

PHP CRASH COURSE by TzeJian Chear (jian2587@gmail.com)

Brief account of PHP

PHP is a server-side scripting language, chiefly processes queries and outputs customized responses to clients' browsers. It's commonly used together with database in web services like emails, forums, CMS, etc.

General structure of PHP

PHP codes are stored in .php files. PHP Codes must be in between <?php and ?> tags, just like HTML. All statements in PHP end with a semi-colon unless otherwise stated.

Output and Comments

```
<?php // This is a comment. Comments are ignored by the PHP interpreter, hence are invisible.
# You can prefix either double slash, //, or a hash symbol, #, to turn a line into comment.
/* This is a C style comment, with comments in between /* and */ tags. */
echo "Hello, World!\n"; //this outputs the sentence Hello, World! to the client's browser
/* \n is a form of escape character. \n will cause the next sentence to be printed on the next line. Other
forms of escape characters include \ for slash, \" for double quote, \t for tab, etc. */ ?>
```

Math expressions

```
echo 13.5 + 8; //this outputs the number 21.5.
echo 3 + 7 * 5; //this outputs the number 38. * is multiply, and it has higher precedence than +.
echo (12 + 18) / 4; //this outputs the number 7.5.
echo 20 % 6; // This outputs 2. % is the modulus sign, which calculates the remainder of a division.
```

Date/time

```
echo date("dmY"); //outputs current date in ddmmyyyy format. e.g. 30th june 2007 will be like 30062007
echo date("His"); //outputs current time in hhmmss format. e.g. 11:48pm and 4 seconds will be like 234804
echo date("m/d/Y-H:i:s"); //you can insert characters in between. Output will be like 06/30/2007-23:48:04.
```

Variables

```
$abc = 123; // This assigns the integer 123 to $abc. Variable names are prefixed with the dollar sign.
$distance = 45.76; //This assigns the number 45.76, which is a floating point number.
$myname = "TJ Davis"; // This is a string variable. A string is put in between double or single quotes.
$logged_in = TRUE; // This is a Boolean variable. It can contain either TRUE or FALSE.
$ghi = $abc + $distance; // This assigns the sum of $abc and $distance, 168.76 to $ghi
```

Strings

```
$lastname = "Felt" . "walt" . "er"; //The period is used to concatenate two strings.
$fullname = "TJ " . $lastname; // $fullname now stores "TJ Feltwalter"
echo "My name is $fullname.\n"; //If you use double quotes, the output will be: My name is TJ Feltwalter.
echo 'My name is $fullname.\n'; //If you use single quotes, the output will be: My name is $fullname.\n
```

Typecasting

```
$abc = 12.7; // Type casting changes the type of a variable, say, from integer to float.
$fuel = (integer) $abc; // $fuel now stores the integer 12, instead of the floating point number 12.7.
$s = (string) $fuel; /* $s now stores the string "12". Other casts include (float), (bool), (array) and (object).
```

Arrays

```
$weights = array(49, 62, 53, 45, 87); // stores a list of weights sequentially numbered in the array $weights.
$weights[2] = 11; // changes the third element's value from 53 to 11. Index starts from 0, so the third is 2.
echo $weights[5]; // this produces error, because the 6th element is not assigned any value.
$age = array("chris" => 23, "ian" => 19, "tina" => 28); // indexed by strings instead of sequential numbers.
$age["ian"] = 37; // we change ian's age to 37.
echo "Ian's age is {" . $age["ian"] . "}"; // array, if within quotes, must be enclosed between curly braces.
print_r($age); // print_r outputs name and value of each and every element of the array passed to it.
```

Functions

```
function sum($a, $b = 5) // this defines a function named sum, with the two parameters $a and $b.
{ // the codes of a function is enclosed within curly braces. The whole thing is called a function body.
  $c = $a + $b; // computes the sum of both $a and $b.
  return $c; // returns the result and exit the function as well.
}
echo "Sum of 3 and 14 is " . sum(3, 14); // the output will be: Sum of 3 and 14 is 17
echo "Sum of 8 and 5 is " . sum(8); // the second parameter is omitted, so it uses its default value, 5.
```

Variable Scope

```
$a = 17; $b = 12; $c = 34; // $a, $b and $c are global variables as they're not defined in a function.
function example() {
  $d = 36; // $d's scope is local to this function. $d cannot be accessed outside of this function.
  echo $a; // This produces error. $a despite being global, cannot be accessed, because it's not defined in here.
  global $b, $c; } // By prefixing the keyword global, both $b and $c can be accessed and modified in here.
echo "Your IP is {$_SERVER['REMOTE_ADDR']}."; // this gives you the client's IP address
/* Superglobals are variables accessible throughout the codes, and they are usually arrays. Other
superglobals include $_COOKIE, $ENV, $GLOBALS, etc. Use print_r to see their stored values. */
```

Comparisons

```
if ($a > $b) {
  // if $a is greater than $b, then the codes within the curly brace pair here will execute.
} else if (($a - 5) < $b) {
  // if $a, upon subtraction by 5, is less than $b, the codes here will execute instead of the above.
} else {
  // if every comparison above is false, the codes here will run instead. Using else if and/or else is optional.
} /* Other comparison signs include >= for greater-than-or-equal-to, <= for less-than-or-equal-to, == for
equal, and != for not equal. To compare strings exactly, you must use === for equal or !== for not equal */
```

Boolean Logic

```
if (($a > $b) && ($c == $d) || (!$g)) { // && means AND, || (double pipe) means OR, ! means NOT.
  // if both ($a > $b) AND ($c == $d) is TRUE, OR $g is FALSE, the codes here will execute.
} /* Another one is xor, for XOR. && requires both operands to be true, || requires either one or both to be
true, xor requires either one but not both to be true, and ! flips the operand from true to false, vice-versa. */
```

While Loops

```
$a = 10;
while($a > 0) { // this while loop repeatedly runs the codes within the curly braces so long ($a > 0) is true.
  echo "T minus $a.<br>"; // <br> will be interpreted by the client's browser as an HTML tag.
  $a -= 1; } // -= is a shortcut for $a = $a - 1. Others include += for self-adding, *= for self-multiplying, etc.
```

Do While Loops

```
do { // As the condition evaluation is after the codes, it'll loop atleast once, even if the condition is false.
  echo "T plus $a. <br>";
} while ($a++ <= 10); // if a is less than or equal to 10, continue looping. Increase $a by one as well.
```

For Loops

```
$primes = array(2, 3, 5, 7, 11, 13);
$num = count($primes); // count is a function which gives the total number of elements. Here, it's 6.
echo "The prime numbers below 15 are: ";
for ($i = 0; $i < $num; $i++) { // ++ means increment by one. -- means decrement by one.
  echo $primes[$i] . ' '; // the output of this code will be: The prime numbers below 15 are: 2,3,5,7,11,13,
} /* a for-loop has three sections, separated by a semi-colon. First one initializes variables, second part is a
condition. The loop continues so long the condition is true. Third part is an iterator, which gets executed
once every loop. Each section is optional and may be omitted. */
```

Break and Continue

```
$i = 0;
while (TRUE) {
if ($i > 10) break; // break causes the loop to halt, and takes the code execution out of the while block.
if ($i % 2 == 1) continue; // continue causes the loop to skip once and continue on the next iteration.
echo "$i,"; // because of continue, the output will be only even numbers: 0,2,4,6,8,10,
$i++; } // by the way, break and continue works with do while loops and for loops as well
```

Switch

```
Switch($drinks) { // switch compares a variable to many different values. Must be enclosed in curly braces.
case "lemonade": // $drinks === "lemonade"
    echo "I love lemonade."; // if $drinks equals "lemonade", this line will execute.
    break; // break takes the code execution out of the switch block
case "sky juice": // if $drinks equals "sky juice", the line after will execute.
    echo "Sky juice is water, and "; // there is no break; The code falls through
case "water": // $drinks === "water"
    echo "water is good!"; // if drinks equals "water", the line after will execute.
    break;
default: // the default code if none of the above is right. default is optional.
    echo "I don't like $drinks."; // no break needed; because default is the last case.
}
```

String Processing

```
echo substr("abcdefg", 0, 2); // This extracts 2 characters starting from position zero, which is ab.
echo substr("abcdefg", 3); // This extracts everything starting from position 3, which is defg.
echo str_repeat("#", 5); // This repeats "#" 5 times. The output is #####.
echo strchr("gordon", "o"); // This gets the string starting from the first occurrence of 'o', which is ordon.
echo strrchr("alligator", 'a'); // This gets the string starting from the last occurrence of 'a', which is ator.
echo trim(" hello "); // This trims the spaces before and after the string, which is hello.
echo addslashes("\hello\n"); // add slashes before escape characters. Useful for storing data in database.
$a = explode(',', "ian,tim,ed"); // separates ian,tim and ed by comma and store them individually in an array.
$s = implode('-', $a); // rejoins separate items in an array as a string, with each item separated by a dash.
```

HTML Forms (Post and Get)

```
<?php /* we can use $_POST or $_GET to obtain data from user through web forms. Get method appends
the data behind the URI, e.g. www.abc.com/tst.php?name=tina&gender=f. Each argument is separated by
an ampersand. Post method is used for bigger set of data, and the data submitted, unlike Get, is invisible. */
if (isset($_POST['age'])) { // isset is a function which returns true if the variable exists
    echo "POST: You are born in the year " . ((int)date("Y") - (int)$_POST['age']) . ' ';
} else if (isset($_GET['name'])) {
    echo "GET: Hello, {$_GET['name']}!";
} //Alternatively, $_REQUEST works for both Post and Get method. ?>
<!-- $_SERVER['PHP_SELF'] gives you the name of this file. -->
<form method="get" action="<?php echo $_SERVER['PHP_SELF']; ?>"> <!-- using get method -->
Name: <input name="name" type="text"><br>
<input name="getname" type="submit" value="Send"></form><br>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>"> <!-- using post method -->
Age: <input name="age" type="text"><br>
<input name="postage" type="submit" value="Send"></form>
```

Include

```
include "externalfile.php"; /* use include to include the codes in other files. The files' global variables will
be visible to the file that includes them. */
include_once "externalfile.php"; /* If you include both b.php and c.php, but c.php also includes b.php,
error will result. use include_once to avoid including the file more than once. */
```