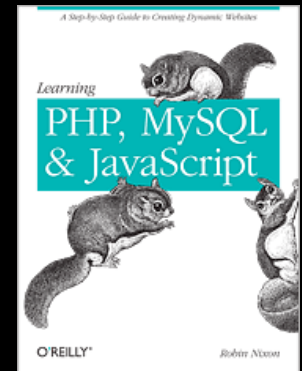


Expressions and Control Flow in PHP

Chapter 4
Dr. Charles Severance

To be used in association with the book:
PHP, MySQL, and JavaScript by Robin Nixon



open.michigan

Unless otherwise noted, the content of this course material is licensed under a Creative Commons Attribution 3.0 License.

<http://creativecommons.org/licenses/by/3.0/>.

Copyright 2011, Charles Severance



Expressions

- Expressions evaluate to a value. The value can be a string, number, boolean, etc...
- Expressions often use operations and function calls, and there is an order of evaluation when there is more than one operator in an expression
- Expressions can also produce objects like arrays

Operator Precedence

High



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



Low

Operators of Note

- Increment / Decrement (++ --)
- String concatenation (.)
- Equality (== !=)
- Identity (=== !==)
- Ternary (? :)
- Side-effect Assignment (+= -= .= etc.)
- Ignore the rarely-used bitwise operators (>> << ^ | &)

Increment / Decrement

- These operators allow you to both retrieve and increment / decrement a variable
- They are generally avoided in civilized code.

```
$x = 12;
```

```
$y = 15 + $x++;
```

```
echo "x is $x and y is $y \n";
```

```
x is 13 and y is 27
```

String Concatenation

- PHP uses the period character for concatenation because the plus character would instruct PHP to to the best it could do to add the two things together, converting if necessary.

```
$a = 'Hello ' . 'World!';  
echo $a . "\n";
```

```
Hello World!
```

Equality versus Identity

- The equality operator (==) in PHP is **far more aggressive** than in most other languages. It will convert data

```
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");
```

[-] Description

[Report a bug](#)

```
int strpos ( string $haystack , mixed $needle [, int $offset = 0 ] )
```

Returns the numeric position of the first occurrence of *needle* in the *haystack* string.

[+] Parameters

[-] Return Values

[Report a bug](#)

Returns the position as an integer. If *needle* is not found, **strpos()** will return [boolean](#) FALSE.



Warning

This function may return Boolean FALSE, but may also return a non-Boolean value which evaluates to FALSE, such as 0 or "". Please read the section on [Booleans](#) for more information. Use [the === operator](#) for testing the return value of this function.

<http://php.net/manual/en/function.strpos.php>

```
$vv = "Hello World!";  
echo "First:" . strpos($vv, "Wo") . "\n";  
echo "Second: " . strpos($vv, "He") . "\n";  
echo "Third: " . strpos($vv, "ZZ") . "\n";  
if (strpos($vv, "He") == FALSE ) echo "Wrong A\n";  
if (strpos($vv, "ZZ") == FALSE ) echo "Right B\n";  
if (strpos($vv, "He") === FALSE ) echo "Right C\n";  
if (strpos($vv, "ZZ") === FALSE ) echo "Right D\n";  
print_r(FALSE); print FALSE;  
echo "Where were they?\n";
```

```
First:6  
Second: 0  
Third:  
Wrong A  
Right B  
Right D  
Where were they?
```

Beware FALSE variables. They are detectable but not visible...

Ternary

- The ternary operator comes from C. It allows conditional expressions. It is like a one-line if-then-else . Like all "contraction" syntaxes, we use it carefully.

```
$www = 123;  
$msg = $www > 100 ? "Large" : "Small" ;  
echo "First: $msg \n";  
$msg = ( $www % 2 == 0 ) ? "Even" : "Odd";  
echo "Second: $msg \n";  
$msg = ( $www % 2 ) ? "Odd" : "Even";  
echo "Third: $msg \n";
```

First: Large
Second: Odd
Third: Odd

Side-Effect Assignment

- These are pure contractions. Civilized programmers use them sparingly.

```
echo "\n";  
$out = "Hello";  
$out .= " ";  
$out .= "World!";  
$out .= "\n";  
echo $out;  
$count = 0;  
$count += 1;  
echo "Count: $count\n";
```

```
Hello World!  
Count: 1
```

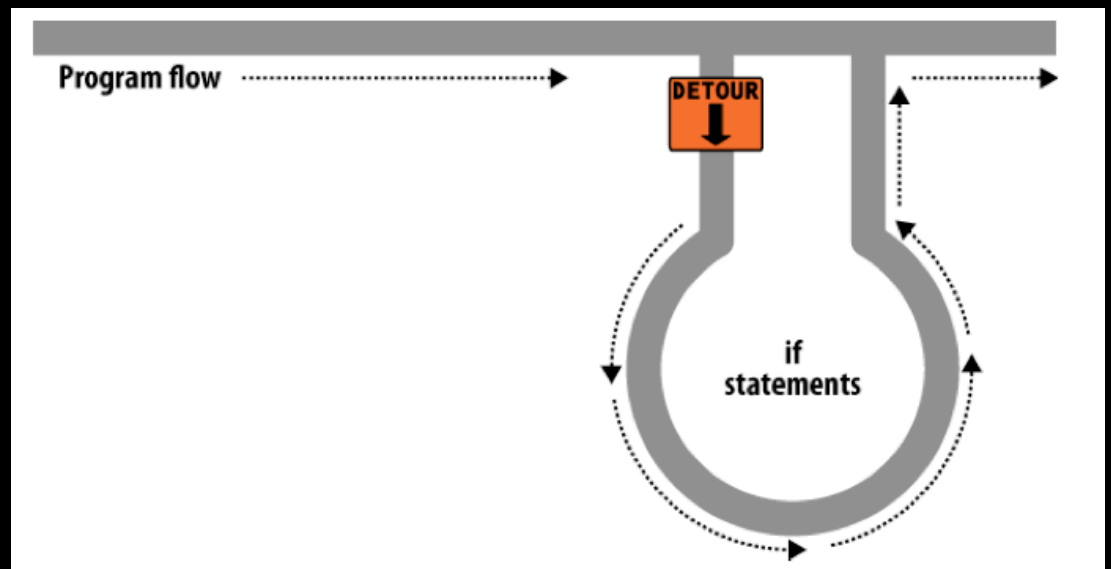
Side-Effect Assignment

- These are pure contractions. Civilized programmers use them sparingly.

```
echo "\n";  
$out = "Hello";  
$out .= " ";  
$out .= "World!";  
$out .= "\n";  
echo $out;  
$count = 0;  
$count += 1;  
echo "Count: $count\n";
```

```
Hello World!  
Count: 1
```

Control Structures

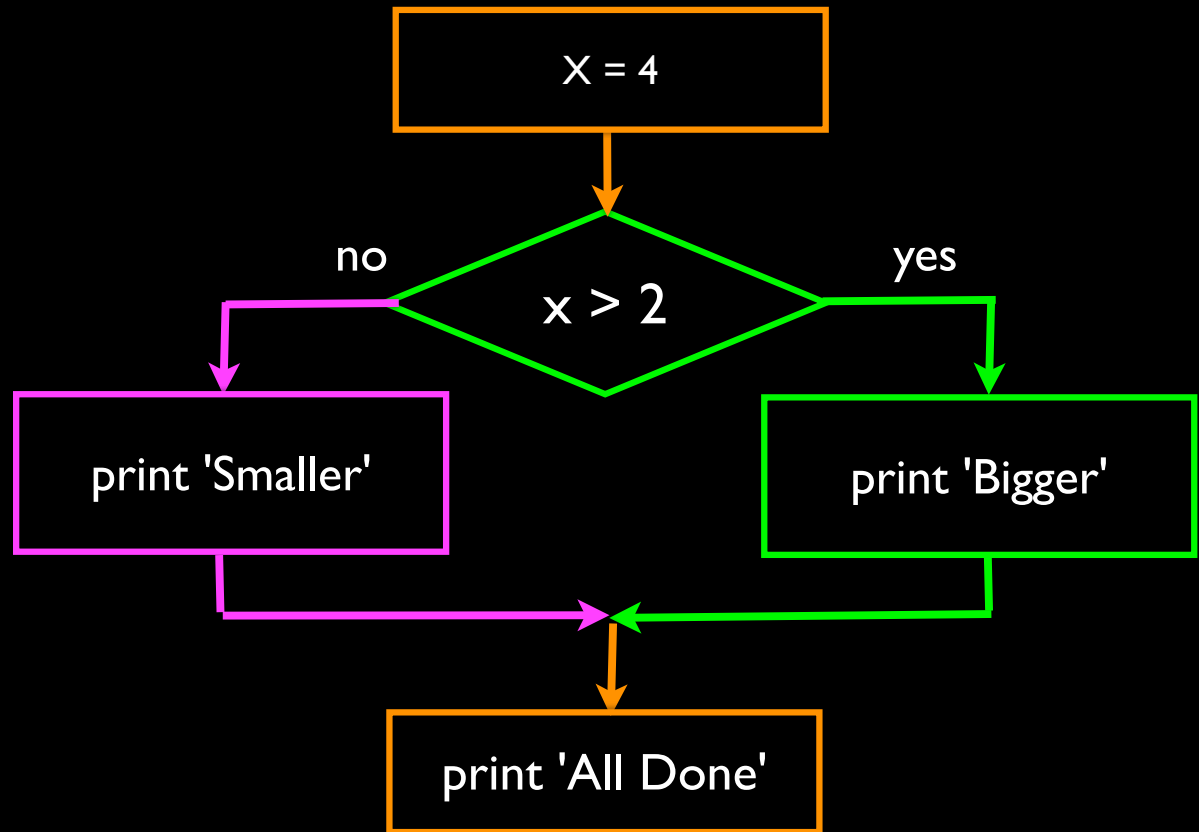


Two-way using else :

```
x = 4
```

```
if x > 2 :  
    print 'Bigger'  
else :  
    print 'Smaller'
```

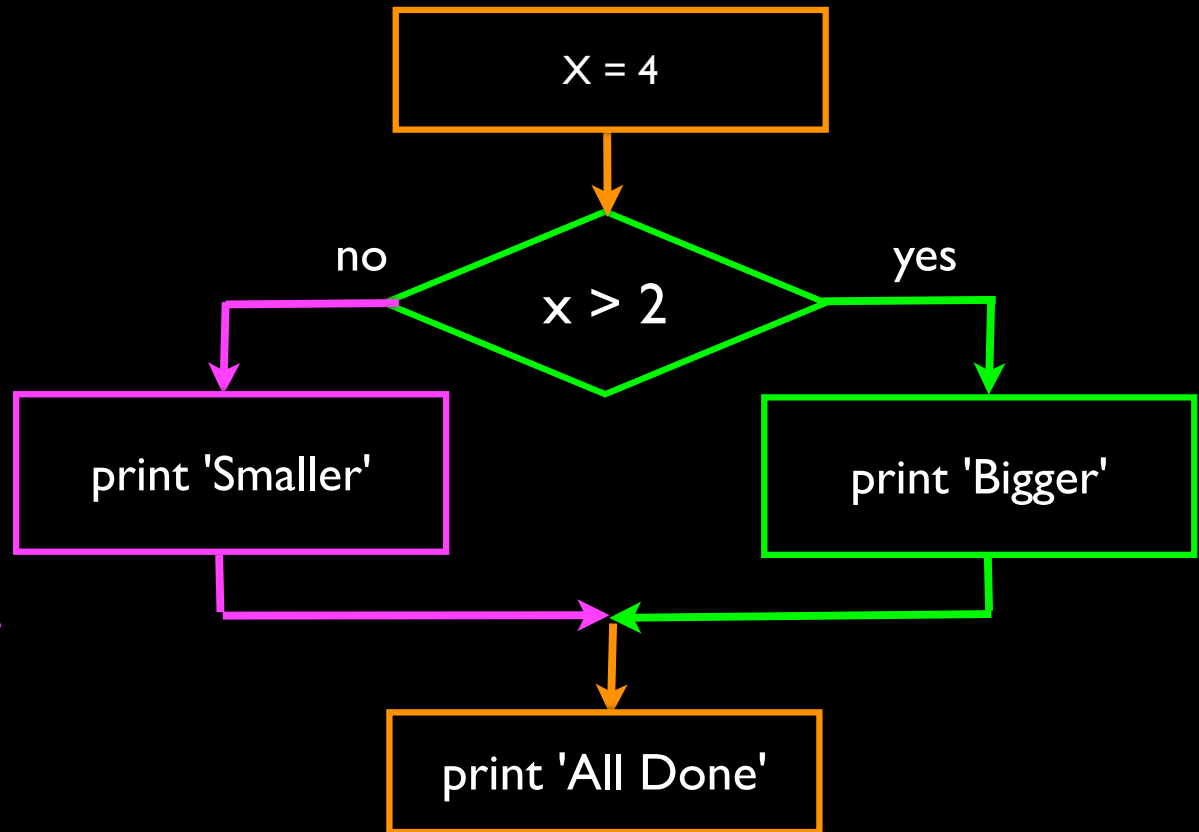
```
print 'All done'
```



Two-way using else :

```
$x = 4;
```

```
if ($x > 2) {  
    print "Bigger\n";  
} else {  
    print "Smaller\n";  
}  
print "All done\n";
```



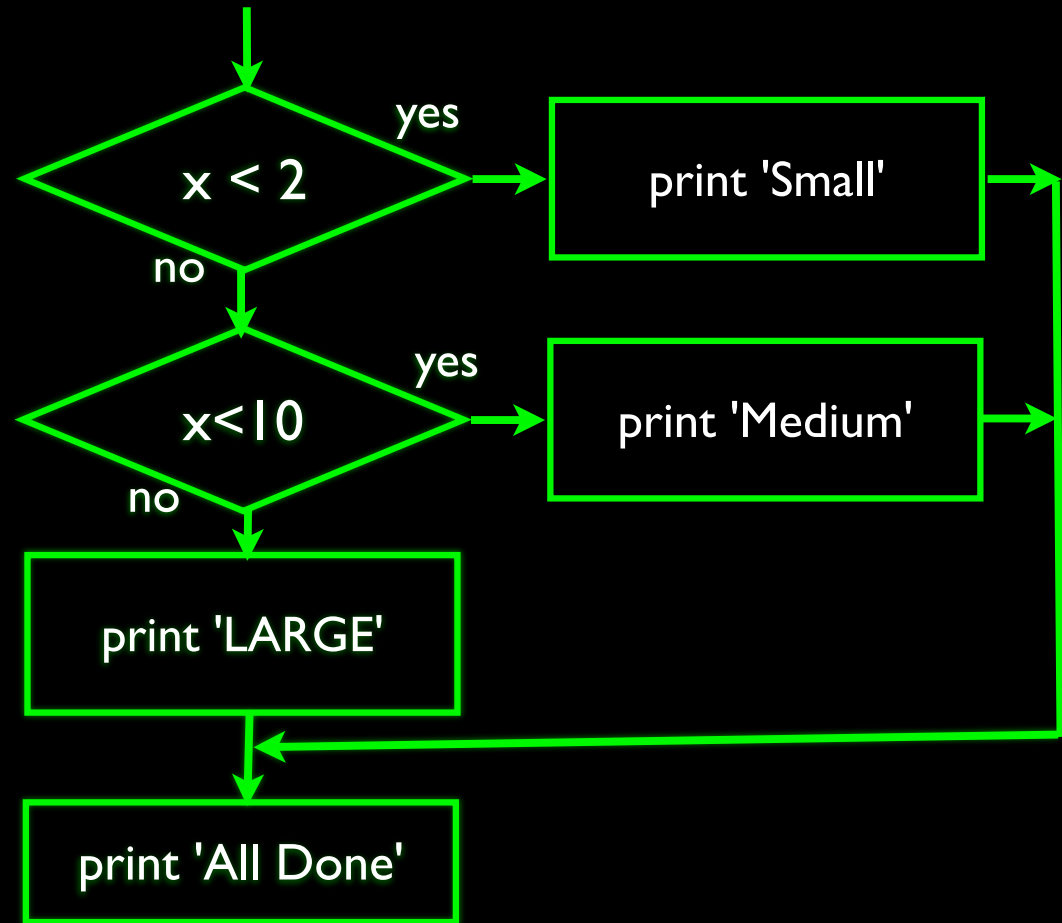
What is wrong?

Smaller II
All done

```
$x = 4;  
  
if (x > 2) {  
    print "Bigger II\n";  
} else {  
    print "Smaller II\n";  
}  
print "All done\n";
```

Multi-way

```
$x = 7;  
  
if ( $x < 2 ) {  
    print "Small\n";  
} elseif ( $x < 10 ) {  
    print "Medium\n";  
} else {  
    print "LARGE\n";  
}  
  
print "All done\n";
```



Curly Braces are not Required

```
if      ($page == "Home")   echo "You selected Home";  
elseif ($page == "About")  echo "You selected About";  
elseif ($page == "News")   echo "You selected News";  
elseif ($page == "Login")  echo "You selected Login";  
elseif ($page == "Links")  echo "You selected Links";
```

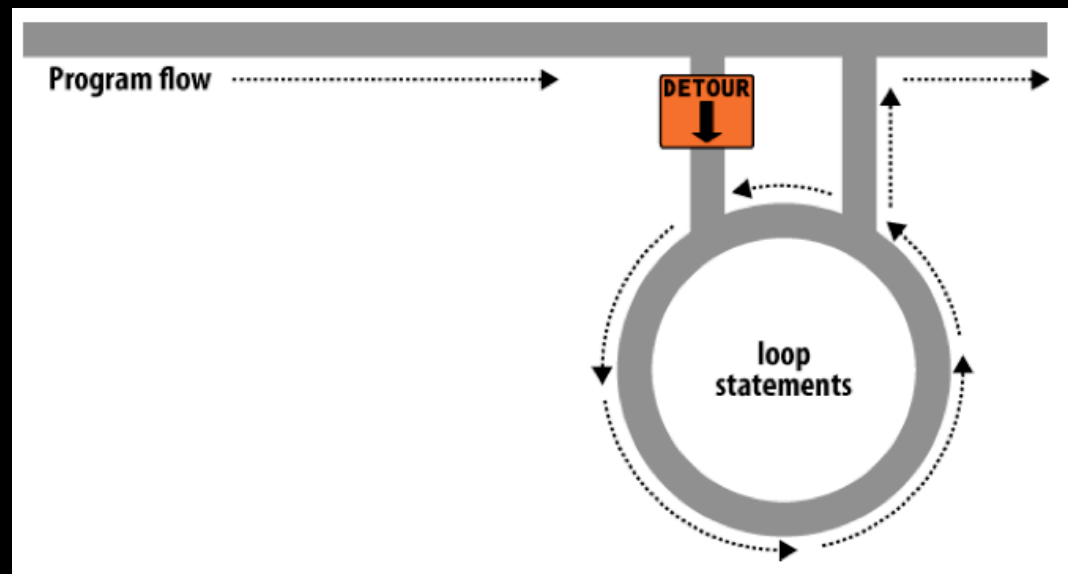
```
if      ($page == "Home")   { echo "You selected Home"; }  
elseif ($page == "About")  { echo "You selected About"; }  
elseif ($page == "News")   { echo "You selected News"; }  
elseif ($page == "Login")  { echo "You selected Login"; }  
elseif ($page == "Links")  { echo "You selected Links"; }
```

```
switch ($page)
{
    case "Home":
        echo "You selected Home";
        break;
    case "About":
        echo "You selected About";
        break;
    case "News": echo "You selected News";
        break;
    case "Login": echo "You selected Login";
        break;
    case "Links": echo "You selected Links";
        break;
}
```

```
$page = "x";  
switch($page) {  
    case "w":  
        echo "A\n";  
        break;  
    case "x":  
        echo "X\n";  
    case "y":  
        echo "Y\n";  
        break;  
    case "z":  
        echo "Z\n";  
        break;  
    case "x":  
        echo "XX\n";  
        break;  
}
```

X
Y

Looping Structures



```
$fuel = 10;
while ($fuel > 1) {
    print "Vroom vroom\n";
}
```

A **while** loop is a "zero-trip" loop with the test at the top. before the first iteration starts. We hand construct the **iteration variable** to implement a counted loop.

```
$fuel = 10;
while ($fuel > 1) {
    print "Vroom vroom\n";
    $fuel = $fuel - 1;
}
```

```
$count = 1;
do {
    echo "$count times 5 is " . $count * 5;
    echo "\n";
} while (++$count <= 5);
```

A **do-while** loop is a "one-trip" loop with the test at the bottom after the first iteration completes.

```
1 times 5 is 5
2 times 5 is 10
3 times 5 is 15
4 times 5 is 20
5 times 5 is 25
```

```
for($count=1; $count<=6; $count++ ) {  
    echo "$count times 6 is " . $count * 6;  
    echo "\n";  
}
```

A **for** loop is the simplest way
to construct a counted loop.

```
1 times 6 is 6  
2 times 6 is 12  
3 times 6 is 18  
4 times 6 is 24  
5 times 6 is 30  
6 times 6 is 36
```

Loop runs while TRUE (top-test)

Before loop starts

Run after each iteration.

```
for($count=1; $count<=6; $count++ ) {  
    echo "$count times 6 is " . $count * 6;  
    echo "\n";  
}
```

A **for** loop is the simplest way to construct a counted loop.

```
1 times 6 is 6  
2 times 6 is 12  
3 times 6 is 18  
4 times 6 is 24  
5 times 6 is 30  
6 times 6 is 36
```

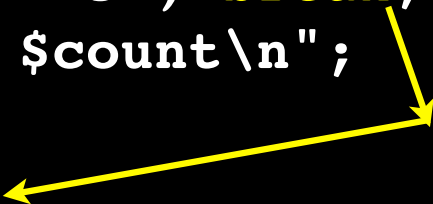
Loop Controls

- Like many C-inspired languages, PHP has two control structures that work within a loop
 - **break** - exit the loop immediately
 - **continue** - finish the current iteration and jump to the next iteration, starting at the top of the loop

Breaking Out of a Loop

- The **break** statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

```
for($count=1; $count<=600; $count++ ) {  
    if ( $count == 5 ) break;  
    echo "Count: $count\n";  
}  
echo "Done\n";
```

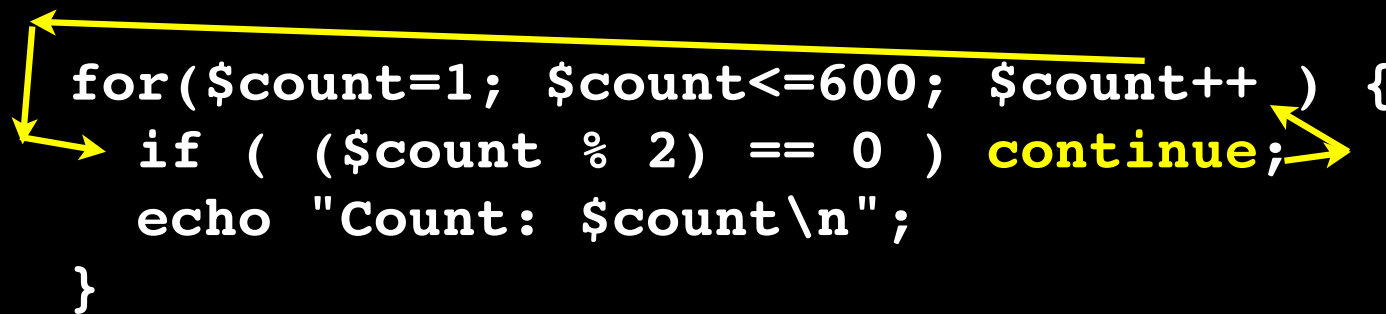


```
Count: 1  
Count: 2  
Count: 3  
Count: 4  
Done
```

Finishing an Iteration with continue

- The **continue** statement ends the current iteration and jumps to the top of the loop and starts the next iteration

```
for($count=1; $count<=600; $count++) {  
  if ( ($count % 2) == 0 ) continue;  
  echo "Count: $count\n";  
}  
echo "Done\n";
```



Count: 1
Count: 3
Count: 5
Count: 7
Count: 9
Done

Conversion / Casting

- As PHP evaluates expressions, at times values in the expression need to be converted from one type to another as the computations are done.
- PHP does aggressive implicit type conversion (casting)
- You can also make type conversion (casting) explicit with casting operators.

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: 4.6666666666667  
D: 137.25  
D2: 137.25  
E: 8  
F: 25  
G: sam25
```

Explicit Casting

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

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.

Summary

- Expressions
- Operators
- Conditional Structures
- Looping Structures
- Type Conversion and Casting