PHP Basic

PHP Syntax

- PHP Syntax is simple and easy to learn.
- PHP Syntax is derived from many languages (e.g. Java, Perl, C and others).
- PHP code can be directly inserted into processed text files (e.g. XML, HTML etc.) using special tags (AKA PHP Source Files Tags).

PHP Source Files Tags

- The PHP source files tags allow embedding PHP code within processed text files (HTML, XML etc.).
- There are four type of PHP source files tags:

Standard Tags

```
<?php  ...  ?>
```

Short Tags

```
<? ... ?>  <?= $variable ?>
```

Script Tags

```
<script language=”php”>  ...   </script>
```

ASP Tags

```
<% .... %>
```

PHP Source Files Short Tags

- PHP 5.4 supports the following short tags by default. We don't need to introduce any change in `php.ini` in order to use them.

```
<?
…
?>
```

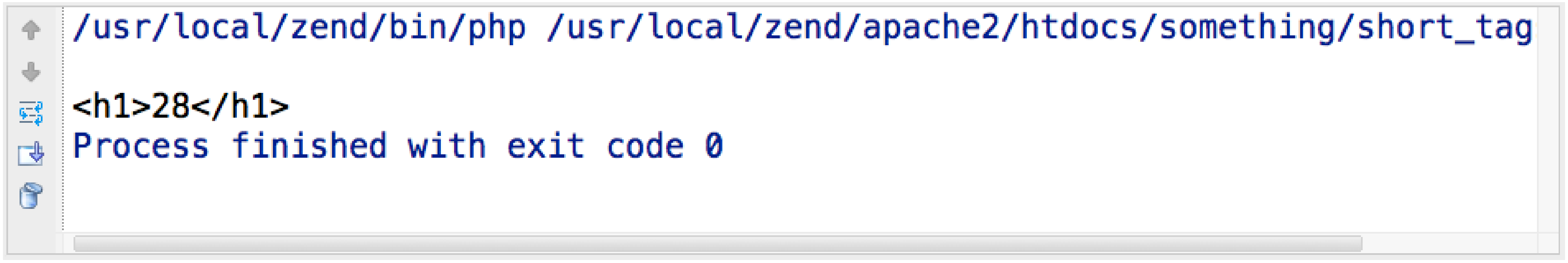
and

```
<?= expression ?>
```

PHP Source Files Short Tags

```
<?
$numA = 24;
$numB = 4;
?>

<h1><?=($numA+$numB)?></h1>
```

PHP Source Files Short Tags



```
/usr/local/zend/bin/php /usr/local/zend/apache2/htdocs/something/short_tag

<h1>28</h1>
Process finished with exit code 0
```

Script Structure

- The PHP script is composed of statements such as function calls, variable assignments etc.
- In most cases, a PHP statement should end with a semi colon, ';'.

Comments

- PHP allows four different syntax possibilities to write a comment inside the code.

```
// single line comment

# single line comment

/* multi line comment
   multi line comment */

/**
*  API comment
*/
```

Whitespace

- PHP is a whitespace insensitive language. We can include as many spaces as we want. It won't effect the execution of our code.

Compound Statement

- A compound statement (AKA “Code Block”) is a simple series of statements enclosed between two braces.

```
{
    $a = 12;
    $b = 14;
    $sum = $a + $b;
}
```

The `echo` Statement

- The `echo` statement is a built-in language command. This is not a function. Using `echo` we can write data back to the script's output.

```
echo “Hello”;  // will outout Hello
```

PHP Data Types

- PHP supports various different data types, categorized into two categories.
- The two most important categories are “Compound Data Types” & “Scalar Data Types”.

PHP Scalar Data Types

- A PHP scalar data type includes one value.
- PHP supports four scalar types:

  boolean

  A boolean can be 'true' or 'false' only.

  int

  An int is a signed numeric integer value.

  float

  A float is signed floating point value.

  string

  A string is a collection of binary data.

PHP Scalar Data Types

- A PHP scalar data type includes one value.
- PHP supports four scalar types:

  boolean

  A boolean can be 'true' or 'false' only.

  int

  An int is a signed numeric integer value.

  float

  A float is signed floating point value.

  string

  A string is a collection of binary data.

Binary Number Format

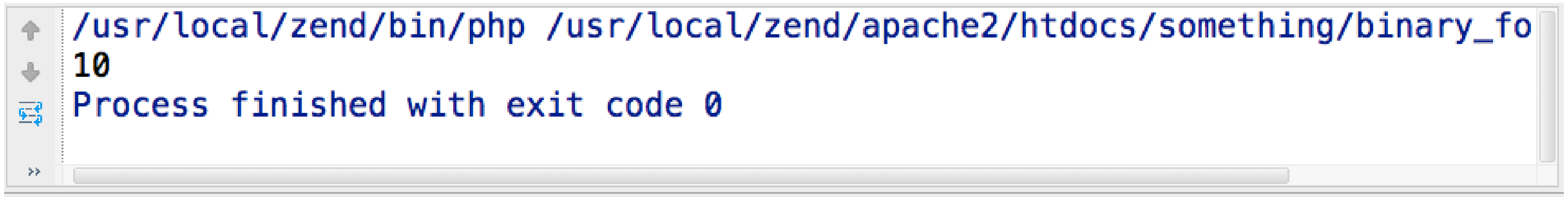
- As of PHP 5.4 we can write binary numbers using the following syntax:

```
$num = 0b000101001010;
```

Binary Number Format

```
<?
$a = 0b1110; //14
$b = 0b1011; //11
$c = $a & $b; //0b1010
echo $c;
?>
```

Binary Number Format



```
/usr/local/zend/bin/php /usr/local/zend/apache2/htdocs/something/binary_fo
10
Process finished with exit code 0
```

PHP Compound Data Types

- A PHP compound data type can include more than one value.
- PHP supports two compound data types:

  Arrays

  An array is a container of ordered data elements. These data elements can be of any type.

  Objects

  An object is a container of data together with code.

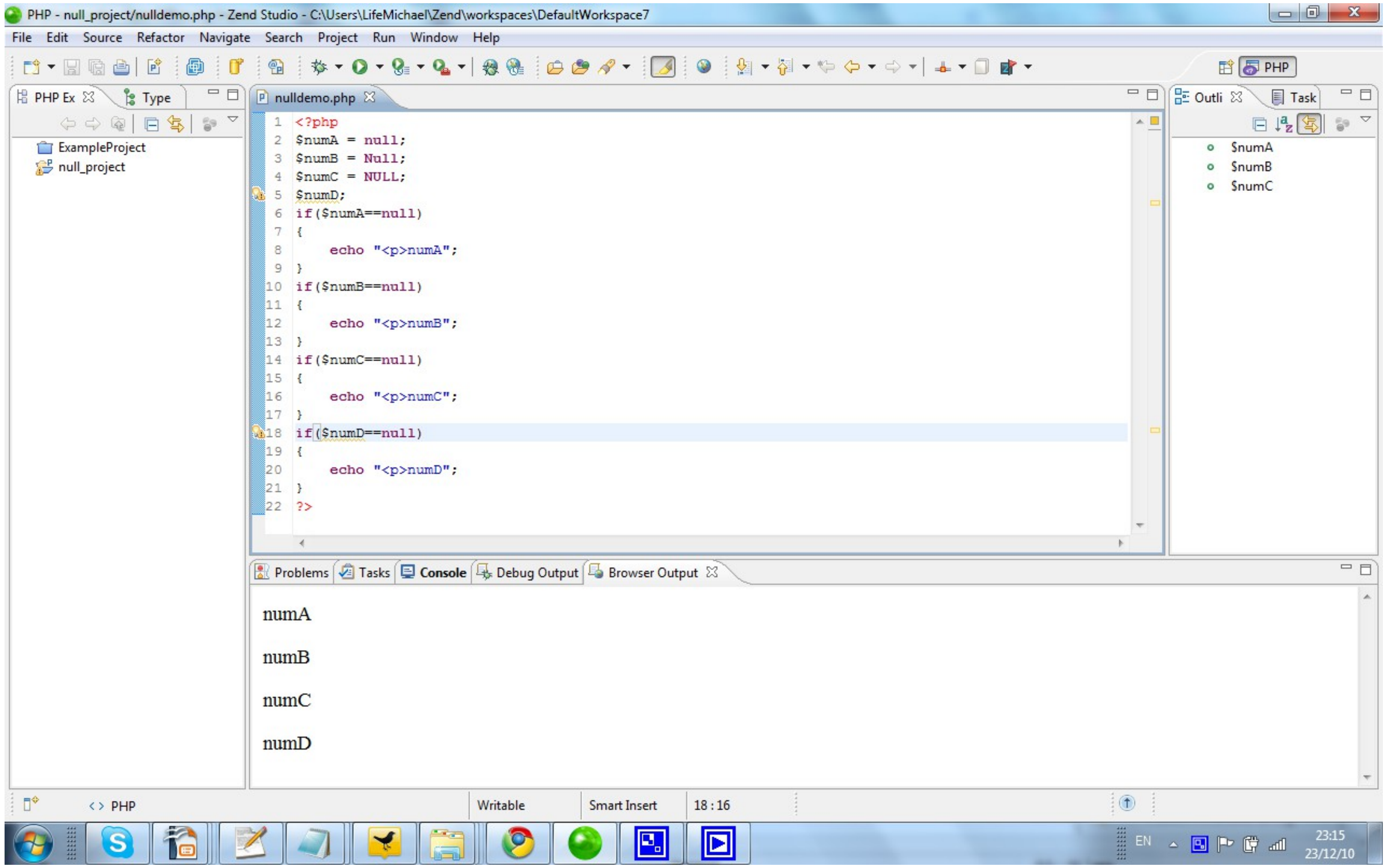
The `null` Data Type

- The `null` keyword is a special PHP Data Type, and its purpose is to indicate that a variable has no value.
- A variable is considered to be `null` if it has been assigned with the special `null` value or if it still hasn't been assigned a value.
- The `null` value can be expressed using any of the following possible keywords: `Null`, `null`, `NULL`.

The `null` Data Type

```
<?php
$numA = null;
$numB = Null;
$numC = NULL;
$numD;
if($numA==null)
{
    echo "<p>numA";
}
if($numB==null)
{
    echo "<p>numB";
}
if($numC==null)
{
    echo "<p>numC";
}
if($numD==null)
{
    echo "<p>numD";
}
?>
```

The `null` Data Type



PHP - null_project/nulldemo.php - Zend Studio - C:\Users\LifeMichael\Zend\workspaces\DefaultWorkspace7
File Edit Source Refactor Navigate Search Project Run Window Help
PHP Ex
Type
ExampleProject
null_project
nulldemo.php
Outli
Task
$numA
$numB
$numC
Problems
Tasks
Console
Debug Output
Browser Output
PHP
Writable
Smart Insert
18 : 16

```php
<?php
$numA = null;
$numB = Null;
$numC = NULL;
$numD;
if($numA==null)
{
	echo "<p>numA";
}
if($numB==null)
{
	echo "<p>numB";
}
if($numC==null)
{
	echo "<p>numC";
}
if($numD==null)
{
	echo "<p>numD";
}
?>
```

```
numA

numB

numC

numD
```

The Resource Data Type

- The Resource is a special PHP Data Type that refers to external resource (e.g. file, image etc.) which is not part of the PHP native language.

The Type Conversion Operator

- Converting the data type of a given expression to another data type is done by writing the name of the type to which we want to convert within brackets and place them before the expression.

```
$num1 = 10.5;
$num2 = 10.8;
$num3 = ((int)$num1)+((int)$num2);
echo $num3;  //output would be 20
```

Variables

- A variables is a temporary containers that can hold a value.
- A variable can hold any type of data (e.g. strings, integers, objects etc.).
- PHP is loosely typed programming language.
- We identify the variables by adding the dollar sign $ before their name.
- Variables names must include letters (a-z,A-Z), numbers and underscores only.

Variables

- A variables name must start either with a letter or an underscore.
- PHP Variables names are case sensitive.

$_num1	OK
~~$2num~~	NOT OK
$number12	OK

Variable Variables

- A variable variables is a variable that its name is contained within another variable.

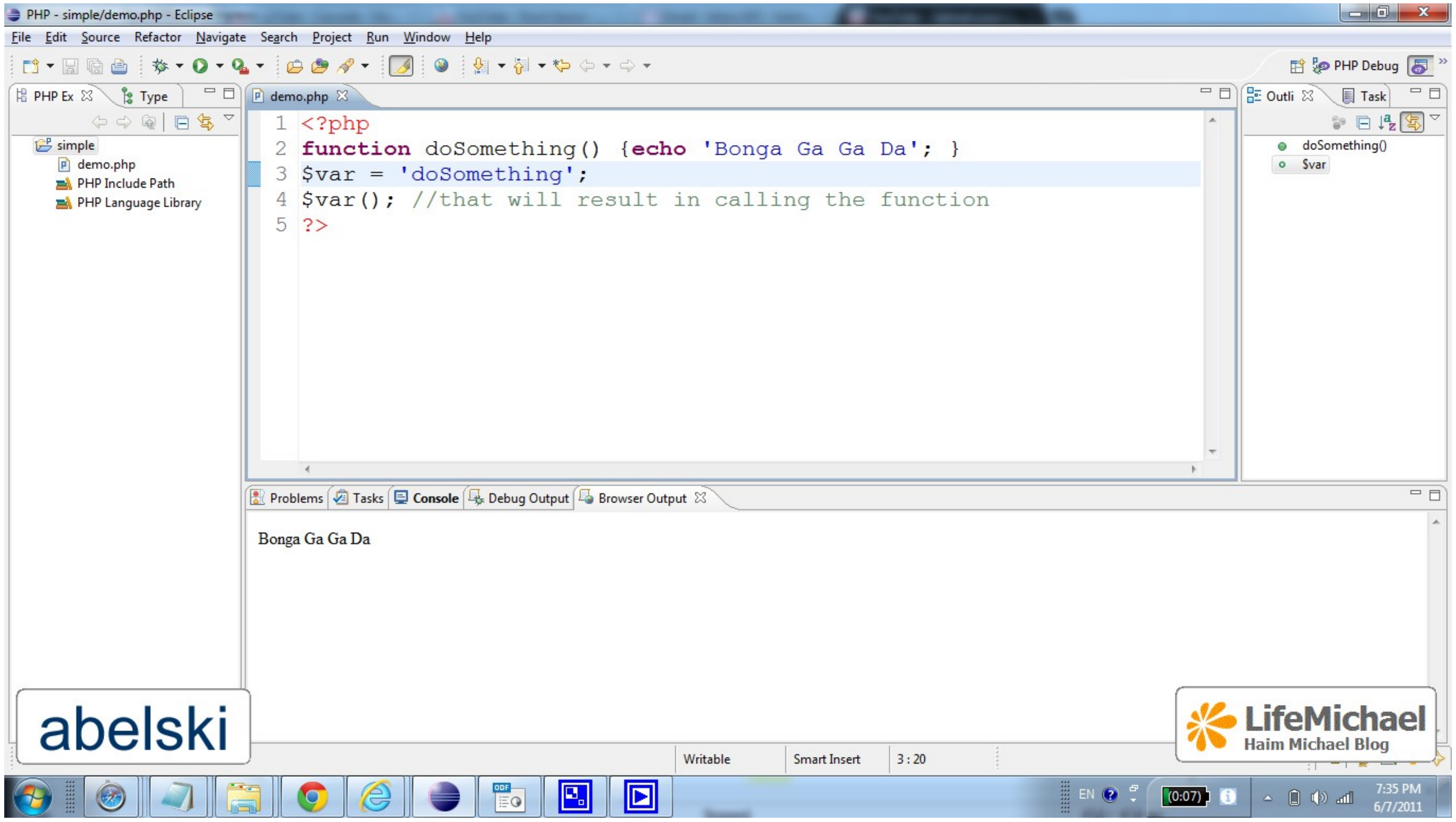
```
<?php
$var = 'abc';
$$var = 'hello';
echo $abc;   //that should display 'hello'
?>
```

Function Name Within Variable

- We can assign a function name to be the value of a variable we have. We can later use that variable in order to call the function.

```
<?php
function doSomething() {echo 'Bonga Da'; }
$var = 'doSomething';
$var(); //that will result in calling the function
?>
```

Function Name Within Variable



```php
<?php
function doSomething() {echo 'Bonga Ga Ga Da'; }
$var = 'doSomething';
$var(); //that will result in calling the function
?>
```

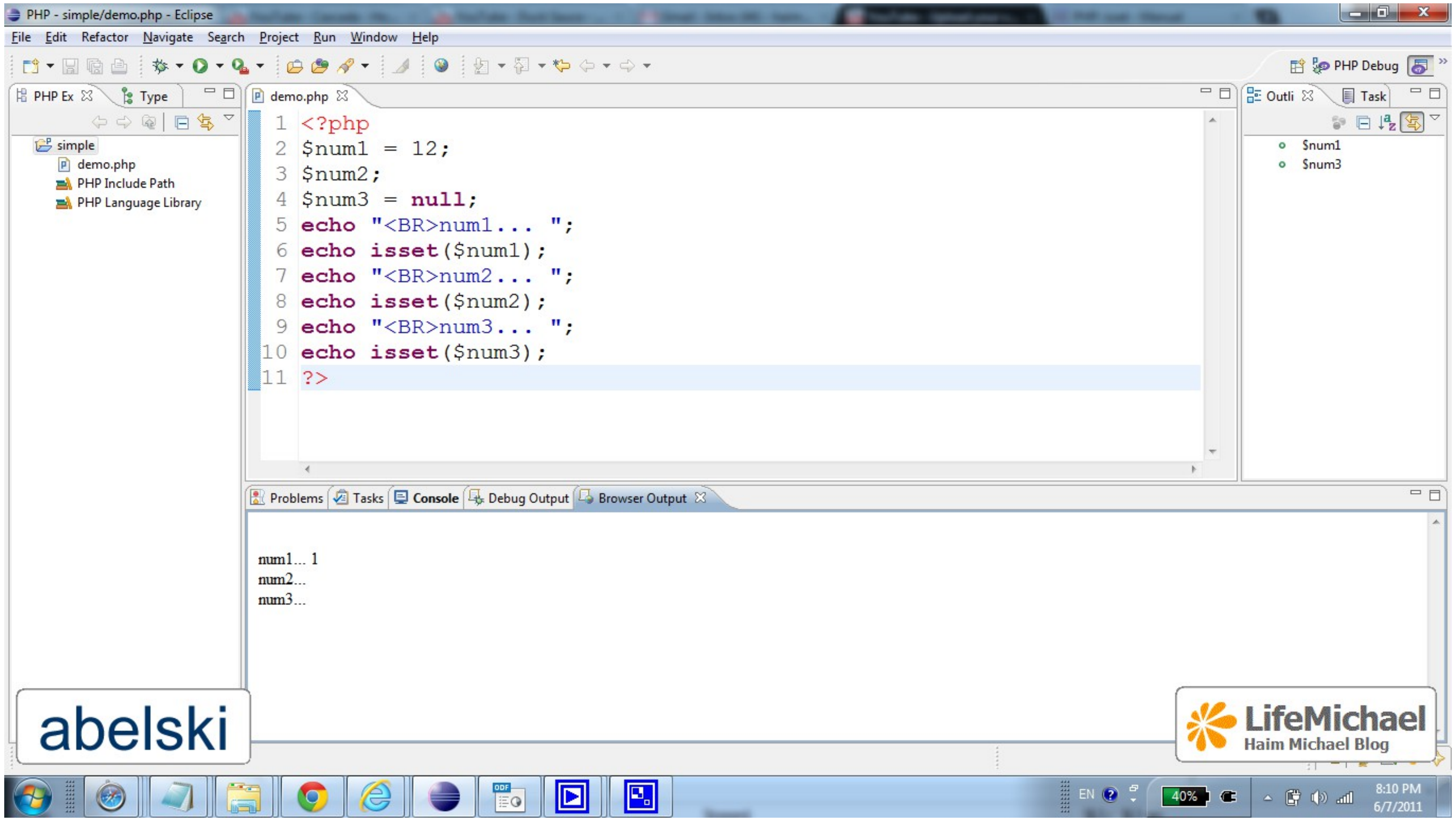
```
Bonga Ga Ga Da
```

Variables Existence Validation

❖ Using the `isset($var)` function we can verify a required variable does exist before we try to use it. If the variable exists and has a value other than NULL we should get true.

```
<?php
$num1 = 12;
$num2;
$num3 = null;
echo "<BR>num1... ";
echo isset($num1);
echo "<BR>num2... ";
echo isset($num2);
echo "<BR>num3... ";
echo isset($num3);
?>
```

Variables Existence Validation



```
<?php
$num1 = 12;
$num2;
$num3 = null;
echo "<BR>num1... ";
echo isset($num1);
echo "<BR>num2... ";
echo isset($num2);
echo "<BR>num3... ";
echo isset($num3);
?>
```

```
num1... 1
num2...
num3...
```

Constants

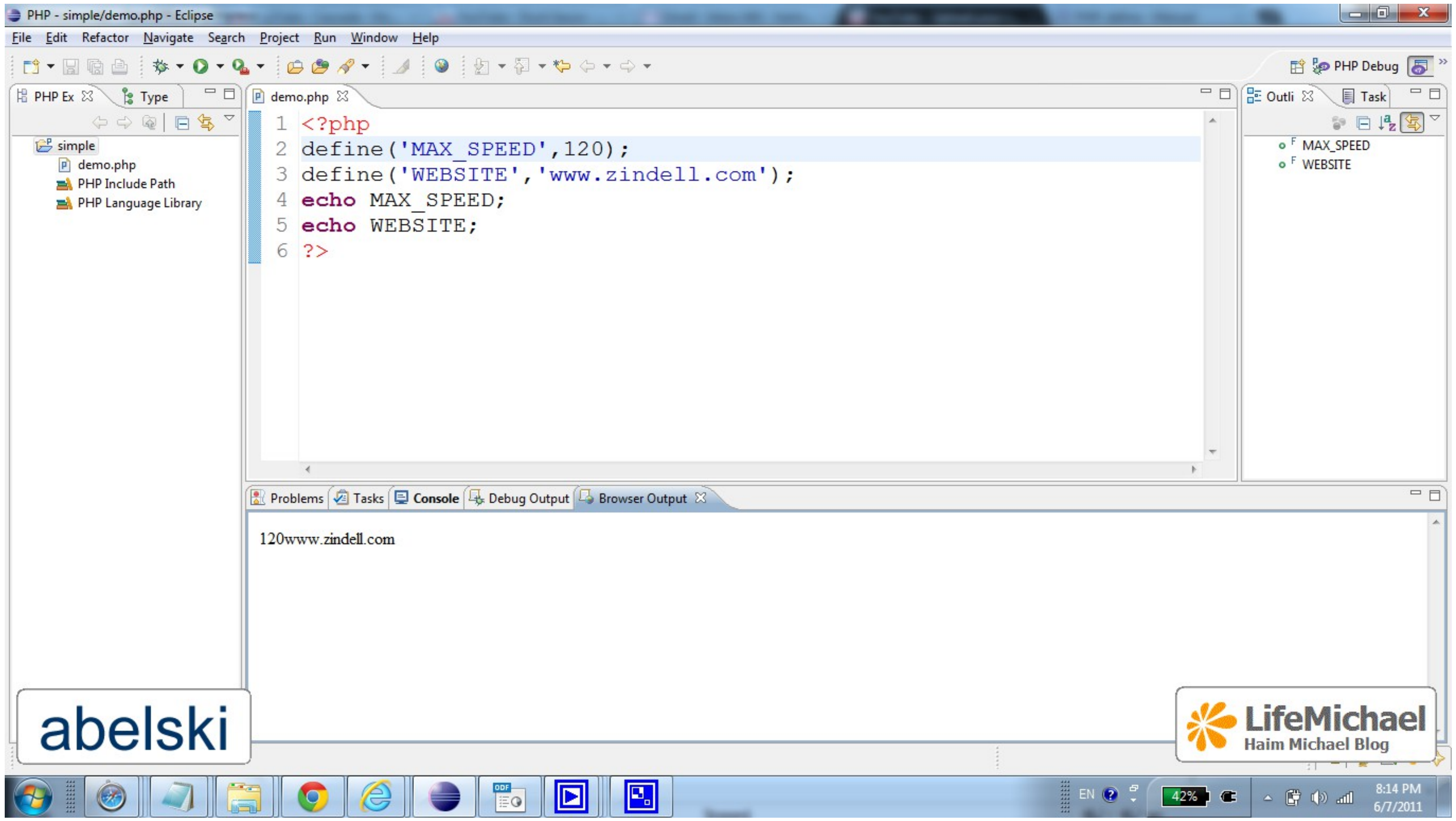
- Constants are immutable values.
- Constants in PHP can hold scalar data types only.
- As with variables, constants names are case sensitive.
- The rules for naming constants are the same rules for naming variables (except for the leading $).
- Using upper case when defining constants is a common practice.

Constants

- In order to define a constant we need to use the 'define' function in the following way:
  define('CONSTANT_NAME', 'constant_value');

```
<?php
define('MAX_SPEED',120);
define('WEBSITE','www.zindell.com');
echo MAX_SPEED;
echo WEBSITE;
?>
```

Constants



```php
<?php
define('MAX_SPEED',120);
define('WEBSITE','www.zindell.com');
echo MAX_SPEED;
echo WEBSITE;
?>
```

```
120www.zindell.com
```

Operators

- PHP has the following types of operators:

  Assignment Operators

  Arithmetic Operators

  String Operators

  Comparison Operators

  Logical Operators

  Bitwise Operators

  Error Control Operator

  Execution Operator

  Incrementing / Decrementing Operators

  Type Operators

Arithmetic Operators

- Perform basic mathematical operations:

+	Addition	$num = 24 + 2;
-	Subtraction	$num = 24-3;
*	Multiplication	$num = 4*5;
/	Division	$num = 40/8;
%	Modules	$num = 23 % 7;

Incrementing / Decrementing Operators

- These are unary operators (work on one operand only) that work on a variable and increment/decrement its value by 1.
- Their notation is ++ and - -.
- If placed before the variable then the variable is first been incremented/decremented and then it is evaluated.
- If placed after the variable then the variable is first been evaluated and then it is incremented/decremented.

Incrementing / Decrementing Operators

```
<?php
$num1 = 12;
$num2 = 24;
$num3 = 32;
$num4 = ++$num2;
$num5 = $num3++;
$num6 = $num1--;
echo
"<BR>num1=$num1";
echo
"<BR>num2=$num2";
echo
"<BR>num3=$num3";
echo
"<BR>num4=$num4";
echo
"<BR>num5=$num5";
echo
"<BR>num6=$num6";
?>
```

num5 is first getting the old value of num3 num3 is incremented afterwards.

num6 is first getting the old value of num1 num1 is decremented afterwards.

String Operators

- The concatenation operator allows us concatenate two separated strings into one.
- The string concatenation operator is a simple dot '.'.

```
<?php
$var1 = "Hello";
$var2 = "World!";
$total = $var1 . $var2;
echo "total=$total";
?>
```

Bitwise Operators

- These operators allow manipulating bits of data.

  &

  Bitwise AND. Each bit will be set if (and only if) it is set in both operands.

  |

  Bitwise OR. Each bit will be set if it is set at least in one of the operands.

  ^

  Bitwise XOR. Each bit will be set if (and only if) it is set in one of the operands only.

  >>

  Bitwise right shift. Unset bits are inserted in the shifted positions.

  <<

  Bitwise left shift. Unset bits are inserted in the shifted positions.

Assignment Operators

- This is the simple '=' used to assign a value inside a variable.

```
$var = 24+3;
$var = $var + 3;
```

- The assignment operator works 'by value'. Adding '&' before the other variable its value is assigned to our variable, will perform a 'by reference' assignment.

```
$var = 24;
$num = &$var;
$var=6;
echo $num;     // The output will be 6.
```

Comparison Operators

❖ Perform basic mathematical operations:

==

Equivalence (e.g. if(num1==num2)). This operator returns 'true' if the two operands are of the same data type or can be converted to a common data type, and have the same value in that type.

===

Identity (e.g. if(num1===num2)). This operator returns 'true' if the two operands are of the same data type and have the same value in that type.

!=

Non Equivalent (e.g. if(num1!=num2)). This operator returns 'true' if the two operands are not equivalent. Their data type is not important.

Comparison Operators

!==

Non Identical (e.g. if(num1!==num2)). This operator returns 'true' if the two operands are not identical.

<

Less Than (e.g. if(num1<num2)). This operator returns 'true' if the right operand is less than the right one.

<=

Less Than or Equal (e.g. if(num1<=num2)). This operator returns 'true' if the right operand is less than or equal the right one.

Comparison Operators

>

Bigger Than (e.g. if(num1>num2)). This operator returns 'true' if the left operand is bigger than the right one.

>=

Bigger Than or Equal (e.g. if(num1>=num2)). This operator returns 'true' if the left operand is bigger than or equal the right one.

Logical Operators

- Binary logical operators that connect separated boolean values:

  &&

  Evaluates to true if both the right and left operands evaluate to true.

  ||

  Evaluates to true if at least one of the right and left operands evaluate to true.

  ^

  Evaluates to true if one (and only one) of the right and left operands evaluate to true.

- Unary logical operator that works on one operand:

  !

  Returns true if the operand is false and returns false if the operand is true.

Error Control Operator

- Adding the error suppression operator `@` to expression will cause PHP runtime environment to ignore nearly all error messages that occur during this expression evaluation.

```
$var = @mysql_connect();
```

Execution Operator

- Using the backtick operator (`...`) it is possible to execute code directly on the operation system, as if it was written in the command line.

```
$temp = `ls`;
```

Operators Precedence & Associativity

Associativity	Operator
left	[
non associative	++
non associative	! ~ - (int) (float) (string) (array) (object) @
left	* / %
left	+ - .
left	<< >>
non associative	< <= > >=
non associative	== != === !==
left	&
left	^
left	|
left	&&
left	||
left	? :
right	= += -= *= /= .= %= &= |= ^== <<= >>=
left	and
left	xor
left	or
left	,

PHP Shorthand Operators

- Similarly to other software programming languages, PHP allows using the operators in the following shorthand way.

  Given an expression with the following structure:

  ```
  [Variable Name] = [Variable Name] [Operator] [Expression]
  ```

  We can get the same outcome using the following syntax:

  ```
  [Variable Name] [Operator]= [Expression]
  ```

- The following are examples for this shorthand possibility.

  `$var+=12;` is the same as `$var=$var+12;`

  `$var%=5;` is the same as `$var=$var%5;`

Control Structures

- PHP supports most of the common control structures you know from other languages.
- In addition, PHP supports unique control structures that simplify script development.

The `if` and `if-else` Statements

- The well known `if` and `if..else` statements function similarly as in most other languages.

```
if(expression1)
{
    ...
}
else
{
    ...
}
```

The Ternary Operator

- The ternary operator enables embedding an if-then-else statement inside one expression.

```
$temp=(expression)?'yes':'no'
```

The Switch Case Statement

❖ The switch case statement in PHP works similarly to the switch case construct in Java / C / C++.

```
switch($data)
{
    case ___:
               ...
               break;
    case ___:
               ...
               break;
    default:
               ...
}
```

The `while` Statement

- The `while` statement in PHP works similarly to the `while` statement in Java / C / C++.

```
while(boolean_expression)
{
    ...
    ...
    ...
}
```

The `do..while` Statement

- The `do..while` statement in PHP works similarly to the `do..while` statement in Java / C / C++.

```
do
{
    ...
    ...
    ...
}
while(boolean_expression)
```

The `for(..;..;..)` Statement

- The `for(..;..;..)` statement in PHP works similarly to the `for(..;..;..)` statement in Java / C / C++.

```
for(exp_1; boolean_exp; exp_2)
{
    ...
    ...

}
```

The `break` Keyword

❖ The `break` keyword in PHP works similarly to the `break` keyword in Java / C / C++.

```
for(exp_1; boolean_exp; exp_2)
{
    ...
    ...
    if(...) break;
    ...
}
```

The `break` Keyword

- The `break` keyword in PHP has an optional parameter through which we can exit both from this loop and from the other loop\s surrounding it.

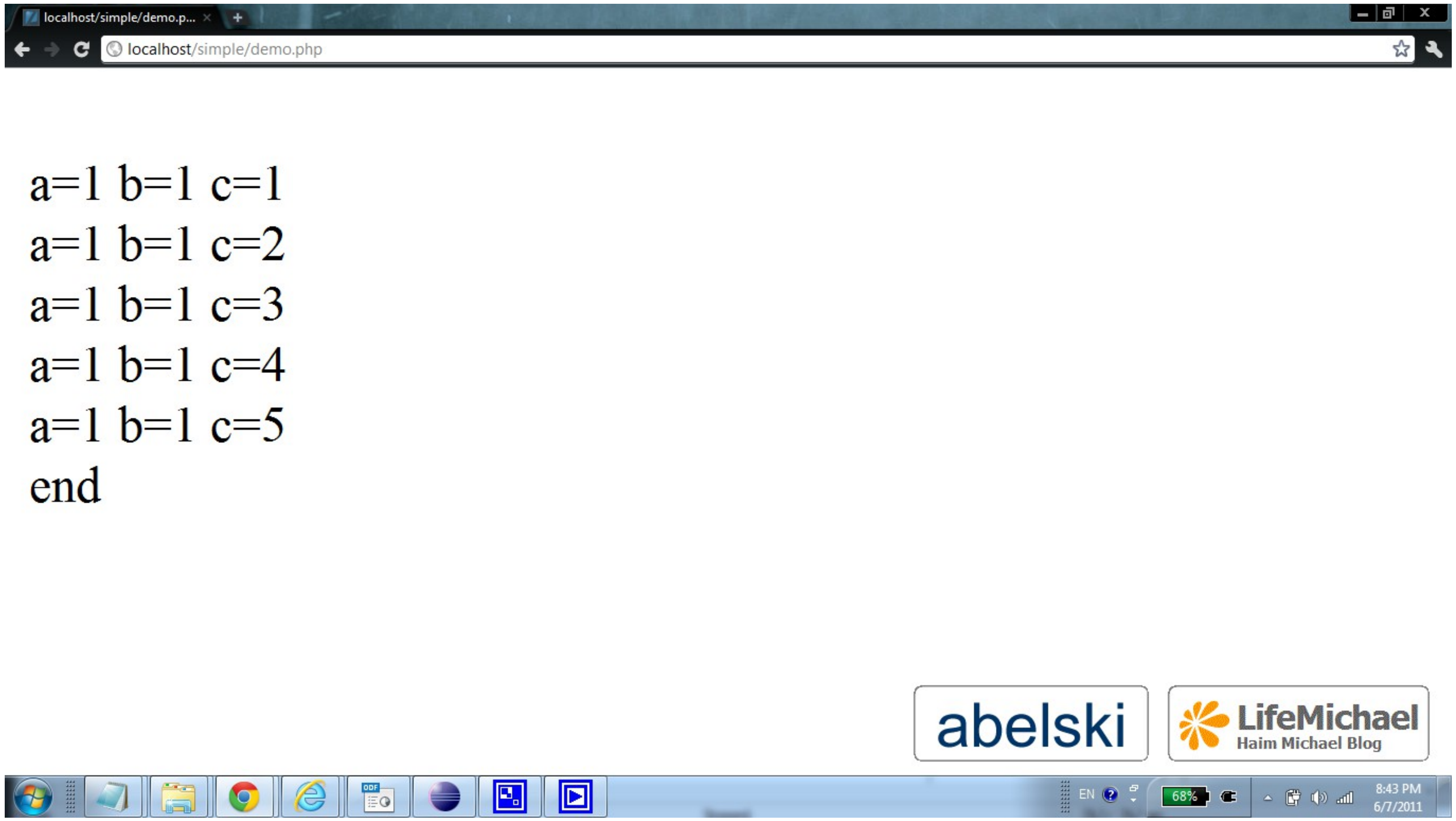
```
for(exp_1; boolean_exp; exp_2)
{
   for(exp_1; boolean_exp; exp_2)
   {
      ...
      if(...) break 2;  //exit both loops
   }
}
```

as of PHP 5.4 it is no longer possible to write variable arguments after the break keyword. static arguments still work. as a side effect of this change it is no longer possible to use the 0 value.

The `break` Keyword

```php
<?php
for($a=1; $a<=10; $a++)
{
    for($b=1; $b<=10; $b++)
    {
        for($c=1; $c<=10; $c++)
        {
            echo "<br>"."a=".$a." b=".$b." c=".$c;
            if($c==5) break 3;
        }
    }
}
echo "<br/>end";
?>
```

The `break` Keyword



```
a=1 b=1 c=1
a=1 b=1 c=2
a=1 b=1 c=3
a=1 b=1 c=4
a=1 b=1 c=5
end
```

The `continue` Keyword

- The `continue` keyword in PHP works similarly to the `continue` keyword in Java / C / C++. Similarly to `break` we can append it with a number in order to specify which loop we want to continue to its next iteration.

```
for(exp_1; boolean_exp; exp_2)
{
    ...
    ...
    if(...) continue;

}
```

The `include` Statement

- The `include` function allows us to include within the current PHP file another PHP file.
- Useful when there is another PHP file that includes the definition of functions\classes (or global variables) we want to use.

```
<?php
include('another_file.php');
...
?>
```

The `require` Statement

- The `require` function works the same as `include` with one difference. In both cases, when errors occur a warning message is produced. When using `require` we might also get a fatal error.

The `include_once` Statement

- The `include_once` function works the same as `include` with one difference. If the other PHP file was already included it won't be included again.

```
<?php
include_once('another_file.php');
...
?>
```

The `require_once` Statement

- The `require_once` function works the same as `require` with one difference. If the other PHP file was already included it won't be included again.

```
<?php
require_once('another_file.php');
...
?>
```

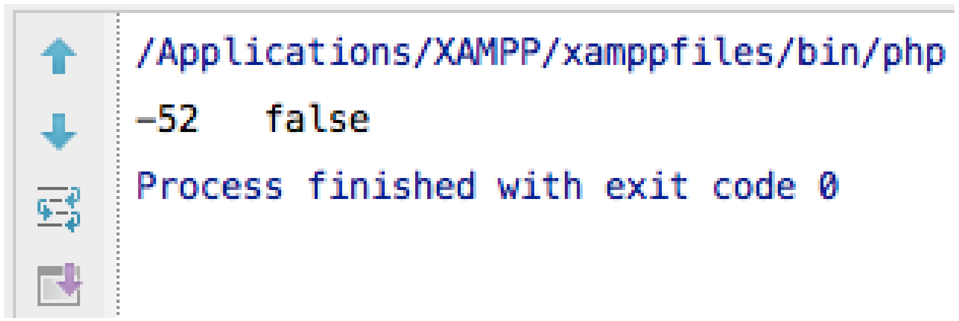
The `empty()` Function

- This function receives a variable and returns true if that variable is considered to be empty. The variable is considered to be empty if it doesn't exist or if its value is false.
- As of PHP5.5 we can pass over to this function an expression. If the expressions is evaluated to false then the empty function will return true.

The `empty()` Function

```
<?php
function checknum($num) {
    if($num>0) return true; else return false;
}

if (empty(checknum(42))) {
    echo "42   ";
}

if (empty(checknum(-52))) {
    echo "-52   ";
}

if (empty(false)) {
    echo "false   ";
}

if (empty(true)) {
    echo "true   ";
}
?>
```

The `empty()` Function



```
/Applications/XAMPP/xamppfiles/bin/php
-52   false
Process finished with exit code 0
```

The Output

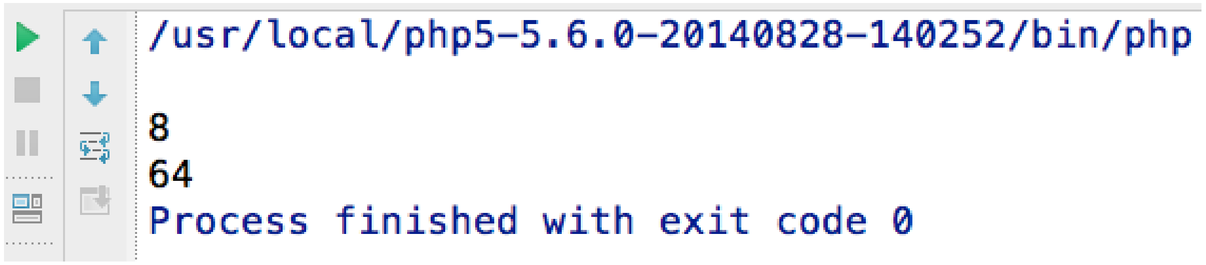
The Exponentiation Operator

- As of PHP 5.6, the ** exponentiation operator allows us to calculate the exponentiation of two numbers.

The Exponentiation Operator

```php
<?php
$number = 2;
$result = $number ** 3;
echo "\n".$result;
$num = 2;
$num **= 3; //$num = $num ** 3
$num **= 2; //$num = $num ** 2
echo "\n".$num;
?>
```

The Exponentiation Operator



```
/usr/local/php5-5.6.0-20140828-140252/bin/php

8
64
Process finished with exit code 0
```

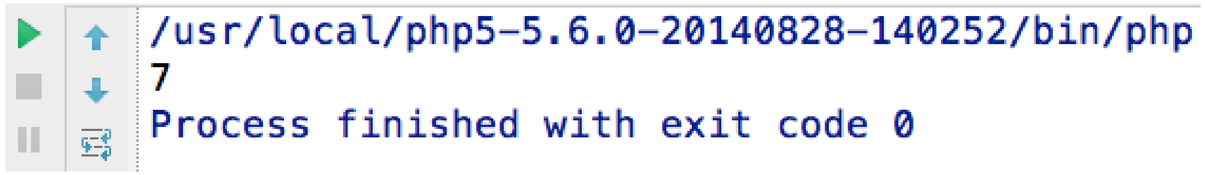
Constants Scalar Expressions

- As of PHP 5.6, when creating a constant we can assign it with a value of expression that includes the use of other constants and scalars.

Constants Scalar Expressions

```
<?php
const SUNDAY = 1;
const MONDAY = SUNDAY + 1;

class Something {
    const TUESDAY = MONDAY + 1;
    const FRIDAY = 2 * Something::TUESDAY;
    const STR = 'The value of FRIDAY is '.Something::FRIDAY;

    public function getSeventhDay($number = Something::FRIDAY + 1)
    {
        return $number;
    }
}

echo (new Something())->getSeventhDay();
?>
```

Constants Scalar Expressions


```
/usr/local/php5-5.6.0-20140828-140252/bin/php
7
Process finished with exit code 0
```

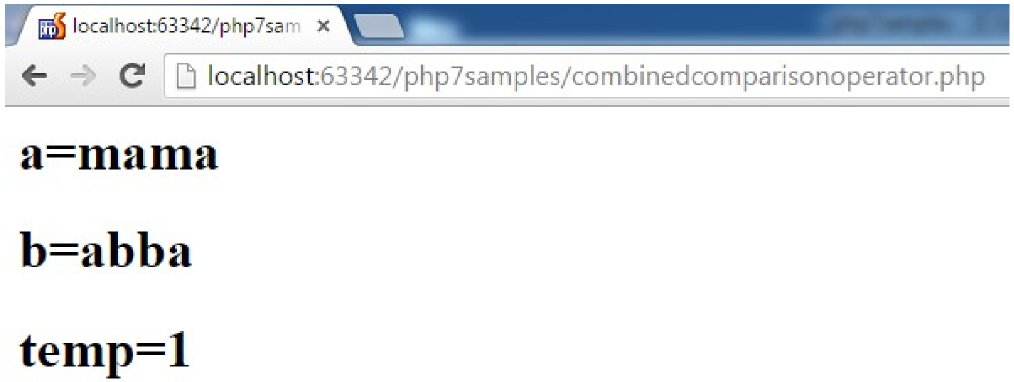
The <=> Operator

- The <=> operator is known as the combined comparison operator. Its other name is the spaceship operator.
- It is a shorthand for performing three way comparisons on two operands. The returned value is an integer, that can be either positive, negative or 0.

The <=> Operator

```
<?php
$a = "mama";
$b = "abba";
echo "<h1>a=$a</h1>";
echo "<h1>b=$b</h1>";
$temp = $a <=> $b;
echo "<h1>temp=$temp</h1>";
?>
```

The <=> Operator



localhost:63342/php7sam

localhost:63342/php7samples/combinedcomparisonoperator.php

```
a=mama

b=abba

temp=1
```

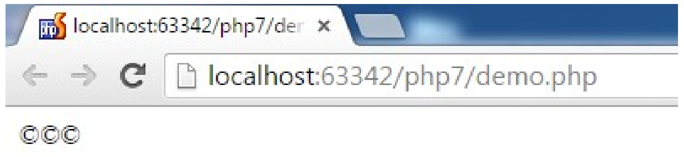
The <=> Operator

- When using the <=> operator for comparing strings the comparison will be a lexicographic one.
- We can use this operator for comparing arrays. The comparison will be between the elements.
- We cannot use it for comparing objects.

Unicode

- PHP 7 allows us to refer specific characters in the unicode table.

```
<?php
echo "\u{0000a9}";
echo "\u{00a9}";
echo "\u{a9}";
?>
```



```
©©©
```

The `IntlChar` Class

- This new class includes the definition for various static methods and constants that assist with manipulating unicode characters.

```
echo IntlChar::charName('@');
var_dump(IntlChar::ispunct('!'));
```

- In order to use this class we should install the Intl extension.

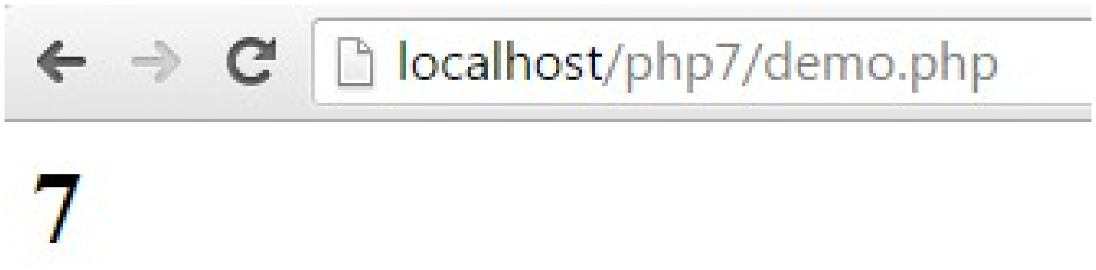
The `intdiv()` Function

- Using this new function, that was introduced by PHP 7, we can divide two int numbers and get a result, which is an int number as well.

The `intdiv()` Function

```
<?php
$a = 30;
$b = 4;
$c = intdiv($a,$b);
//$c = $a / $b;
echo "<h1>".$c."</h1>";
?>
```

The `intdiv()` Function



```
7
```

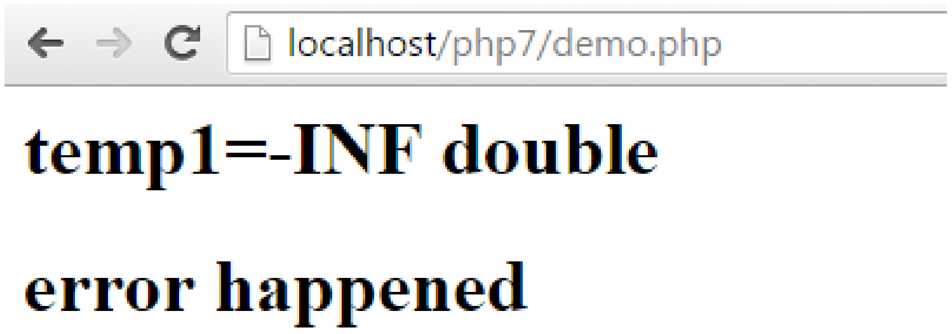
Division By Zero Changes

- Before PHP 7, when dividing by 0 or calculating modulo by 0 we got the value false of the type boolean.
- As of PHP 7, when calculating the modulo by 0 the DivisionByZeroError exception will be thrown and when trying to divide by 0 we will get +INF, -INT or NAN.

Division By Zero Changes

```php
<?php
$a = -512;
$b = 0;
$temp1 = $a / $b;
echo "<h1>temp1=".$temp1."     ".gettype($temp1)."</h1>";
try
{
    $temp2 = $a % $b;
    echo "<h1>temp2=" . $temp2 . "     " . gettype($temp2) . "</h1>";
}
catch(Throwable $throwable)
{
    echo "<h1>error happened</h1>";
}
?>
```

Division By Zero Changes



```
temp1=-INF double

error happened
```

Numerical Strings Hex Support

❖ As of PHP 7, strings we create that include hexadecimal numbers can no longer recognized as numerical.