Chapter 1

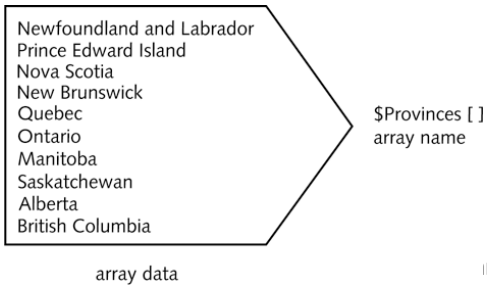
Getting Started with PHP

PHP Programming with MySQL
2nd Edition

Lecture 2

Arrays

An **array** contains a set of data represented by a single variable name



Declaring and Initializing Indexed Arrays

An **element** refers to each piece of data that is stored within an array

An **index** is an element's numeric position within the array

- By default, indexes begin with the number zero (0)
- An element is referenced by enclosing its index in brackets at the end of the array name:

```
$Provinces[1]
```


Declaring and Initializing Indexed Arrays (continued)

The `array()` construct syntax is:

```
$array_name = array(values);
```

```
$Provinces = array(
      "Newfoundland and Labrador",
      "Prince Edward Island",
      "Nova Scotia",
      "New Brunswick",
      "Quebec",
      "Ontario",
      "Manitoba",
      "Saskatchewan",
      "Alberta",
      "British Columbia"
      );
```

Declaring and Initializing Indexed Arrays (continued)

Array name and brackets syntax is:

```
$array_name[ ]
```

```
$Provinces[] = "Newfoundland and Labrador";
$Provinces[] = "Prince Edward Island";
$Provinces[] = "Nova Scotia";
$Provinces[] = "New Brunswick";
$Provinces[] = "Quebec";
$Provinces[] = "Ontario";
$Provinces[] = "Manitoba";
$Provinces[] = "Saskatchewan";
$Provinces[] = "Alberta";
$Provinces[] = "British Columbia";
```


Accessing Element Information (continued)

```
echo "<p>Canada's smallest province is $Provinces[1].<br
  />";
echo "Canada's largest province is $Provinces[4].</p>";
```

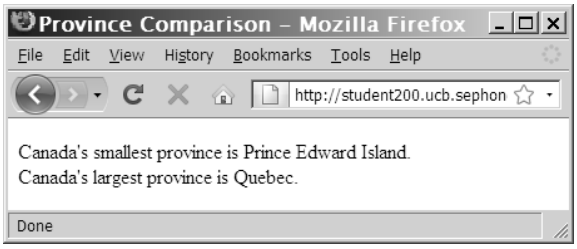


Figure 1-18 Output of elements in the `$Provinces[]` array

Accessing Element Information (continued)

Use the **`count()`** function to find the total number of elements in an array

```
$Provinces = array("Newfoundland and Labrador", "Prince
Edward
Island","Nova Scotia", "New Brunswick","Quebec",
"Ontario", " Manitoba", "Saskatchewan","Alberta", "British
Columbia");


      $Territories = array("Nunavut", "Northwest
Territories", "Yukon
      Territory");


      echo "<p>Canada has ", count($Provinces), "provinces
and ",
       count($Territories), " territories.</p>";
```


Accessing Element Information (continued)

Figure 1-19 Output of the `count()` function

Accessing Element Information (continued)

Use the `print_r()`, `var_dump()` or `var_export()` functions to display or return information about variables

- The `print_r()` function displays the index and value of each element in an array
- The `var_dump()` function displays the index, value, data type and number of characters in the value
- The `var_export()` function is similar to `var_dump()` function except it returns valid PHP code

Accessing Element Information (continued)

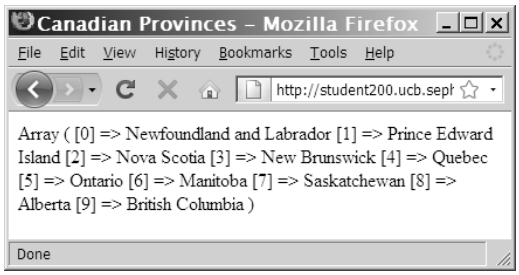


Figure 1-21 Output of the `$Provinces[ ]` array with the `print_r()` function

Modifying Elements

To modify an array element. include the index for an individual element of the array:

```
$HospitalDepts = array(
      "Anesthesia",              // first element(0)
      "Molecular Biology",       // second element
(1)
      "Neurology");              // third element (2)
```

To change the first array element in the `$HospitalDepts[]` array from "Anesthesia" to "Anesthesiology" use:

```
$HospitalDepts[0] = "Anesthesiology";
```


Avoiding Assignment Notation Pitfalls

Assigns the string "Hello" to a variable named $list

```
$list = "Hello";
```

Assigns the string "Hello" to a new element appended to the end of the $list array

```
$list[] = "Hello";
```

Replaces the value stored in the first element (index 0) of the $list array with the string "Hello"

```
$list[0] = "Hello";
```

Building Expressions

An **expression** is a literal value or variable that can be evaluated by the PHP scripting engine to produce a result

Operands are variables and literals contained in an expression

A **literal** is a static value such as a literal string or a number

Operators are symbols (+) (*) that are used in expressions to manipulate operands

Building Expressions (continued)

Type	Description
Array	Performs operations on arrays
Arithmetic	Performs mathematical calculations
Assignment	Assigns values to variables
Comparison	Compares operands and returns a Boolean value
Logical	Performs Boolean operations on Boolean operands
Special	Performs various tasks; these operators do not fit within other operator categories
String	Performs operations on strings

Table 1-2 PHP operator types

Building Expressions (continued)

A **binary operator** requires an operand before and after the operator

- `$MyNumber = 100;`

A **unary operator** requires a single operand either before or after the operator

Arithmetic Operators

Arithmetic operators are used in PHP to perform mathematical calculations (+ - x ÷)

Symbol	Operation	Description
+	Addition	Adds two operands
-	Subtraction	Subtracts the right operand from the left operand
*	Multiplication	Multiplies two operands
/	Division	Divides the left operand by the right operand
%	Modulus	Divides the left operand by the right operand and returns the remainder

Table 1-3 PHP arithmetic binary operators

Arithmetic Operators (continued)

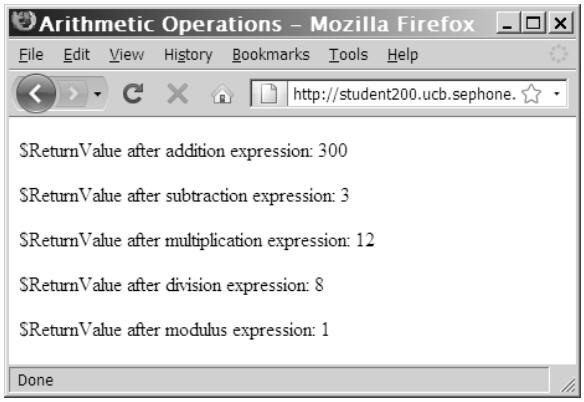
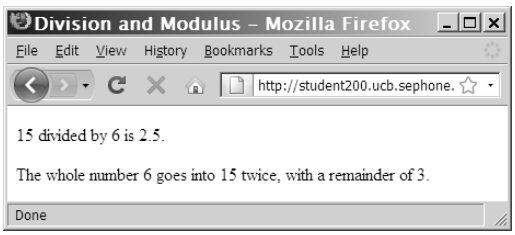


Figure 1-22 Results of arithmetic expressions

Arithmetic Operators (continued)

```
$DivisionResult = 15 / 6;
$ModulusResult = 15 % 6;
echo "<p>15 divided by 6 is
        $DivisionResult.</p>"; // prints '2.5'
echo "The whole number 6 goes into 15 twice, with a
        remainder of $ModulusResult.</p>"; // prints '3'
```



Arithmetic Binary Operators

Symbol	Operation	Description
+	Addition	Adds two operands
-	Subtraction	Subtracts the right operand from the left operand
*	Multiplication	Multiplies two operands
/	Division	Divides the left operand by the right operand
%	Modulus	Divides the left operand by the right operand and returns the remainder

Table 1-3 PHP arithmetic binary operators

Arithmetic Unary Operators

The increment (++) and decrement (--) unary operators can be used as prefix or postfix operators

A **prefix operator** is placed before a variable

A **postfix operator** is placed after a variable

Symbol	Operation	Description
++	Increment	Increases an operand by a value of 1
--	Decrement	Decreases an operand by a value of 1

Table 1-4 PHP arithmetic unary operators

Arithmetic Unary Operators (continued)

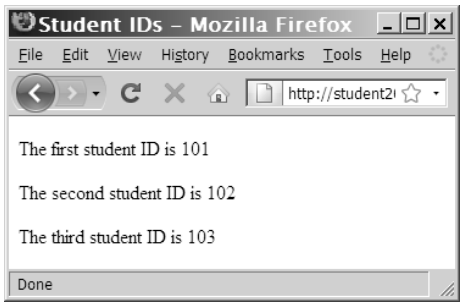
```
$StudentID = 100;
$CurStudentID = ++$StudentID; // assigns '101'
echo "<p>The first student ID is ",
     $CurStudentID, "</p>";
$CurStudentID = ++$StudentID; // assigns '102'
echo "<p>The second student ID is ",
     $CurStudentID, "</p>";
$CurStudentID = ++$StudentID; // assigns '103'
echo "<p>The third student ID is ",
     $CurStudentID, "</p>";
```

prefix increment operator

Figure 1-24 Script that uses the prefix increment operator

Arithmetic Unary Operators (continued)

Figure 1-25 Output of the prefix version of the student ID script



Arithmetic Unary Operators (continued)

```
$StudentID = 100;
$CurStudentID = $StudentID++; // assigns '100'
echo "<p>The first student ID is ",
     $CurStudentID, "</p>";
$CurStudentID = $StudentID++; // assigns '101'
echo "<p>The second student ID is ",
     $CurStudentID, "</p>";
$CurStudentID = $StudentID++; // assigns '102'
echo "<p>The third student ID is ",
     $CurStudentID, "</p>";
```

postfix increment operator

Figure 1-26 Script that uses the postfix increment operator

Arithmetic Unary Operators (continued)

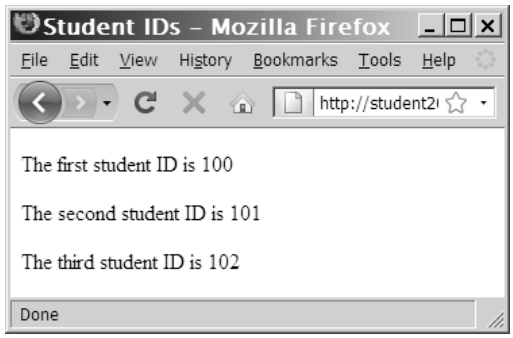


Figure 1-27 Output of the postfix version of the student ID script

Assignment Operators

Assignment operators are used for assigning a value to a variable:

```
$MyFavoriteSuperHero = "Superman";
$MyFavoriteSuperHero = "Batman";
```

Compound assignment operators perform mathematical calculations on variables and literal values in an expression, and then assign a new value to the left operand

Assignment Operators (continued)

Symbol	Operation	Description
=	Assignment	Assigns the value of the right operand to the left operand
+=	Compound addition assignment	Adds the value of the right operand to the value of the left operand and assigns the new value to the left operand
-=	Compound subtraction assignment	Subtracts the value of the right operand from the value of the left operand and assigns the new value to the left operand
*=	Compound multiplication assignment	Multiplies the value of the right operand by the value of the left operand and assigns the new value to the left operand
/=	Compound division assignment	Divides the value of the left operand by the value of the right operand and assigns the new value to the left operand
%=	Compound modulus assignment	Divides the value of the left operand by the value of the right operand and assigns the remainder (modulus) to the left operand

Table 1-5 Common PHP assignment operators

Comparison and Conditional Operators

Comparison operators are used to compare two operands and determine how one operand compares to another

A Boolean value of `TRUE` or `FALSE` is returned after two operands are compared

The comparison operator *compares* values, whereas the assignment operator *assigns* values

Comparison operators are used with **conditional statements** and **looping statements**

Comparison and Conditional Operators (continued)

Symbol	Operation	Description
==	Equal	Returns TRUE if the operands are equal
===	Strict equal	Returns TRUE if the operands are equal and of the same data type
!= or <>	Not equal	Returns TRUE if the operands are not equal
!==	Strict not equal	Returns TRUE if the operands are not equal or not of the same data type
>	Greater than	Returns TRUE if the left operand is greater than the right operand
<	Less than	Returns TRUE if the left operand is less than the right operand
>=	Greater than or equal to	Returns TRUE if the left operand is greater than or equal to the right operand
<=	Less than or equal to	Returns TRUE if the left operand is less than or equal to the right operand

Table 1-6 PHP comparison operators

Comparison and Conditional Operators (continued)

The **conditional operator** executes one of two expressions, based on the results of a conditional expression

The syntax for the conditional operator is:

```
conditional expression ? expression1 : expression2;
```

If the conditional expression evaluates to `TRUE`, *`expression1`* executes

If the conditional expression evaluates to `FALSE`, *`expression2`* executes

Comparison and Conditional Operators (continued)

```
$BlackjackPlayer1 = 20;
 ($BlackjackPlayer1 <= 21) ? $Result =
     "Player 1 is still in the game." : $Result =
     "Player 1 is out of the action.";
 echo "<p>", $Result, "</p>";
```

Figure 1-31 Output of a script with a conditional operator

Logical Operators

Logical operators are used for comparing two Boolean operands for equality

A Boolean value of `TRUE` or `FALSE` is returned after two operands are compared

Symbol	Operation	Description
&& or AND	Logical And	Returns TRUE if both the left operand and right operand return a value of TRUE; otherwise, it returns a value of FALSE
|| or OR	Logical Or	Returns TRUE if either the left operand or right operand returns a value of TRUE; otherwise (neither operand returns a value of TRUE), it returns a value of FALSE
XOR	Logical Exclusive Or	Returns TRUE if only one of the left operand or right operand returns a value of TRUE; otherwise (neither operand returns a value of TRUE or both operands return a value of TRUE), it returns a value of FALSE
!	Logical Not	Returns TRUE if an expression is FALSE and returns FALSE if an expression is TRUE

Table 1-7 PHP logical operators

Special Operators

Symbol	Operation
[and]	Accesses an element of an array
=>	Specifies the index or key of an array element
,	Separates arguments in a list
? and :	Executes one of two expressions based on the results of a conditional expression
instanceof	Returns TRUE if an object is of a specified object type
@	Suppresses any errors that might be generated by an expression to which it is prepended (or placed before)
(int), (integer), (bool), (boolean), (double), (string), (array), (object)	Casts (or transforms) a variable of one data type into a variable of another data type

Table 1-8 PHP special operators

Type Casting

Casting or **type casting** copies the value contained in a variable of one data type into a variable of another data type

The PHP syntax for casting variables is:

```
$NewVariable = (new_type) $OldVariable;
```

`(new_type)` refers to the type-casting operator representing the type to which you want to cast the variable

Type Casting (continued)

Returns one of the following strings, depending on the data type:

- Boolean
- Integer
- Double
- String
- Array
- Object
- Resource
- NULL
- Unknown type

Understanding Operator Precedence

Operator precedence refers to the order in which operations in an expression are evaluated

Associativity is the order in which operators of equal precedence execute

Associativity is evaluated on a left-to-right or a right-to-left basis

Understanding Operator Precedence (continued)

Symbol	Operator	Associativity
`new clone`	New object—highest precedence	None
`[]`	Array elements	Right to left
`++ --`	Increment/Decrement	Right to left
`(int) (double) (string) (array) (object)`	Cast	Right to left
`@`	Suppress errors	Right to left

Understanding Operator Precedence (continued)

instanceof	Types	None
!	Logical Not	Right to left
* / %	Multiplication/division/modulus	Left to right
+ - .	Addition/subtraction/string concatenation	Left to right
< <= > >= <>	Comparison	None
== != === !==	Equality	None
&&	Logical And	Left to right
||	Logical Or	Left to right
?:	Conditional	Left to right
= += -= *= /= %= .=	Assignment	Right to left
AND	Logical And	Left to right
XOR	Logical Exclusive Or	Left to right
OR	Logical Or	Left to right
,	List separator—lowest precedence	Left to right

Table 1-9 Operator precedence in PHP