M_E	M_LN2

Int and Float Types

```php
$a = 7 / 2; # float: 3.5
$b = (int) $a; # int: 3
$c = round($a); # float: 4.0
$d = "123"; # string: "123"
$e = (int) $d; # int: 123
```

PHP

- int for integers and float for reals
- Division between two int values can produce a float

Implicit and Explicit Casting

- PHP is a loosely typed language that allows you to declare a variable and its type simply by using it.
- It also automatically converts values from one type to another whenever required. This is called implicit casting.
- However, at times PHP's implicit casting may not be what you want

```php
<?php
$a = 56;
$b = 12;
$c = $a / $b;
echo $c;
?>
```

- The inputs to the division are integers. By default, PHP converts the output to floating point so it can give the most precise value—4.66 recurring.

Implicit and Explicit Casting

- But what if we had wanted $c to be an integer instead?
- There are various ways we could achieve this, one of which is to force the result of $a / $b to be cast to an integer value using the integer cast type (int) This is called explicit casting.

```php
$c = (int) ($a / $b);
```

PHP

- We place the expression within parentheses. Otherwise, only the variable $a would have been cast to an integer—a pointless exercise, as the division by $b would still have returned a floating-point number.

Cast type	Description
(int) (integer)	Cast to an integer by dropping the decimal portion.
(bool) (boolean)	Cast to a Boolean.
(float) (double) (real)	Cast to a floating-point number.
(string)	Cast to a string.
(array)	Cast to an array.
(object)	Cast to an object.

Escaping characters

- Sometimes a string needs to contain characters with special meanings that might be interpreted incorrectly.

```
$text = 'My spelling's atroshus'; // Erroneous syntax
```

```
$text = 'My spelling\'s still atroshus';
```

```
$text = "She wrote upon it, \"Return to sender\".";
```

- you can use escape characters to insert various special characters into strings, such as tabs, newlines, and carriage returns. These are represented, as you might guess, by \t, \n, and \r.

```
$heading = "Date\tName\tPayment";
```

Multiline Commands

- There are times when you need to output quite a lot of text from PHP, and using several echo (or print) statements would be time-consuming and messy.
- To overcome this, just put multiple lines between quotes

```php
<?php
$author = "Steve Ballmer";
echo "Developers, developers, developers, developers, developers,
developers, developers, developers, developers!
- $author.";
?>
```

```php
<?php
$author = "Bill Gates";
$text = "Measuring programming progress by lines of code is like
Measuring aircraft building progress by weight.
- $author.";
?>
```

Predefined Constants

- There are a few—known as the magic constants—that you will find useful.
- The names of the magic constants always have two underscores at the beginning and two at the end so that you won't accidentally try to name one of your own constants with a name that is already taken.

Magic constant	Description
`__LINE__`	The current line number of the file.
`__FILE__`	The full path and filename of the file. If used inside an `include`, the name of the included file is returned. Some operating systems allow aliases for directories, called *symbolic links*; in `__FILE__` these are always changed to the actual directories.
`__DIR__`	The directory of the file. If used inside an `include`, the directory of the included file is returned. This is equivalent to *`dirname`*`(__FILE__)`. This directory name does not have a trailing slash unless it is the root directory.
`__FUNCTION__`	The function name. Returns the function name as it was declared (case-sensitive). In PHP 4, its value is always lowercase.
`__CLASS__`	The class name. Returns the class name as it was declared (case-sensitive). In PHP 4, its value is always lowercase.
`__METHOD__`	The class method name. The method name is returned as it was declared (case-sensitive).
`__NAMESPACE__`	The name of the current namespace. This constant is defined at compile time (case-sensitive).

Functions

- Functions separate sections of code that perform a particular task.
  - For example, maybe you often need to look up a date and return it in a certain format. That would be a good example to turn into a function.
  - The code doing it might be only three lines long, but if you have to paste it into your program a dozen times
- **Don't Repeat yourself**

```php
<?php
function longdate($timestamp)
{
      return date("l F jS Y", $timestamp);
}
echo longdate(time());
echo longdate(time() - 17 * 24 * 60 * 60);
?>
```

Variable Scope

- If you have a very long program, it's quite possible that you could start to run out of good variable names with PHP you can decide the scope of a variable
- For example, tell it that you want the variable $temp to be used only inside a particular function and to forget it was ever used when the function returns.
- In fact, this is the default scope for PHP variables.
- Alternatively, you could inform PHP that a variable is global in scope and thus can be accessed by every other part of your program.

Local variables

- Local variables are variables that are created within, and can be accessed only by, a function.
- They are generally temporary variables that are used to store partially processed results prior to the function's return.
- One set of local variables is the list of arguments to a function.

```php
<?php
function longdate($timestamp)
{
	$temp = date("l F jS Y", $timestamp);
	return "The date is $temp";
}
?>
```

Local variables

- Now, to see the effects of variable scope, let's look at some similar code in

```php
<?php
$temp = "The date is ";
echo longdate(time());
function longdate($timestamp)
{
    return $temp . date("l F jS Y", $timestamp);
}
?>
```

- **Output?**
- Outputs only the date, not the preceding text.
- In fact, depending on how PHP is configured, it may first display the error message Notice: Undefined variable: temp
- **Fix ?**

Local variables

- Now, to see the effects of variable scope, let's look at some similar code in

```php
<?php
$temp = "The date is ";
echo $temp . longdate(time());
function longdate($timestamp)
{
	return date("l F jS Y", $timestamp);
}
?>
```

```php
<?php
$temp = "The date is ";
echo longdate($temp, time());
function longdate($text, $timestamp)
{
	return $text . date("l F jS Y", $timestamp);
}
?>
```

Global variables

- There are cases when you need a variable to have global scope, because you want all your code to be able to access it.
- Also, some data may be large and complex, and you don't want to keep passing it as arguments to functions.
- To access variables from global scope, add the keyword global.

```
global $is_logged_in;
```

- You should use variables given global access with caution, though. I recommend that you create them only when you absolutely cannot find another way of achieving the result you desire.

Variable Scope

- The value of a local variable is wiped out when the function ends.
- If a function runs many times, it starts with a fresh copy of the variable, and the previous setting has no effect.
- What if you have a local variable inside a function that you don't want any other parts of your code to have access to, but you would also like to keep its value for the next time the function is called? **Why**?
  - Perhaps because you want a counter to track how many times a function is called.
- The solution is to declare a static variable

```php
<?php
function test()
{
    static $count = 0;
    echo $count;
    $count++;
}
?>
```

Static variables

- If you plan to use static variables, you should note that you cannot assign the result of an expression in their definitions.
- They can be initialized only with predetermined values

```php
<?php
static $int = 0; // Allowed
static $int = 1+2; // Correct (as of PHP 5.6)
static $int = sqrt(144); // Disallowed
?>
```

Superglobal variables

- Starting with *PHP 4.1.0*, several predefined variables are available. These are known as superglobal variables,
  - which means that they are provided by the PHP environment but are global within the program, accessible absolutely everywhere.
- These superglobals contain lots of useful information about the currently running program and its environment.
- They are structured as associative arrays

Superglobal variables

Superglobal name	Contents
$GLOBALS	All variables that are currently defined in the global scope of the script. The variable names are the keys of the array.
$_SERVER	Information such as headers, paths, and locations of scripts. The entries in this array are created by the web server, and there is no guarantee that every web server will provide any or all of these.
$_GET	Variables passed to the current script via the HTTP GET method.
$_POST	Variables passed to the current script via the HTTP POST method.
$_FILES	Items uploaded to the current script via the HTTP POST method.
$_COOKIE	Variables passed to the current script via HTTP cookies.
$_SESSION	Session variables available to the current script.
$_REQUEST	Contents of information passed from the browser; by default, $_GET, $_POST, and $_COOKIE.
$_ENV	Variables passed to the current script via the environment method.

- Among the many nuggets of information supplied by superglobal variables is the URL of the page that referred the user to the current web page.

```php
$came_from = $_SERVER['HTTP_REFERER'];
```

Superglobals and security

- A **word of caution** is in order before you start using superglobal variables, because they are often used by hackers trying to find exploits to break into your website.
- What they do is load up $_POST, $_GET, or other superglobals with malicious code, such as Unix or MySQL commands that can damage or display sensitive data if you naively access them.
- You should always sanitize superglobals before using them.
  - One way to do this is via the PHP htmlentities function.
  - It converts all characters into HTML entities.
  - For example, less-than and greater-than characters (< and >) are transformed into the strings < and > so that they are rendered harmless, as are all quotes and backslashes, and so on.

```php
$came_from = htmlentities($_SERVER['HTTP_REFERER']);
```

PHP exercise 1

- PHP includes all the standard arithmetic operators. For this PHP exercise, you will use them along with variables to print equations to the browser. In your script, create the following variables:

```
$x=10;
$y=7;
```

- Write code to print out the following:

```
10 + 7 = 17
10 - 7 = 3
10 * 7 = 70
10 / 7 = 1.4285714285714
10 % 7 = 3
```

- Use numbers only in the above variable assignments, not in the echo statements.

PHP exercise 2

- In this PHP exercise, you will use a conditional statement to determine what gets printed to the browser.
- Write a script that gets the current month and prints one of the following responses, depending on whether it's August or not:

  It's August, so it's really hot.
  Not August, so at least not in the peak of the heat.

- *Hint*: the function to get the current month is *'date('F', time())'* for the month's full name.

PHP exercise 3

- Loops are very useful in creating lists and tables.
- In this PHP exercise, you will use a loop to create a list of equations for squares.
- Using a for loop, write a script that will send to the browser a list of squares for the numbers 1-12.
- Use the format, `"1 * 1 = 1"`, and be sure to include code to print each formula on a different line.

PHP exercise 4

- HTML tables involve a lot of repetitive coding - a perfect place to use for loops. You can do even more if you nest the for loops.
- In this PHP exercise, use two for loops, one nested inside another.
- Create the following multiplication table:

1	2	3	4	5	6	7
2	4	6	8	10	12	14
3	6	9	12	15	18	21
4	8	12	16	20	24	28
5	10	15	20	25	30	35
6	12	18	24	30	36	42
7	14	21	28	35	42	49

THANK YOU