Week 4 • Chapters 4–5

CIS 86 • PHP and MySQL • Mission College

Getting Started Tonight

- Download this PPT file from Angel to your computer
  - CIS_086_Week_4_Conditionals_Loops.pdf

- Start WAMP/MAMP on your computer

Summary

- Relational and Logical operators
- Truth tables
- Conditional statements
  - Nested If statements
  - Switch statements
- Ternary operator
- Loops
  - Break statements
  - Infinite loops
- Common mistakes with functions, loops, and conditions
- Functions? (chapter 5)

Relational and Logical Operators

- Relational operators take two quantifiable values as input.
  - Numbers
  - Strings
- Logical operators take two Boolean values as input.
  - True
  - False
- Both types of operators return a Boolean value.
  - True
  - False

Relational Operators

Operator	What it does
==	Equal value
===	Equal value and type
!=	Not equal value
!==	Not equal value or not the same type
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Logical Operators

Operator	Name	What it does
&&	AND	True if both operands are true
||	OR	True if either operand is true
XOR	XOR	True if one and only one of the operands is true
!	NOT	True if its operand is false

Truth Table(s)

a	b	a && b	a || b	a XOR b
T	T	T	T	F
T	F	F	T	T
F	T	F	T	T
F	F	F	F	F

Conditionals

```
if (a) {
  do_stuff_b();
}
else if (c) {
  do_stuff_d();
}
```

- `do_stuff_b()` happens only if `a` is true.
- `do_stuff_d()` happens only if `a` is false and `c` is true.

Nested IF statements

- Best practice is to always use braces explicitly, otherwise the statements may not nest the way you intended.

```
if ($color == "blue")
  if ($shape == "rect")
    drawBlueRect();
  else if ($shape == "circle")
    drawBlueCircle();

else if ($color == "red")
  if ($shape == "rect")
    drawRedRect();
  else if ($shape == "circle")
    drawRedCircle();
```

```
if ($color == "blue") {
  if ($shape == "rect") {
    drawBlueRect();
  }
  else if ($shape == "circle") {
    drawBlueCircle();
  }
}
else if ($color == "red") {
  if ($shape == "rect") {
    drawRedRect();
  }
  else if ($shape == "circle") {
    drawRedCircle();
  }
}
```

SWITCH statements

- Use switch instead of a long chain of if…else statements.
- Each possible value goes in a CASE label.
- Unlike some other languages, the conditional can be a string.
- Always include the BREAK statement.
- Always include the DEFAULT label.
- Switch statements can sometimes be replaced with a function that accesses an associative array.
- Most useful when the different cases get very different processing.

SWITCH statements

```
switch ($colorName)
{
  case "red"     : $colorHex = "ff0000"; break;
  case "orange"  : $colorHex = "ff8000"; break;
  case "yellow"  : $colorHex = "ffff00"; break;
  case "green"   : $colorHex = "00ff00"; break;
  case "cyan"    : $colorHex = "00ffff"; break;
  case "blue"    : $colorHex = "0000ff"; break;
  case "magenta" : $colorHex = "ff00ff"; break;
  case "violet"  : $colorHex = "8000ff"; break;
  default        : $colorHex = "000000"; break;
}
```

(Equivalent associative array)

```
$colorValues = [
  “red”  => “ff0000”,
  “orange” => “ff8000”,
  “yellow” => “ffff00”,
  “green” => “00ff00”,
  “cyan” => “00ffff”,
  “blue” => “0000ff”,
  “magenta” => “ff00ff” ];

$colorHex = $colorValues[$colorName];
```

Loops

- **for**
  - When you know how many times you want to go through the loop.
  - Example: printing 12 icons onto the web page.
- **while**
  - When you don't know how many times you'll go through the loop, but you have a way to know when you're finished.
  - For example: reading from a file. You don't know how many lines the file has, but you can detect an EOF marker.
  - Not guaranteed to ever go through the loop.
- **do…while**
  - Same as while, but guaranteed to go through the loop at least once.

Loop equivalence

```
for ($i=0; $i<12; $i++) {
    echo $i . “<br />\n”;
}
```

```
$i = 0;
do {
    echo $i . “<br />\n”;
    $i++;
}
  while ($i<12);
```

```
$i = 0;
while ($i<12) {
    echo $i . “<br />\n”;
    $i++;
}
```

Break: Used to exit a loop early

```
// Example without Break
// always runs N times if there are N array elements
for ($i=0; $i<$n; $i++) {
  if ($myArray[$i] == $testValue)
    $indexValue = $i;
}


// Example with Break
// stops when it finds the matching array element
for ($i=0; $i<$n; $i++) {
  if ($myArray[$i] == $testValue) {
    $indexValue = $i;
    break;
  }
}
```

Infinite Loop

- Forgetting to change the counter variable

```
$counter = 0;
while ($counter < 100)
{
  echo $myArray[$counter];
  $counter++;
}
```

- Changing the counter variable the wrong way

```
// count down 100 to 0

$counter = 100;
while ($counter >= 0)
{
  echo $myArray[$counter];
  // $counter++; // WRONG
  $counter--;
}
```

Infinite Loop

- Checking the counter variable incorrectly

```
// count by threes

$counter = 0;
while ($counter != 100)
{
  echo $myArray[$counter];
  $counter+=3;
}
```

```
// count by threes

$counter = 0;
while ($counter < 100)
{
  echo $myArray[$counter];
  $counter+=3;
}
```

Common mistakes

- Missing or misplaced semicolon (;)
- Blocks nested incorrectly
- Forgot BREAK in CASE clause of SWITCH statement

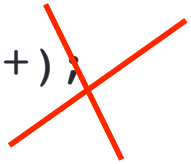
- Missing $ in variable name
  - `while (i < 100) // this will not work`
  - `while ($i < 100) // this will work`

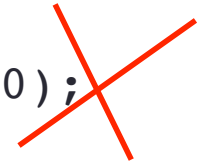
Semicolons: Functions

```
// don't put a semicolon at the
// beginning of a function definition.
// this happens when you copy and paste

function getColorHex ($colorName);
{
  // does some stuff here
}

// but function invocation does need a semicolon
$colorHex = getColorHex ("red");
```

Semicolons: Loops

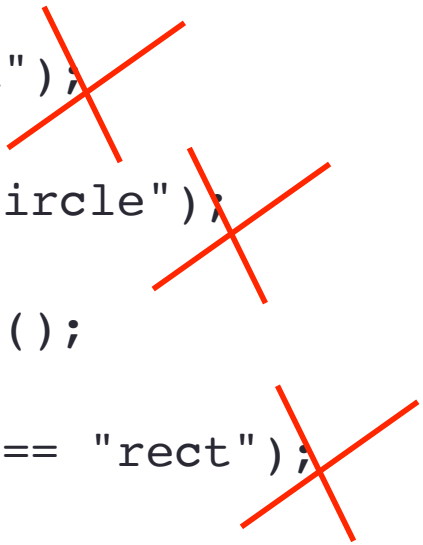
```
// don't put a semicolon at the beginning of a loop

for ($i=0; $i<100; $i++);
{
  echo $colors[$i];
}

$i = 0;
while ($i < 100);
{
  echo $colors[$i];
  $i++;
}
```





Semicolons: If … else

```
// don't put a semicolon at the end
// of an if or else clause

if ($color == "red");
{
  if ($shape == "circle");
  {
    drawRedCircle ();
  }
  else if ($shape == "rect");
  {
    drawRedRect ();
  }
}
```



More common problems

- http://www.tuxradar.com/practicalphp/19/12/3
- http://code.tutsplus.com/articles/are-you-making-these-10-php-mistakes--net-2894
- http://www.primitivetype.com/articles/common_php_errors.php

Chapter 5: Functions and more

- Using functions
- Writing your own functions
- Returning values
- Returning an array
- Passing by reference
- Accessing global variables
- Variable scope
- Including and requiring files

Pass by value vs. reference

- When a variable is passed by value, the function makes a local copy of the variable and uses the local copy for all calculations. When the function returns, the local copy is thrown away, and any changes made to it are thrown away too.

- When a variable is passed by reference, the called function gets the actual variable to manipulate. This means it can both use and change the value of that variable. If the function changes the variable, after the variable is returned, the calling function will see the changed value.

Pass by value vs. reference

```
// Returning 2 values

function getColorAndShape (&$color, &$shape)
{
  $color = "red";
  $shape = "circle";
  return true;
}

$color = $shape = "";
getColorAndShape ($color, $shape);
```

Pass by value vs. reference

```
// Example: Swap 2 values
// If passed by value, this will not work.

function swap (&$value1, &$value2)
{
  $temp = $value1;
  $value1 = $value2;
  $value2 = $temp;
}

if ($colors[$i] > $colors[$i+1])
  swap ($colors[$i], $colors[$i+1]);
```

Pass by value vs. reference

```
// Example: changing a value
// If passed by value, this will not work.

function makeLowerCase (&$inputString)
{
  $inputString = strtolower($inputString);
}

$color = "ORanGe";
makeLowerCase ($color);
```