PHP Case Sensitivity

In PHP, keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are not case-sensitive.

In the example below, all three echo statements below are equal and legal:

However; all variable names are case-sensitive!

Look at the example below; only the first statement will display the value of the $color variable! This is because $color, $COLOR, and $coLOR are treated as three different variables:

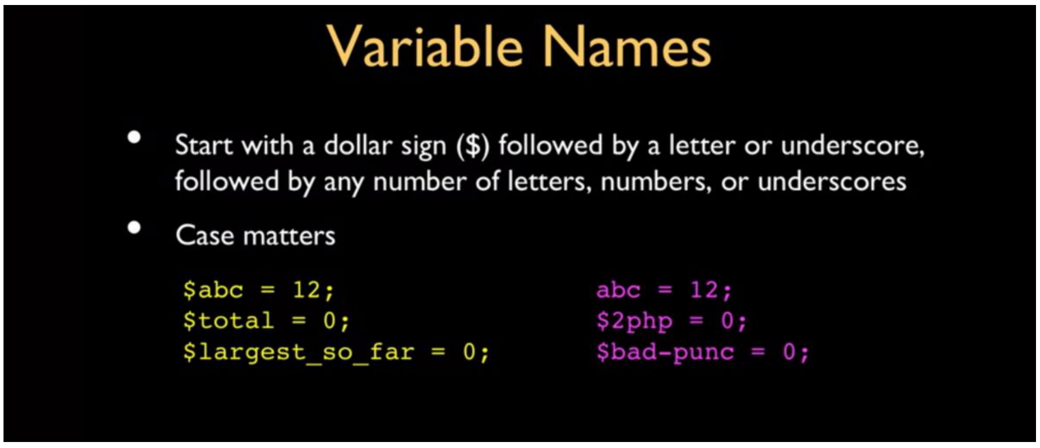
Variables:

Variables are "containers" for storing information.

Strongly/strictly Typed and Loosely Typed Languages

Strongly typed means a language requires explicit declaration of variable types. A loosely typed languages don't require a declaration of variable type, and automatically convert variable type depending upon the context in which they are used and operations you perform to a specific data type as needed.

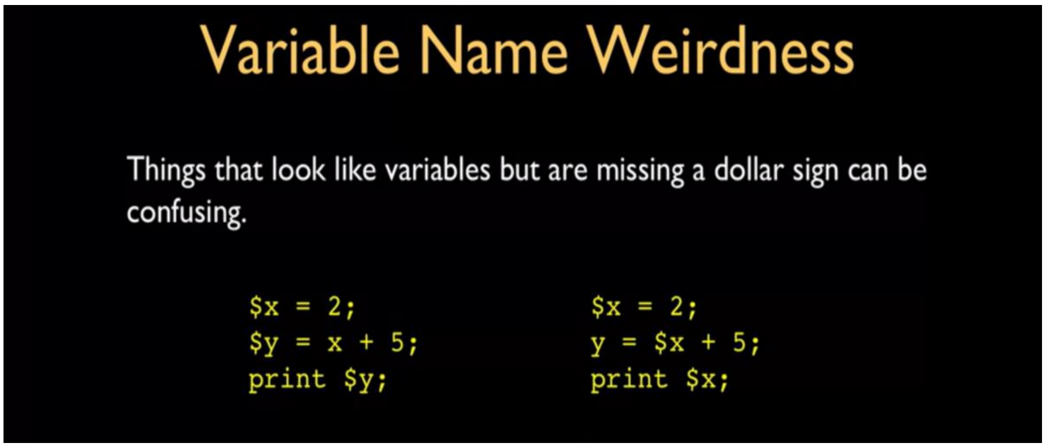
PHP is a loosely typed language.



Variable Names

- Start with a dollar sign ($) followed by a letter or underscore, followed by any number of letters, numbers, or underscores
- Case matters

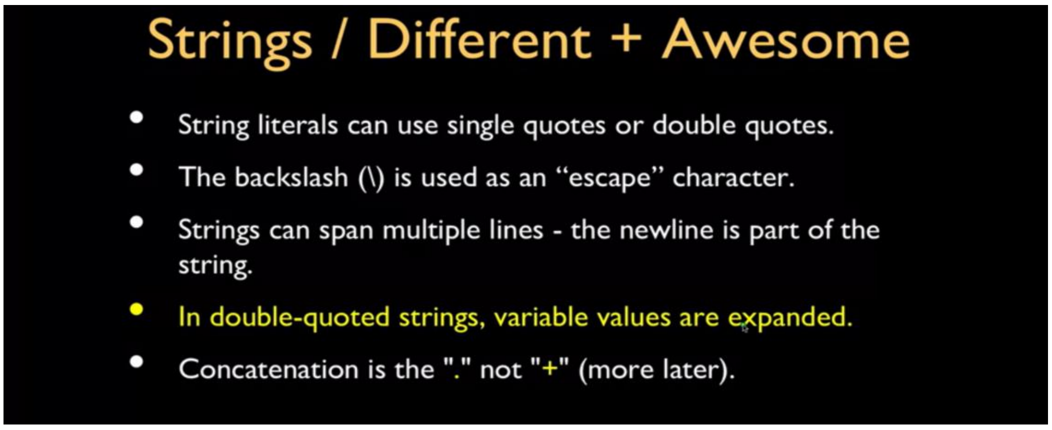
```
$abc = 12;                    abc = 12;
$total = 0;                   $2php = 0;
$largest_so_far = 0;          $bad-punc = 0;
```



Variable Name Weirdness

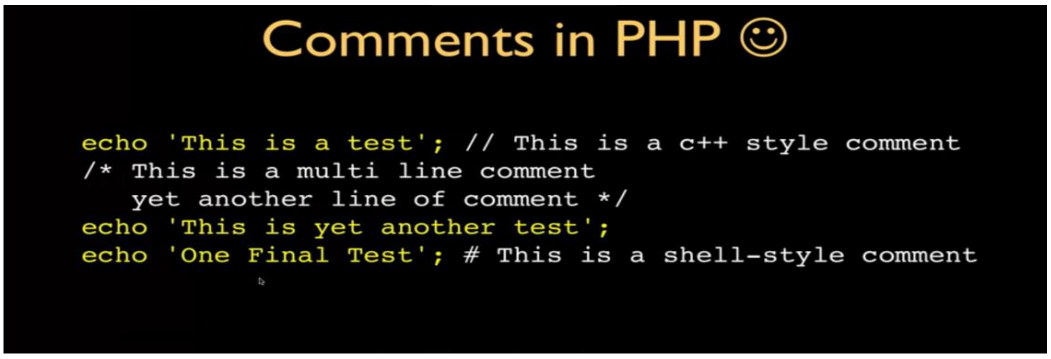
Things that look like variables but are missing a dollar sign can be confusing.

```
$x = 2;               $x = 2;
$y = x + 5;           y = $x + 5;
print $y;             print $x;
```



Strings / Different + Awesome

- String literals can use single quotes or double quotes.
- The backslash (\) is used as an "escape" character.
- Strings can span multiple lines - the newline is part of the string.
- In double-quoted strings, variable values are expanded.
- Concatenation is the "." not "+" (more later).



Comments in PHP ☺

```
echo 'This is a test'; // This is a c++ style comment
/* This is a multi line comment
   yet another line of comment */
echo 'This is yet another test';
echo 'One Final Test'; # This is a shell-style comment
```

PHP echo and print Statements

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

PHP echo statements

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

The PHP echo Statement

The echo statement can be used with or without parentheses: echo or echo().

Display Text

he following example shows how to output text with the echo command (notice that the text can contain HTML markup)

Example

```
<?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.";
?>
```

Display Variables

The following example shows how to output text and variables with the echo statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "BCA 3rd Year";
$x = 5;
$y = 4;

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

The PHP print Statement

The print statement can be used with or without parentheses: print or print().

Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

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

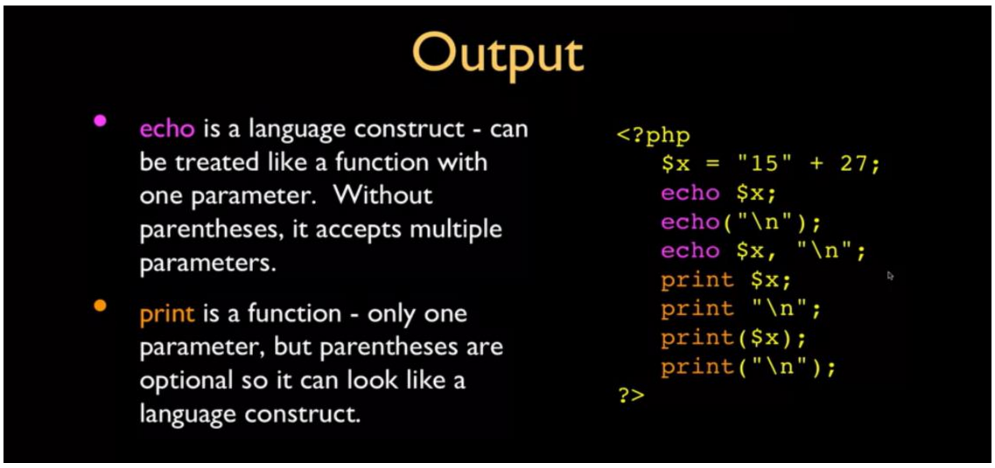
Display Variables

The following example shows how to output text and variables with the print statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "BCA 3rd Year";
$x = 5;
$y = 4;

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



Output

- echo is a language construct - can be treated like a function with one parameter. Without parentheses, it accepts multiple parameters.
- print is a function - only one parameter, but parentheses are optional so it can look like a language construct.

```
<?php
    $x = "15" + 27;
    echo $x;
    echo("\n");
    echo $x, "\n";
    print $x;
    print "\n";
    print($x);
    print("\n");
?>
```

PHP 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

Global Scope

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

Example:

```
<?php
$a = 5; // global scope

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

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

Local Scope

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

Example:

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

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

The global Keyword

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):

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

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

sample(); //function call
echo $y; // outputs 15
?>
```

Static Variables

The final type of variable scoping that I discuss is known as static. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

```
<?php
   function keep_track() {
      STATIC $count = 0;
      $count++;
      print $count;
      print "<br />";
   }
```

```
  keep_track();
  keep_track();
  keep_track();
?>
```

This will produce the following result −

```
1
2
3
```

Datatype

PHP has a total of eight data types which we use to construct our variables −

- **Integers** − are whole numbers, without a decimal point, like 4195.
- **Doubles** − are floating-point numbers, like 3.14159 or 49.1.
- **Booleans** − have only two possible values either true or false.
- **NULL** − is a special type that only has one value: NULL.
- **Strings** − are sequences of characters, like 'PHP supports string operations.'
- **Arrays** − are named and indexed collections of other values.
- **Objects** − are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources** − are special variables that hold references to resources external to PHP (such as database connections).

Integer

They are whole numbers, without a decimal point, like 4195. They are the simplest type. They correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so −

```
$int_var = 12345;
$another_int = -12345 + 12345;
```

the largest integer is 2,147,483,647 and the smallest integer is -2,147,483,647.

Double

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code −

```
<?php
  $many = 2.2888800;
  $many_2 = 2.2111200;
  $few = $many + $many_2;

  print("$many + $many_2 = $few <br>");
?>
```

Boolean

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so −

```
if (TRUE)
  print("This will always print<br>");

else
  print("This will never print<br>");
```

Interpreting other types as Booleans

Here are the rules for determine the "truth" of any value not already of the Boolean type −

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or the string is "0", and is true otherwise.
- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.
- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;
$true_str = "Tried and true"
$true_array[49] = "An array element";
$false_array = array();
$false_null = NULL;
$false_num = 999 - 999;
$false_str = "";
```

NULL

NULL is a special type that only has one value: NULL. A variable of data type NULL is a variable that has no value assigned to it. If a variable is created without a value, it is automatically assigned a value of NULL. To give a variable the NULL value, simply assign it like this −

```
$my_var = NULL;
  or
$my_var = null;
```

A variable that has been assigned NULL has the following properties −

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with IsSet() function.

String

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

```
$string_1 = "This is a string in double quotes";
$string_2 = 'This is a somewhat longer, singly quoted string';
$string_39 = "This string has thirty-nine characters";
$string_0 = ""; // a string with zero characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?php
  $variable = "name";
  $literally = 'My $variable will not print!';

  print($literally);
  print "<br>";
  $literally = "My $variable will print!";
  print($literally);
?>
```

This will produce following result −

```
My $variable will not print!
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP −

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are −

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \$ is replaced by the dollar sign itself ($)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

Array

An array stores multiple values in one single variable.

In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

Example

```
<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>
```

Object

Classes and objects are the two main aspects of object-oriented programming.

A class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Let's assume we have a class named Car. A Car can have properties like model, color, etc. We can define variables like $model, $color, and so on, to hold the values of these properties.

When the individual objects (Volvo, BMW, Toyota, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

If you create a __construct() function, PHP will automatically call this function when you create an object from a class.

Example

```
<?php
class Car {
  public $color;
  public $model;
  public function __construct($color, $model) {
    $this->color = $color;
    $this->model = $model;
  }
  public function message() {
    return "My car is a " . $this->color . " " . $this->model . "!";
  }
}

$myCar = new Car("black", "Volvo");
echo $myCar -> message();
echo "<br>";
$myCar = new Car("red", "Toyota");
echo $myCar -> message();
?>
```

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.

Constant

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default, a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use define() function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a $. You can also use the function constant() to read a constant's value if you wish to obtain the constant's name dynamically.

constant() function

As indicated by the name, this function will return the value of the constant.

This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function.

constant() example

```php
<?php
  define("MINSIZE", 50);

  echo MINSIZE; //50
  echo constant("MINSIZE"); //50 same thing as the previous line
?>
```

Only scalar data (boolean, integer, float and string) can be contained in constants.

Differences between constants and variables are

- There is no need to write a dollar sign ($) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the define() function.
- Constants may be defined and accessed anywhere without regard to variable scoping rules.

- Once the Constants have been set, may not be redefined or undefined.

Valid and invalid constant names

```
// Valid constant names
define("ONE",     "first thing");
define("TWO2",    "second thing");
define("THREE_3", "third thing");
define("__THREE__", "third value");

// Invalid constant names
define("2TWO",    "second thing");
```

PHP String

A string is a sequence of characters, like "Hello world!".

String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator −

```
<?php
   $string1="Hello World";
   $string2="1234";

   echo $string1 . " " . $string2;
?>
```

This will produce the following result −

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

PHP String Functions

strlen() - Return the Length of a String

The PHP strlen() function returns the length of a string.

Example

Return the length of the string "Hello world!":

```php
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

str_word_count() - Count Words in a String

The PHP str_word_count() function counts the number of words in a string.

Example

Count the number of word in the string "Hello world!":

```php
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

strrev() - Reverse a String

The PHP strrev() function reverses a string.

Example

Reverse the string "Hello world!":

```php
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

strpos() - Search For a Text Within a String

The PHP strpos() function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.

Example

Search for the text "world" in the string "Hello world!":

```php
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

str_replace() - Replace Text Within a String

The PHP str_replace() function replaces some characters with some other characters in a string.

Example

Replace the text "world" with "Dolly":

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!
?>
```

strcasecmp()

Compare two strings (case-insensitive):

```
<?php
echo strcasecmp("Hello world!","HELLO WORLD!");// 0
?>
```

strcmp()

Compare two strings (case-sensitive):

```
<?php
echo strcmp("Hello World!","Hello world!"); // -1
?>
```

strtolower()

Convert all characters to lowercase:

```
<?php
echo strtolower("Hello WORLD.");
?>
```

strtoupper()

Convert all characters to uppercase:

```
<?php
echo strtoupper("Hello WORLD!");
?>
```

rtrim()

Remove characters from the right side of a string:

```php
<?php
$str = "Hello World!";
echo $str . "<br>";
echo rtrim($str,"World!"); // Hello
?>
```

ltrim()

Remove characters from the left side of a string:

```php
<?php
$str = "Hello World!";
echo $str . "<br>";
echo ltrim($str,"Hello"); // World
?>
```

trim()

Remove characters from the both left and right side of a string:

```php
<?php
$str = "Hello World!";
echo $str . "<br>";
echo trim($str,"Hed!"); //ello Worl
?>
```

strtr()

Replace the characters "ia" in the string with "eo":

```php
<?php
echo strtr("Hilla Warld","ia","eo");
?>
```

str_split()

Split the string "Hello" into an array:

```php
<?php
print_r(str_split("Hello"));
?>
```

join()

Join array elements with a string:

```
<?php
$arr = array('Hello','World!','Beautiful','Day!');
echo join(" ",$arr);
?>
```

substr()

Return "world" from the string:

```
<?php
echo substr("Hello world",6);
?>
```

PHP Operators

Operators are used to perform operations on variables and values. PHP language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

+, - , *, /, %, ** (exponentiation), ++, --

Comparison Operators

== 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]

>, <, >=, <=

Logical Operators

and [Called Logical AND operator. If both the operands are true then condition becomes true.]

or [Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.]

&& [Called Logical AND operator. If both the operands are non zero then condition becomes true.]

|| [Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.]

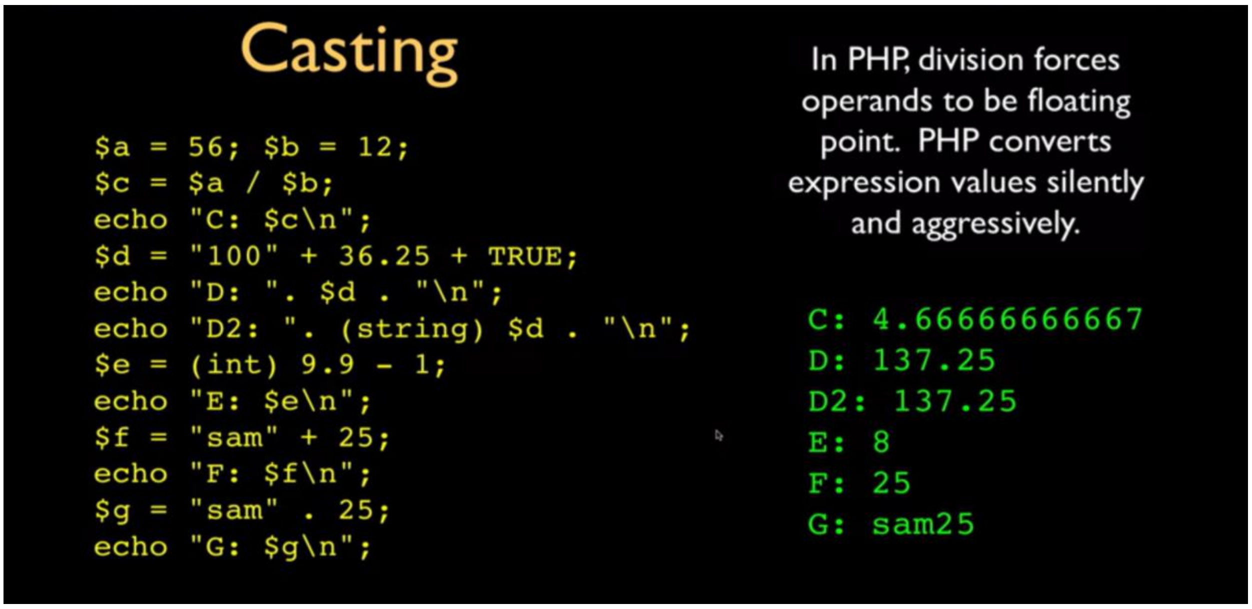
! [Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.]

Assignment Opertors

=, +=, -=, *=, /=, %=

Conditional Operator

? : [If Condition is true ? Then value X : Otherwise value Y]

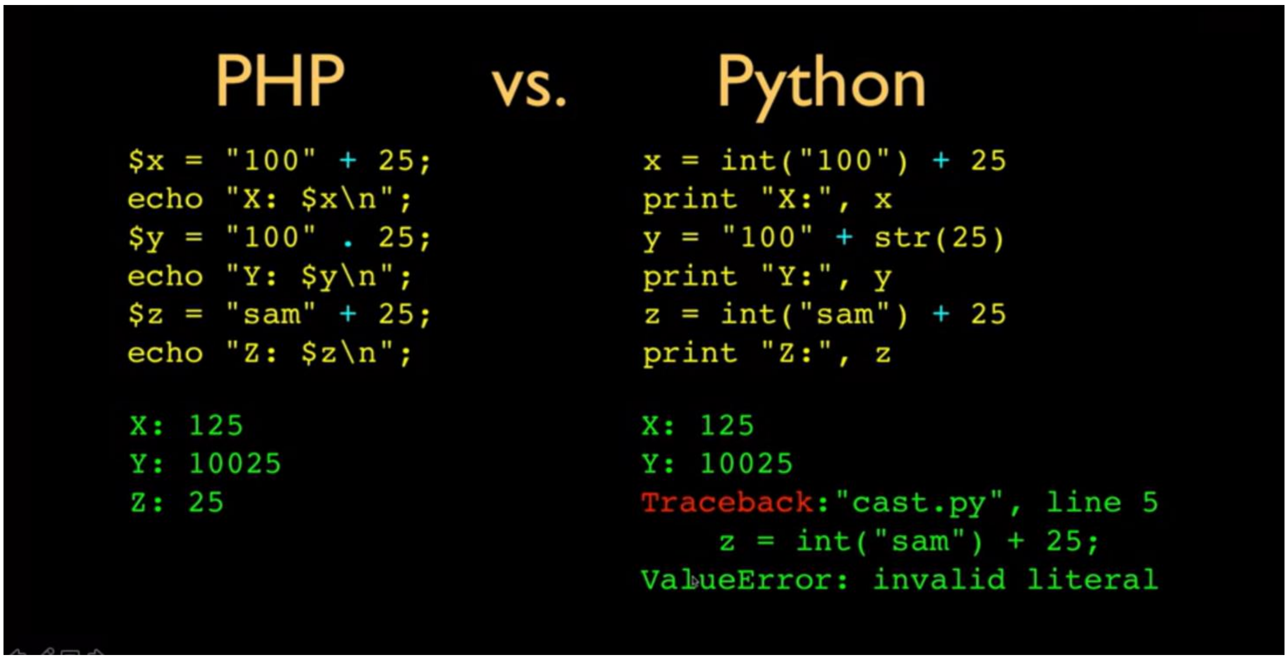


Casting

```
$a = 56; $b = 12;
$c = $a / $b;
echo "C: $c\n";
$d = "100" + 36.25 + TRUE;
echo "D: ". $d . "\n";
echo "D2: ". (string) $d . "\n";
$e = (int) 9.9 - 1;
echo "E: $e\n";
$f = "sam" + 25;
echo "F: $f\n";
$g = "sam" . 25;
echo "G: $g\n";
```

In PHP, division forces operands to be floating point. PHP converts expression values silently and aggressively.

```
C: 4.66666666667
D: 137.25
D2: 137.25
E: 8
F: 25
G: sam25
```



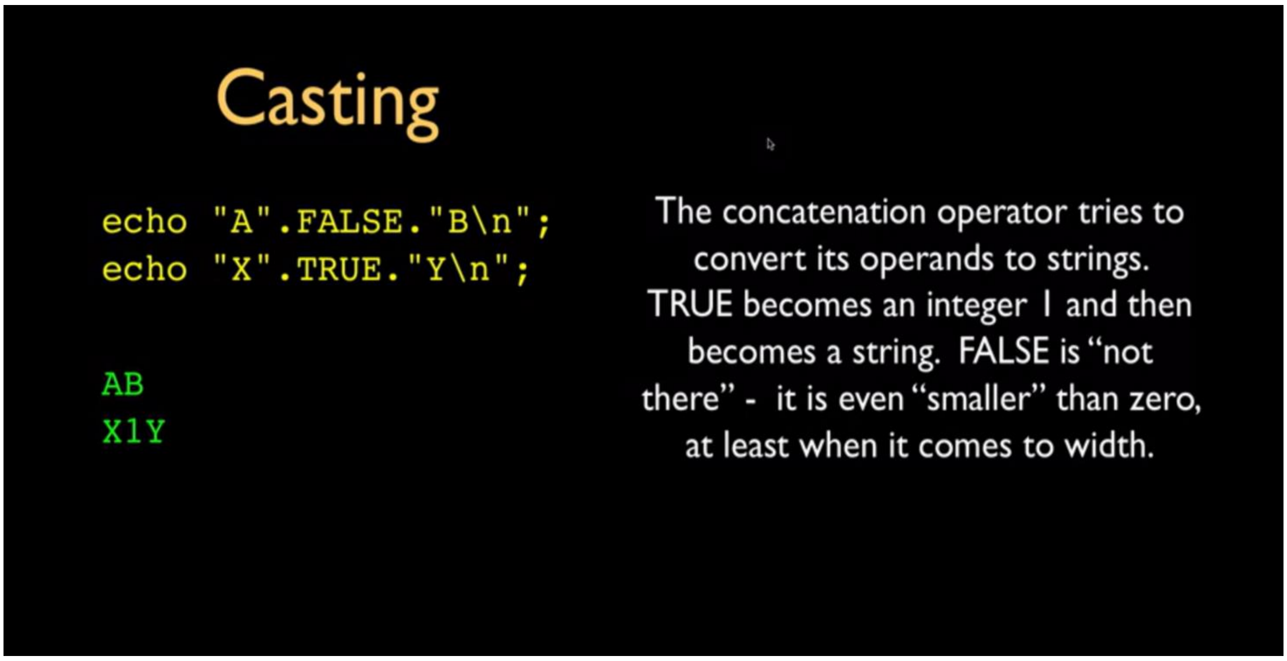
PHP vs. Python

```
$x = "100" + 25;
echo "X: $x\n";
$y = "100" . 25;
echo "Y: $y\n";
$z = "sam" + 25;
echo "Z: $z\n";

X: 125
Y: 10025
Z: 25
```

```
x = int("100") + 25
print "X:", x
y = "100" + str(25)
print "Y:", y
z = int("sam") + 25
print "Z:", z

X: 125
Y: 10025
Traceback:"cast.py", line 5
    z = int("sam") + 25;
ValueError: invalid literal
```

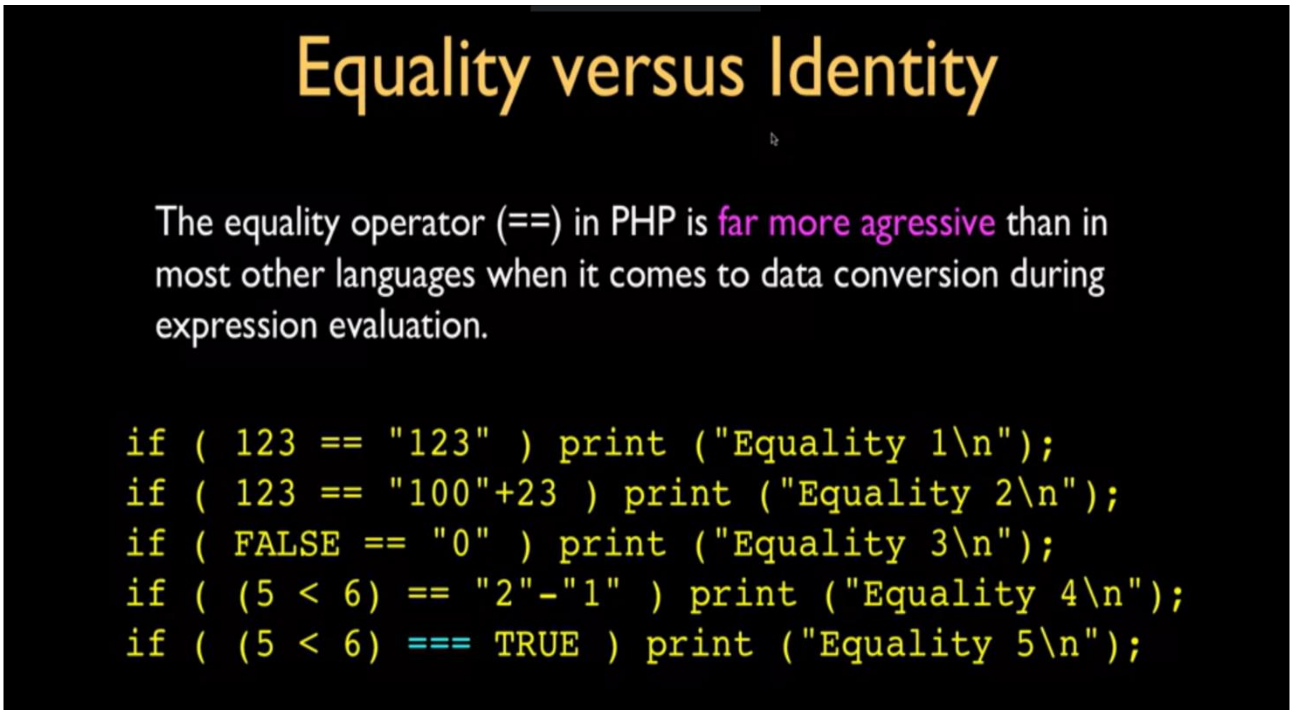


Casting

```
echo "A".FALSE."B\n";
echo "X".TRUE."Y\n";

AB
X1Y
```

The concatenation operator tries to convert its operands to strings. TRUE becomes an integer 1 and then becomes a string. FALSE is "not there" - it is even "smaller" than zero, at least when it comes to width.



Equality versus Identity

The equality operator (==) in PHP is far more agressive than in most other languages when it comes to data conversion during expression evaluation.

```
if ( 123 == "123" ) print ("Equality 1\n");
if ( 123 == "100"+23 ) print ("Equality 2\n");
if ( FALSE == "0" ) print ("Equality 3\n");
if ( (5 < 6) == "2"-"1" ) print ("Equality 4\n");
if ( (5 < 6) === TRUE ) print ("Equality 5\n");
```