codecademy

Conditionals and Logic in PHP

PHP else statement

A PHP else statement can follow an `if` block. If the condition of the `if` does not evaluate to `TRUE`, the code block following `else` will be executed.

```
$condition = FALSE;
if ($condition) {
  // This code block will not execute
} else {
  // This code block will execute
}
```

PHP comparison operators

PHP *comparison operators* are used to compare two values and return `TRUE` or `FALSE` depending on the validity of the comparison. Comparison operators include:

- identical (===)
- not identical (!==)
- greater than (>)
- less than (<)
- greater than or equal (>=)
- less than or equal (<=)

```
// Comparison operators
1 > 3; // FALSE
3 > 1; // TRUE
250 >= 250; // TRUE
1 === 1; // TRUE
1 === 2; // FALSE
1 === "1"; // FALSE
```

PHP If Statements

PHP `if` statements evaluate a boolean value or expression and execute the provided code block if the expression evaluates to `TRUE`.

```
if (TRUE){
  echo "TRUE is always true";
}

$condition1 = TRUE;
if ($condition1) {
  // This code block will execute
}

$condition2 = FALSE;
if ($condition2) {
  // This code block will not execute
}
```

PHP elseif statements

PHP `elseif` statements must be paired with an `if` statement, but many `elseif` s can be chained from a single `if` .

`elseif` s provide an additional condition to check (and corresponding code to execute) if the conditional statements of the `if` block and any preceding `elseif` s are not met.

```
$fav_fruit = "orange";

if ($fav_fruit === "banana"){
  echo "Enjoy the banana!";
} elseif ($fav_fruit === "apple"){
  echo "Enjoy the apple!";
} elseif ($fav_fruit === "orange"){
  echo "Enjoy the orange!";
} else {
  echo "Enjoy the fruit!";
}
// Prints: Enjoy the orange!
```

PHP switch statement

PHP `switch` statements provide a clear syntax for a series of comparisons in which a value or expression is compared to many possible matches and code blocks are executed based on the matching `case` .

In PHP, once a matched `case` is encountered, the code blocks of all subsequent cases (regardless of match) will be executed until a `return` , `break` , or the end of the statement is reached. This is known as *fall through*.

```
switch ($letter_grade){
  case "A":
    echo "Terrific";
    break;
  case "B":
    echo "Good";
    break;
  case "C":
    echo "Fair";
    break;
  case "D":
    echo "Needs Improvement";
    break;
  case "F":
    echo "See me!";
    break;
  default:
    echo "Invalid grade";
}
```

PHP readline()

The PHP built-in `readline()` function takes a string with which to prompt the user. It waits for the user to enter text into the terminal and returns that value as a string.

```
echo "\nWhat's your name?\n";
$name = readline(">> "); // receives user
input
```

PHP Boolean Values

PHP Boolean values are either `TRUE` or `FALSE`, which are the only members of the `boolean` type

```
// booleans
$is_true = TRUE;
$is_false = FALSE;

echo gettype($is_true);
// Prints: boolean
echo gettype($is_false);
// Prints: boolean
```

PHP Truthy and Falsy

PHP values within a condition will always be evaluated to `TRUE` or `FALSE`. Values that will evaluate to `TRUE` are known as *truthy* and values that evaluate to `FALSE` are known as *falsy*.
Falsy values include:

- `false`
- `0`
- empty strings
- `null`
- `undefined`
- `NaN`.

All other values are *truthy*.

```
if ("What's going on?"){   // evaluates to
TRUE
  echo "Let us explain…";
}
// Prints: Let us explain…
```

PHP ternary operator

In PHP, the ternary operator allows for a compact syntax in the case of binary (`if/else`) decisions. It evaluates a single condition and executes one expression and returns its value if the condition is met and the second expression otherwise.
The syntax for the ternary operator looks like the following:

```
condition ? expression1 : expression2
```

```
// Without ternary
$isClicked = FALSE;
if ($isClicked) {
  $link_color = "purple";
} else {
  $link_color = "blue";
}
// $link_color = "blue";


// With ternary
$isClicked = FALSE;
$link_color = $isClicked ? "purple" : "blue";
//  $link_color = "blue";
```

PHP Nested Conditionals

In PHP, *nested conditional* statements deepen the complexity of our programs' decision-making capabilities. They allow us to create programs where each decision made sends our program on a different route where it might encounter additional decisions.

```
$num = 5;

// nested conditional
if ($num > 0){
    echo 'The number is positive. <br>';
      if ($num % 2 === 0){
            echo 'The number is even.';
      }
} else {
      echo 'The number is negative.';
}
```

PHP Logical Operators

In PHP, expressions that use logical operators evaluate to boolean values. Logical operators include:

- or (`||`)
- and (`&&`)
- exclusive or (`xor`)
- not (`!`)

```
// Examples of Logical Operators:

TRUE || TRUE;   // Evaluates to: TRUE
FALSE || TRUE;  // Evaluates to: TRUE
TRUE && TRUE;    // Evaluates to: TRUE
FALSE && TRUE;   // Evaluates to: FALSE
!TRUE;    // Evaluates to: FALSE
!FALSE;   // Evaluates to: TRUE
TRUE xor TRUE;   // Evaluates to: FALSE
FALSE xor TRUE;  // Evaluates to: TRUE
```

PHP && Operator

The logical operator `&&` returns:

- `TRUE` only if both of its operands evaluate to true.
- `FALSE` if either or both of its operands evaluate to false.

```
TRUE && TRUE;    // Evaluates to: TRUE
FALSE && TRUE;   // Evaluates to: FALSE
TRUE && FALSE;   // Evaluates to: FALSE
FALSE && FALSE;  // Evaluates to: FALSE

$passingGrades = TRUE;
$extracurriculars = TRUE;
if ($passingGrades && $extracurriculars){
  echo "You meet the graduation
requirements.";
}
// Prints: You meet the graduation
requirements.
```

PHP ! (not) Operator

In PHP, the not operator (`!`) is used to invert a Boolean value or expression.

```
!TRUE;    // Evaluates to: FALSE
!FALSE;   // Evaluates to: TRUE
```

PHP Operator Precedence

Each operator in PHP holds a different *operator precedence*.
We can avoid operator precedence confusion by using parentheses for force the evaluation we want.

```
TRUE || TRUE && FALSE // Evaluates to: TRUE
(TRUE || TRUE) && FALSE // Evaluates to: FALSE
```

PHP Xor Operator

In PHP, the logical operator `xor` stands for exclusive or. It takes two different boolean values or expressions as its operands and returns a single boolean value.

`xor` evaluates to `TRUE` **only** if either its left operand or its right operand evaluate to `TRUE`, but **not both**.

```
TRUE xor TRUE;   // Evaluates to: FALSE
FALSE xor TRUE;  // Evaluates to: TRUE
TRUE xor FALSE;  // Evaluates to: TRUE
FALSE xor FALSE; // Evaluates to: FALSE
```

Logical Operators - Alternate Syntax

PHP provides an alternate syntax for the `||` operator — the `or` operator.

It also provides an alternate syntax for `&&` operator — the `and` operator.
These operators have the advantage of making our code more human readable.

```
// The or Operator:
TRUE or TRUE;   // Evaluates to: TRUE
FALSE or TRUE;  // Evaluates to: TRUE
TRUE or FALSE;  // Evaluates to: TRUE
FALSE or FALSE; // Evaluates to: FALSE

// The and Operator:
TRUE and TRUE;   // Evaluates to: TRUE
FALSE and TRUE;  // Evaluates to: FALSE
TRUE and FALSE;  // Evaluates to: FALSE
FALSE and FALSE; // Evaluates to: FALSE
```

Multi-File Programs: include

A way to improve our code and separate concerns is with *modularity*, separating a program into distinct, manageable chunks where each provides a piece of the overall functionality. Instead of having an entire program located in a single file, code is organized into separate files.
In PHP, files can be included in another file with the keyword `include`. An include statement is followed by a string with a path to the file to be included. The code from the file will be executed.

```
// one.php
echo "How are";

// two.php
echo " you?";

// index.php
echo "Hello! ";
include "one.php";
include "two.php";
// Prints: Hello! How are you?
```