

2.1 Understanding Simple Data Types

Every language has different types of variables—and PHP has no shortage of choices. The language supports a wide variety of data types, including simple numeric, character, string, and Boolean types, and more complex arrays and objects. Table 2-1 lists the four basic types, with examples:

Data Type	Description	Example
Boolean	The simplest variable type in PHP, a Boolean variable simply specifies a true or false value.	<code>\$auth = true;</code>
Integer	An integer is a plain-vanilla number like 75, -95, 2000, or 1.	<code>\$age = 99;</code>
Floating-point	A floating-point number is typically a fractional number such as 12.5 or 3.149391239129. Floating point numbers may be specified using either decimal or scientific notation.	<code>\$temperature = 56.89;</code>
String	A string is a sequence of characters, like 'hello' or 'abracadabra'. String values may be enclosed in either double quotes (") or single quotes ('').	<code>\$name = 'Harry';</code>

2.1.1 Detecting the Data Type of a Variable

To find out what type a particular variable is, PHP offers the `gettype()` function, which accepts a variable or value as argument. The following example illustrates this:

```
<?php
// define variables
$auth = true;
$age = 27;
$name = 'Bobby';
$temp = 98.6;

// returns "string"
echo gettype($name);
// returns "boolean"
echo gettype($auth);
// returns "integer"
echo gettype($age);
// returns "double"
echo gettype($temp);
?>
```

PHP also supports a number of specialized functions to check if a variable or value belongs to a specific type. Table 2-2 has a list.

Function	What It Does
<code>is_bool()</code>	Checks if a variable or value is Boolean
<code>is_string()</code>	Checks if a variable or value is a string
<code>is_numeric()</code>	Checks if a variable or value is a numeric string
<code>is_float()</code>	Checks if a variable or value is a floating point number
<code>is_int()</code>	Checks if a variable or value is an integer
<code>is_null()</code>	Checks if a variable or value is NULL
<code>is_array()</code>	Checks if a variable is an array
<code>is_object()</code>	Checks if a variable is an object

** String values enclosed in double quotes are automatically parsed for variable names; if variable names are found, they are automatically replaced with the appropriate variable value.

```
<?php
$identity = 'James Bond';
$car = 'BMW';
// this would contain the string "James Bond drives a BMW"
$sentence = "$identity drives a $car";
// this would contain the string "$identity drives a $car"
$sentence = ' $identity drives a $car';
?>
```

**if your string contains quotes, carriage returns, or backslashes, it's necessary to escape these special characters with a backslash.

```
<?php
// will cause an error due to mismatched quotes
$statement = 'It's hot outside';
// will be fine
$statement = 'It\'s hot outside';
?>
```

Using Arithmetic Operators

To perform mathematical operations on variables, use the standard arithmetic operators, as illustrated in the following example:

```
<?php
// define variables
$num1 = 101;
$num2 = 5;
// add
$sum = $num1 + $num2;
// subtract
$diff = $num1 - $num2;
// multiply
$product = $num1 * $num2;
```

```
// divide
$quotient = $num1 / $num2;
// modulus
$remainder = $num1 % $num2;
?>
```

To perform an arithmetic operation simultaneously with an assignment, use the two operators together. The following two code snippets are equivalent:

```
<?php
$a = $a + 10;
?>
<?php
$a += 10;
?>
```

Operator	What It Does
=	Assignment
+	Addition
-	Subtraction
*	Multiplication
/	Division; returns quotient
%	Division; returns modulus
.	String concatenation
==	Equal to
===	Equal to and of the same type
!==	Not equal to or not of the same type
<> aka !=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
&&	Logical AND
	Logical OR
xor	Logical XOR
!	Logical NOT
++	Addition by 1
--	Subtraction by 1

To add strings together, use the string concatenation operator, represented by a period (.). The following example illustrates this:

```

<?php
$username = 'john';
$domain = 'example.com';
// combine them using the concatenation operator
$email = $username . '@' . $domain;
?>

```

You can concatenate and assign simultaneously, as in the following:

```

<?php
// define string
$str = 'the';
// add and assign
$str .= 'n';
// $str now contains "then"
?>

```

Using Comparison Operators

To test whether two variables are different, use any one of PHP's many comparison operators. The following listing demonstrates most of the important ones:

```

<?php
// define some variables
$mean = 29;
$median = 40;
$mode = 29;
// less-than operator
// returns true if left side is less than right
// returns true here
$result = ($mean < $median);

// greater-than operator
// returns true if left side is greater than right
// returns false here
$result = ($mean > $median);
// less-than-or-equal-to operator
// returns true if left side is less than or equal to right
// returns false here
$result = ($median <= $mode);
// greater-than-or-equal-to operator
// returns true if left side is greater than or equal to right
// returns true here
$result = ($median >= $mode);
// equality operator
// returns true if left side is equal to right
// returns true here
$result = ($mean == $mode);
// not-equal-to operator
// returns true if left side is not equal to right
// returns false here

```

```

$result = ($mean != $mode);
// inequality operator
// returns true if left side is not equal to right
// returns false here
$result = ($mean <> $mode);
?>

```

The === Operator

An important comparison operator in PHP 4.0 is the === operator, which enables you to test both for equality and type. The following listing demonstrates this:

```

<?php
// define two variables

$str = '14';
$int = 14;
// returns true
// since both variables contain the same value
$result = ($str == $int);
// returns false
// since the variables are not of the same type
// even though they have the same value
$result = ($str === $int);
?>

```

Using Logical Operators

To link together related conditions in a simple and elegant manner, use one of PHP's four logical operators—logical AND, logical OR, logical XOR, and logical NOT—as illustrated in the following listing:

```

<?php
// define some variables
$user = 'joe';
$pass = 'trym3';
$saveCookie = 1;
$status = 1;
// logical AND
// returns true if all conditions are true
// returns true here
$result = (($user == 'joe') && ($pass == 'trym3'));
// logical OR
// returns true if any condition is true
// returns false here
$result = (($status < 1) || ($saveCookie == 0));
// logical NOT
// returns true if the condition is false and vice-versa
// returns false
$result = !($saveCookie == 1);
// logical XOR
// returns true if any of the two conditions are true
// returns false if both conditions are true
// returns false here

```

```
$result = (($status == 1) xor ($saveCookie == 1));  
?>
```

Using the Auto-Increment and Auto-Decrement Operators

The *auto-increment* operator is a PHP operator designed to automatically increment the value of the variable it is attached to by 1. It is represented by a double addition symbol (++). To see it in action, run the following script:

```
<?php  
// define $total as 10  
$total = 10;  
// increment it  
$total++;  
// $total is now 11  
?>
```

There's also a corresponding auto-decrement operator (--), which does exactly the opposite:

```
<?php  
// define $total as 10  
$total = 10;  
// decrement it  
$total--;  
// $total is now 9  
?>
```