Tennie White

Instructor: Denise Carroll

CS330: Programming Language Project (PLP) Assignment 3

02/10/2025

As I work with PHP, I am learning how it handles boolean values and conditional logic to control program flow. In PHP, boolean values are represented by true and false (case-insensitive), but other values can also be evaluated as boolean. For instance, 1, any non-zero number, and non-empty strings are considered true, while 0, "" (empty string), null, and empty arrays evaluate to false. This dynamic type conversion makes PHP flexible but requires careful handling to avoid unexpected behavior.

PHP provides multiple conditional statements, including if/else, if/elseif/else, and switch. The if/else statement is the most basic, allowing a program to execute different blocks of code based on a condition. The if/elseif/else structure enables handling multiple conditions in sequence. Unlike Perl, PHP does not have an unless statement, but the same logic can be achieved using the if (!condition).

Another important concept in PHP is short-circuit evaluation, which optimizes performance by stopping condition evaluation once the result is determined. In an if statement using && (AND), if the first condition is false, PHP does not evaluate the second condition. Similarly, with || (OR), if the first condition is true, the second condition is skipped.

PHP also addresses the dangling else problem by following the nearest-if rule, where an else statement always pairs with the closest unmatched if. To prevent ambiguity, it is best to use curly braces {} to clearly define which of the else belongs to.

The switch-case statement in PHP allows for efficient multi-condition checking. However, PHP requires the use of breaks to prevent execution from falling through to subsequent cases. Unlike some languages, PHP does not support continue within switch, but continue 2; can be used inside loops containing a switch to skip the rest of the iteration and proceed to the next loop iteration.

Understanding these control structures helps me write more effective PHP programs by ensuring logical flow and preventing unintended behavior.

**1.** What are the boolean values in PHP? (e.g., True and False, true and false, 1, and 0, etc)

Boolean Values in PHP

In PHP, boolean values represent true and false conditions. PHP provides the explicit boolean constants true and false, which are case-insensitive (TRUE and FALSE also work but are not the recommended style).

However, PHP is a loosely typed language, meaning it implicitly converts different data types into boolean values when evaluated in a conditional statement. Here are the main values that are treated as true or false in PHP:

Values Evaluating to true:

* true (explicit boolean value)
* Any non-zero integer (e.g., 1, -1, 100)
* Any non-empty string (e.g., "hello", "false")
* Any non-empty array (e.g., [1, 2, 3])
* Any object
* Any resource

Values Evaluating to false:

* false (explicit boolean value)
* 0 (integer zero)
* 0.0 (floating-point zero)
* "" (empty string)
* "0" (a string containing only zero)
* null (absence of value)
* [] (empty array)
* Objects that implement the \_\_toString() method and return an empty string

This means that PHP automatically interprets many different values as boolean true or false in conditional statements, which can sometimes lead to unexpected behavior if not handled carefully.

Example:

$var = 0;

if ($var) {

echo "This is true";

} else {

echo "This is false"; // This will be executed

}

**2.** What types of conditional statements are available in your language? (if/else, if/then/else, if/elseif/else). Does your language allow for statements other than “if” (for example, Perl has an “unless” statement, which does the opposite of “if”!)

Conditional Statements in PHP

PHP provides multiple ways to handle conditional logic, allowing programmers to execute different blocks of code based on conditions.

1. if Statement

The simplest conditional statement, checks a condition and executes a block of code if it evaluates to true.

$temperature = 30;

if ($temperature > 25) {

echo "It's a hot day!";

}

2. if/else Statement

This structure provides an alternative block of code that executes when the condition is false.

$score = 75;

if ($score >= 50) {

echo "You passed the exam!";

} else {

echo "You failed the exam.";

}

**3**. Does PHP use short-circuit evaluation? Include an example of the short-circuit logic working or not working (or both, if your language is like Java and supports both!)

if/elseif/else Statement

Allows checking multiple conditions in a sequence.

$grade = 85;

if ($grade >= 90) {

echo "You got an A!";

} elseif ($grade >= 80) {

echo "You got a B!";

} elseif ($grade >= 70) {

echo "You got a C!";

} else {

echo "You need to improve.";

}

4.If your language supports switch or case statements, do you have to use “break” to get out of them? Can you use “continue” to have all of them evaluated?

switch Statement

The switch statement is an alternative to multiple if/elseif conditions when comparing a single variable to different values.

$day = "Monday";

switch ($day) {

case "Monday":

echo "Start of the workweek.";

break;

case "Friday":

echo "Weekend is near!";

break;

default:

echo "A regular day.";

}

PHP does not support an unless statement like Perl. However, the same effect can be achieved using if (!condition), which negates the condition.

$isAdmin = false;

if (!$isAdmin) {

echo "Access denied.";

}

Short-Circuit Evaluation in PHP

PHP supports short-circuit evaluation, meaning that when evaluating logical expressions using && (AND) or || (OR), PHP stops execution as soon as the final outcome is determined.

How Short-Circuiting Works

* For && (AND): If the first operand is false, the entire expression is false, and the second operand is not evaluated.
* For || (OR): If the first operand is true, the entire expression is true, and the second operand is not evaluated.

Example of Short-Circuiting in Action

$loggedIn = false;

if ($loggedIn || checkCredentials()) {

echo "User is logged in.";

}

function checkCredentials() {

echo "Checking credentials..."; // This won't execute if $loggedIn is true

return true;

}

If $loggedIn were true, checkCredentials() would not execute because the result of || is already determined.

Short-circuit evaluation improves performance and prevents unnecessary computations or function calls.

4.How does your programming language deal with the “dangling else” problem?

Dangling Else Problem in PHP

The dangling else problem occurs when multiple if statements exist without explicit curly braces, leading to ambiguity in how else statements are paired.

PHP’s Nearest-If Rule

PHP resolves this ambiguity by pairing the else with the closest preceding if statement.

Potentially Confusing Example

$x = 5;

$y = 10;

if ($x > 0)

if ($y < 5)

echo "X is positive and Y is small.";

else

echo "Else belongs to which if?";

Here, the else pairs with if ($y < 5), which may not be the intended logic.

Correct Approach Using Curly Braces

if ($x > 0) {

if ($y < 5) {

echo "X is positive and Y is small.";

} else {

echo "Else correctly paired!";

}

}

Using {} ensures that the else statement is clearly associated with the intended if, making the code more readable and predictable.

Switch-Case: break and continue in PHP

PHP’s switch statement behaves differently from some other languages due to its fall-through behavior.

Break is Required to Exit Cases

Without break, execution falls through to the next case.

$grade = "B";

switch ($grade) {

case "A":

echo "Excellent!";

break;

case "B":

case "C":

echo "Good job!"; // Executes for both "B" and "C"

break;

default:

echo "Keep trying!";

}

Here, "Good job!" will be printed for both "B" and "C".

Fall-Through Example (Without break)

$color = "red";

switch ($color) {

case "red":

echo "Red is a primary color. ";

case "blue":

echo "Blue is a primary color. ";

case "yellow":

echo "Yellow is a primary color.";

}

Output:

Red is a primary color. Blue is a primary color. Yellow is a primary color.

Without break, execution continues through all subsequent cases.

Using continue 2; in Loops with switch

PHP allows continue 2; to skip the rest of a loop iteration that contains a switch statement and continue to the next loop iteration.

for ($i = 1; $i <= 3; $i++) {

switch ($i) {

case 1:

echo "One\n";

continue 2; // Skips the remaining loop body and continues to the next iteration

case 2:

echo "Two\n";

break;

}

}

This feature is useful when a switch is embedded inside a loop.

Understanding Conditional Logic in PHP

As I explore PHP, I realize how crucial conditional statements are for controlling the flow of execution. PHP provides several ways to evaluate conditions, including if/else statements, short-circuit logic, and switch-case structures. These control structures help determine how the program responds based on different inputs and logical conditions.

One of the simplest conditionals in PHP is the one-condition if/else statement. This structure checks if a variable meets a certain condition and executes different blocks of code accordingly. For example, an if statement can determine whether a user is logged in and display a message accordingly.

Beyond simple conditions, PHP allows for multi-condition if/else statements, which evaluate multiple logical expressions using operators like && (AND) and || (OR). These statements help refine decision-making, such as checking if a number falls within a particular range.

For more complex logic, PHP supports if/elseif/else statements, allowing multiple conditions to be checked sequentially. This is useful when different ranges or categories of input require different responses, such as grading a student based on their score.

PHP also supports short-circuit logic, where conditions in logical expressions stop evaluating as soon as the outcome is determined. This improves performance and prevents unnecessary function calls. For instance, in an if statement with ||, if the first condition evaluates to true, the second condition is never checked.

Finally, PHP’s switch-case statement provides a structured way to evaluate a single variable against multiple possible values. This is especially useful when a program needs to execute different code blocks depending on a predefined set of conditions, such as responding to different user input commands.

Understanding these control structures allows me to write efficient and structured PHP programs. Each of these statements plays a role in handling decision-making, ensuring that a program responds dynamically based on given inputs.

Understanding these PHP control structures is essential for writing clear, efficient code. Short-circuit evaluation helps optimize performance, while proper use of break and continue ensures switch cases behave as expected. By using {} brackets, I can avoid dangling else ambiguities and make my conditional logic more readable.