Chapter 4 Expressions and Operators

4.1. Expressions

Expressions are used to do some mathematical operations or string manipulations. An operator combines simple expressions into more complex expressions by creating relationships between simple expressions that can be evaluated.

For example

$x=3+4

The numbers 3 and 4 are each valid expressions. The equation 3 + 4 is also a valid expression, whose value, in this case, happens to be 7. The plus sign (+) is an operator. The numbers to either side of it are its arguments, or operands.

Example 4-1. Sum of values

```
<?php
$x = 3; // x is assigned the value 3
$y = 2; // y is assigned the value 2
$z = $x + $y;
echo $z;
?>
```

Example 4-1 outputs:

```
5
```

4.2. Operator Concepts

PHP has many types of operators. The categories are:

- Arithmetic operators
- Array operators
- Assignment operators
- Bitwise operators
- Comparison operators
- Execution operators
- Incrementing/decrementing operators
- Logical operators
- String operators

Arithmetic Operators

The table below lists the arithmetic operators in PHP:

Operator	**Name**	**Description**	**Example**	**Result**
x + y	Addition	Sum of x and y	2 + 2	4
x - y	Subtraction	Difference of x and y	5 - 2	3
x * y	Multiplication	Product of x and y	5 * 2	10
x / y	Division	Quotient of x and y	15 / 5	3
x % y	Modulus	Remainder of x divided by y	5 % 2 10 % 8 10 % 2	1 2 0
- x	Negation	Opposite of x	- 2	
a . b	Concatenation	Concatenate two strings	"Hi" . "Ha"	HiHa

Assignment Operators

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the expression on the right. That is, the value of "$x = 5" is 5.

Assignment	**Same as...**	**Description**
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

a .= b	a = a . b	Concatenate two strings

Incrementing/Decrementing Operators

Operator	Name	Description
++ x	Pre-increment	Increments x by one, then returns x
x ++	Post-increment	Returns x, then increments x by one
-- x	Pre-decrement	Decrements x by one, then returns x
x --	Post-decrement	Returns x, then decrements x by one

Comparison Operators

Comparison operators allows you to compare two values:

Operator	Name	Description	Example
x == y	Equal	True if x is equal to y	5==8 returns false
x === y	Identical	True if x is equal to y, and they are of same type	5==="5" returns false
x != y	Not equal	True if x is not equal to y	5!=8 returns true
x <> y	Not equal	True if x is not equal to y	5<>8 returns true
x !== y	Not identical	True if x is not equal to y, or they are not of same type	5!=="5" returns true
x > y	Greater than	True if x is greater than y	5>8 returns false
x < y	Less than	True if x is less than y	5<8 returns true
x >= y	Greater than or equal to	True if x is greater than or equal to y	5>=8 returns false
x <= y	Less than or equal to	True if x is less than or equal to y	5<=8 returns true

Logical Operators

Operator	**Name**	**Description**	**Example**

x and y	And	True if both x and y are true	x=6
			y=3
			(x < 10 and y > 1) returns true

x or y	Or	True if either or both x and y are true	x=6
			y=3
			(x==6 or y==5) returns true

x xor y	Xor	True if either x or y is true, but not both	x=6
			y=3
			(x==6 xor y==3) returns false

x && y	And	True if both x and y are true	x=6
			y=3
			(x < 10 && y > 1) returns true

x || y	Or	True if either or both x and y are true	x=6
			y=3
			(x==5 || y==5) returns false

! x	Not	True if x is not true	x=6
			y=3
			!(x==y) returns true

Array Operators

Operator	**Name**	**Description**
x + y	Union	Union of x and y
x == y	Equality	True if x and y have the same key/value pairs
x === y	Identity	True if x and y have the same key/value pairs in the same order and of the same types
x != y	Inequality	True if x is not equal to y

x <> y	Inequality	True if x is not equal to y
x !== y	Non-identity	True if x is not identical to y

Example 4-2. Casting a variable

```
$test=1234;
$test_string=(string)$test;
```

Some invalid expressions are

5=$value //bad – a value cannot be assigned to the left

$a + $b=$c //bad – the expression cannot be assigned to the left.

$value=5 //correct – as the value is assigned to the right of variable after = symbol

$c=$a+$b //correct – as an expression can be assigned to the right of a variable

4.3. Conditionals

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if a condition is true and another code if the condition is false
- **if...elseif....else statement** - use this statement to select one of several blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed
- `? : :` (shorthand for an `if` statement)

4.3.1. The if Statement

The `if` statement offers the ability to execute a block of code, if the supplied condition is TRUE; otherwise, the code block doesn't execute. The type of condition can be any expression, including tests for nonzero, null, equality, variables, and returned values from functions.

No matter what, every single conditional you create includes a conditional clause. If a condition is true, the code block in curly brackets (`{}`) is executed. If not, PHP ignores it and moves to the second condition and continues through as many clauses as you write until PHP hits an `else`, then it automatically executes that block.

If statement

The syntax for the `if` statement is:

```
if (conditional expression)
  {
  block of code;
  }
```

If the expression in the conditional block evaluates to `TRUE`, the block of code after it executes. In this example, if the variable `$uname` is set to guest, a welcome message is printed. Otherwise, nothing happens.

```
if ($uname=="guest")
{
  echo ('Welcome to the guest page.');
  }
```

The curly brackets aren't needed if you want to execute only one statement, but it's good practice to always use them, as it makes the code easier to read and more resilient to change.

4.3.1.1. The else statement

Syntax

```
if (condition)
 {
 code to be executed if condition is true;
 }
else
 {
 code to be executed if condition is false;
 }
```

The `else` statement example 4.6 provides for a default block of code that executes if the condition returned is `FALSE`. It must always be part of an `if` statement, as it doesn't take a conditional itself.

Example 4-6. else and if statements

```
if ($username == "guest")
```

```
{
    echo ('Welcome to the guest page.');
}
else {
    echo ('Permission denied.');
}
```

Remember to close out the code block from the `if` conditional if you used brackets to start the block of code. Similar to the `if` block, the `else` block should also use curly brackets to begin and end the code.

4.3.1.2. The elseif statement

All of this is great except for when you want to test for several conditions at a time. To do this, you can use the `elseif` statement. It allows for testing of additional conditions until one is found to be true or you hit the `else` block. Each `elseif` has its own code block that comes directly after the `elseif` condition. The `elseif` must come after the `if` statement and before an `else` statement if one exists.

The `elseif` structure is a little complicated, but Example 4-7 should help you understand it.

Syntax

```
if (condition)
 {
 code to be executed if condition is true;
 }
elseif (condition)
 {
 code to be executed if condition is true;
 }
else
 {
 code to be executed if condition is false;
 }
```

Example 4-7. Checking multiple conditions

```
if ($username == "Admin"){
    echo ('Welcome to the admin page.');
}
elseif ($username == "Guest"){
    echo ('Please take a look around.');
}
else {
    echo ("Welcome back, $username.");
}
```

Here you can check for and take different action based on two values for `$username`. Then you also have the option to do something else if the `$user_name` isn't one of the first two.

The next construct builds on the concepts of the `if/else` statement, but it allows you to efficiently check the results of an expression to many values without having a separate `if/else` for each value.

4.3.2. The ? Operator

The `?` operator is a ternary operator, meaning it takes three operands. It works like an `if` statement but returns a value from one of the two expressions. The conditional expression determines the value of the expression. A colon (`:`) is used to separate the expressions:

```
{expression} ? return_when_expression_true : return_when_expression_false;
```

Example 4-8 tests a value and returns a different string based on it being TRUE or FALSE.

Example 4-8. Using the ? operator to create a message

```
<?php
$logged_in = TRUE;
$user = "Admin";
$banner = ($logged_in==TRUE)?"Welcome back $user!":"Please login.";
echo "$banner";
?>
```

Example 4-8 produces:

```
Welcome back Admin!
```

This can be pretty useful for checking errors. Now, let's look at a statement that lets you check an expression against a list of possible values to pick the executable code.

4.3.3. The switch Statement

The `switch` statement compares an expression to numerous values. It's very common to have an expression, such as a variable, for which you'll want to execute different code for each value stored in the variable. For example, you might have a variable called `$action`, which may have the values `add, modify`, and `delete`. The `switch` statement makes it easy to define a block of code to execute for each of those values.

To illustrate the difference between using the `if` statement and `switch` to test a variable for several values, we'll show you the code for the `if` statement (in Example 4-9), and then for the `switch` statement (in Example 4-10).

Example 4-9. Using if to test for multiple values

```
if ($action == "ADD") {
    echo "Perform actions for adding.";
    echo "As many statements as you like can be in each block.";
}
elseif ($action == "MODIFY") {
    echo "Perform actions for modifying.";
}
elseif ($action == "DELETE") {
    echo "Perform actions for deleting.";
}
```

Example 4-10. Using switch to test for multiple values

```
switch ($action) {
    case "ADD":
        echo "Perform actions for adding.";
        echo "As many statements as you like can be in each block.";
        break;
    case "MODIFY":
        echo "Perform actions for modifying.";
        break;
    case "DELETE":
        echo "Perform actions for deleting.";
        break;
}
```

The `switch` statement works by taking the value after the `switch` keyword and comparing it to the cases in the order they appear. If no case matches, no code is executed. Once a case matches, the code is executed. The code in subsequent cases also executes until the end of the `switch` statement or until a `break` keyword. This is useful for processes that have several sequential steps. If the user has already done some of the steps, he can jump into the process where he left off.

> The expression after the `switch` statement must evaluate to a simple type like a number, integer, or string. An array can be used only if a specific member of the array is referenced as a simple type.

There are numerous ways to tell PHP to not execute cases besides the matching case.

4.3.3.1. Breaking out

If you want only the code in the matching block to execute, you can place a `break` keyword at the end of that block. When PHP comes across the `break` keyword, processing jumps to the next line after the entire `switch` statement. Example 4-11 illustrates how processing works with no `break` statements.

Example 4-11. What happens when there are no break keywords

```
$action="ASSEMBLE ORDER";
switch ($action) {
    case "ASSEMBLE ORDER":
        echo "Perform actions for order assembly.<br>";
    case "PACKAGE":
        echo "Perform actions for packing.<br>";
    case "SHIP":
        echo "Perform actions for shipping.<br>";
}

   echo "Perform actions for shipping.<br>";
}
?>
```

If the value of `$action` is `"ASSEMBLE ORDER"`, the result is:

```
Perform actions for order assembly.
Perform actions for packing.
Perform actions for shipping.
```

However, if a user had already assembled an order, a value of `"PACKAGE"` produces the following:

```
Perform actions for packing.
Perform actions for shipping.
```

4.3.3.2. Defaulting

The `case` statement also provides a way to do something if none of the other cases match, which is similar to the `else` statement in an `if, elseif, else` block.

Use `DEFAULT:` for the switches last `case` statement, as shown in Example 4-12.

Example 4-12. Using the DEFAULT: statement to generate an error

```
switch ($action) {
    case "ADD":
        echo "Perform actions for adding.";
        break;
    case "MODIFY":
        echo "Perform actions for modifying.";
        break;
    case "DELETE":
        echo "Perform actions for deleting.";
        break;
    default:
        echo "Error: Action must be either ADD, MODIFY, or DELETE.";
}
```

The `switch` statement also supports the alternate syntax in which the `switch` and `endswitch` keywords define the start and end of the switch instead of the curly braces `{}`, as shown in Example 4-13.

Example 4-13. Using endswitch to end the switch definition

```
switch ($action):
    case "ADD":
       echo "Perform actions for adding.";
       break;
    case "MODIFY":
       echo "Perform actions for modifying.";
       break;
    case "DELETE":
       echo "Perform actions for deleting.";
       break;
    default:
       echo "Error: Action must be either ADD, MODIFY, or DELETE.";
endswitch;
```

You've learned that you can have your programs execute different code based on conditions called expressions. The `switch` statement provides a convenient format for checking the value of an expression against many possible values.