Module 3

PHP Language Structure: Introduction- Building blocks of PHP-Variables, Data Types -simple PHP program-Converting between Data Types- Operators and Expressions -Flow Control functions - Control statements- Working with Functions- Initialising and Manipulating Arrays-- Objects-String Comparisons-String processing with Regular Expression

- **PHP, or PHP: Hypertext Preprocessor, has become the most popular server-side scripting**
- language for creating dynamic web pages. PHP was created by Rasmus Lerdorf to track users at his website. In 1995, Lerdorf released it as a package called the "Personal Home Page Tools."
- Two years later, PHP 2 featured built-in database support and form handling.
- In 1997, PHP 3 was released after a substantial rewrite, which resulted in a large increase in performance and led to an explosion of PHP use.
- The release of PHP 4 featured the new *Zend Engine from Zend, a PHP software company. This version was considerably* faster and more powerful than its predecessor, further increasing PHP's popularity.
-

- It's estimated that over 15 million domains now use PHP, accounting for more than 20 percent of web pages.1 Currently, PHP 5 features the *Zend Engine 2, which provides further speed* increases, exception handling and a new object-oriented programming model.
- PHP is an open-source technology that's supported by a large community of users and developers.
- PHP is *platform independent—implementations exist for all major UNIX,* Linux, Mac and Windows operating systems. PHP also supports many databases,including MySQL.


```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.1: first.php -->
4  <!-- Simple PHP program. -->
5  <html>
6  <?php
7     $name = "Paul"; // declaration and initialization
8  ?><!-- end PHP script -->
9     <head>
10       <meta charset = "utf-8">
11       <title>Simple PHP document</title>
12    </head>
13    <body>
14       <!-- print variable name's value -->
15       <h1><?php print( "Welcome to PHP, $name!" ); ?></h1>
```

```
16     </body>
17  </html>
```

Type	Description
`int, integer`	Whole numbers (i.e., numbers without a decimal point).
`float, double, real`	Real numbers (i.e., numbers containing a decimal point).
`string`	Text enclosed in either single (`''`) or double (`""`) quotes. [*Note:* Using double quotes allows PHP to recognize more escape sequences.]
`bool, boolean`	`true` or `false`.
`array`	Group of elements.
`object`	Group of associated data and methods.
`resource`	An external source—usually information from a database.
`NULL`	No value.

Fig. 19.2 | PHP types.

Display Data in a web browser from PHP Script

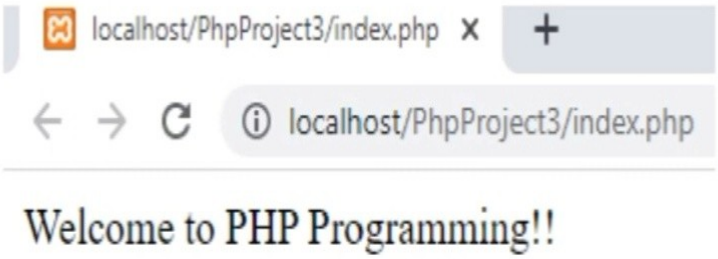
- With PHP, there are two basic ways to get output:
  - echo and print.
- The PHP echo Statement

```
<?php
    echo "Welcome to PHP Programming!!";
?>
```

Display Data in a web browser from PHP Script

- The print statement

```
<?php
    print "Welcome to PHP Programming!!";
?>
```



Creating Comments in PHP Program

Single Line Comments: Can be created using // or #

```
<!DOCTYPE html>
<html>
<body>

<?php
// This is a single-line comment

# This is also a single-line comment
?>

</body>
</html>
```

Multiline Comments

```
<!DOCTYPE html>
<html>
<body>

<?php
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
?>

</body>
</html>
```

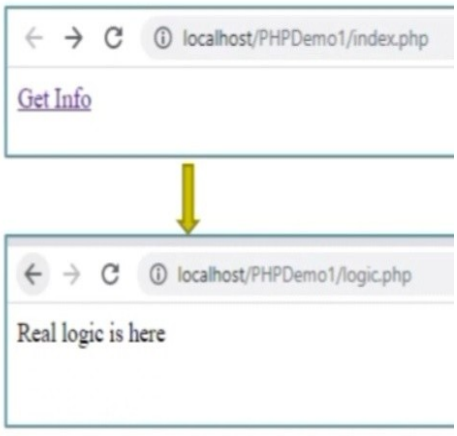
Creating PHP file with only PHP Code

index.php

```
<html>
    <head>
    </head>
    <body>
      <a href="logic.php">Get Info</a>
    </body>
</html>
```

logic.php

```
<?php
  echo "Real logic is here";
?>
```



Including PHP file in another PHP program

- It is possible to insert the content of one PHP file into another PHP file
- The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Syntax

```
include 'filename';

or

require 'filename';
```

Example

```
<!DOCTYPE html>
<?php include 'logic.php' ?>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php require 'logic.php' ?>
    </body>
</html>
```

logic.php

```
<?php
    echo "PHP code logic..";
?>
```

Creating Variables in PHP

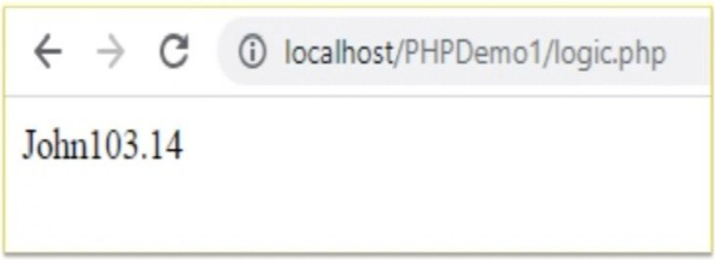
- In PHP, a variable must starts with the $ sign, followed by the name of the variable:
- It is a faster memory access operator like pointer in C programming.

```php
<?php
    $name = "John";
    $company = 'Infosys';
    $number = 10;
    $pi = 3.14;
?>
```

Display variable data

- We can use echo or print command to display variable data.

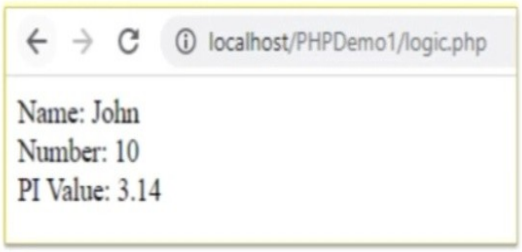
```php
<?php
    $name = 'John';
    $number = 10;
    $pi = 3.14;

    echo $name;
    echo $number;
    echo $pi;
?>
```



String concatenation

- In PHP, the dot(.) operator can be used to concatenate the strings as well as other data values.

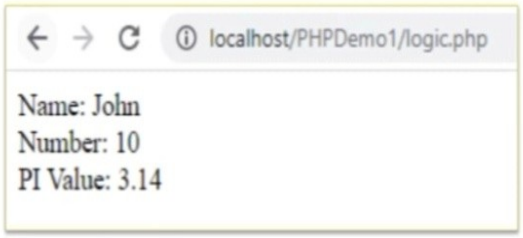
```php
<?php
    $name = 'John';
    $number = 10;
    $pi = 3.14;

    echo "Name: " . $name . "<br>";
    echo "Number: " . $number. "<br>";
    echo "PI Value: " . $pi . "<br>";
 ?>
```



Without (dot .) operator ...

- Using double quotes we can perform the same concatenation.

```php
<?php
    $name = 'John';
    $number = 10;
    $pi = 3.14;

    echo "Name: $name <br>";
    echo "Number: $number <br>";
    echo "PI Value: $pi <br>";
 ?>
```

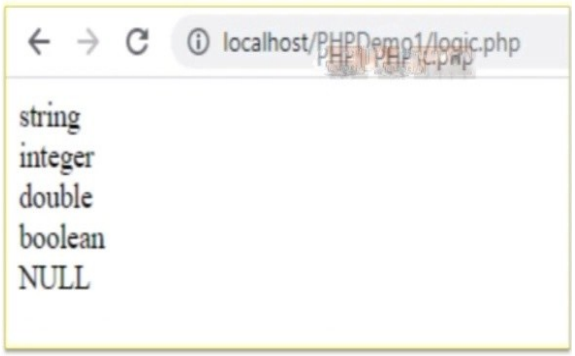


PHP Data Types

- Explicit type declaration is not needed in PHP
- PHP supports the following data types:
  - string
  - integer
  - float (floating point numbers)
  - boolean
  - array
  - object
  - NULL
- The gettype() method used to find the type of the given data.

PHP Data Types

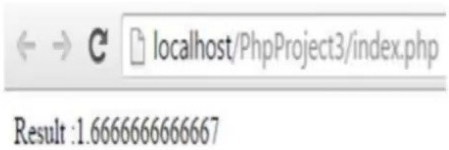
```php
<?php
    $name = 'John';
    $number = 10;
    $pi = 3.14;
    $flag = false;
    $value = NULL ;

    echo gettype($name) . "<br>";
    echo gettype($number) . "<br>";
    echo gettype($pi) . "<br>";
    echo gettype($flag) . "<br>";
    echo gettype($value) . "<br>";
?>
```



Type Casting

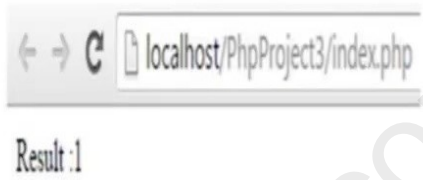
- Typecasting is the explicit conversion of data type because user explicitly defines the data type in which he wants to cast.

```php
<?php
   $a = 10;
   $b = 6;
   $result = $a/$b;
   echo "Result :". $result;
?>
```



To convert this float to a Integer value, need to use type casting operator (data type)

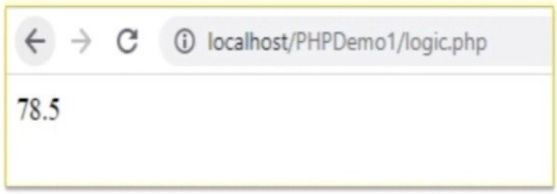
```php
<?php
   $a = 10;
   $b = 6;
   $result = (int)($a/$b);
   echo "Result :". $result;
?>
```



Creating constant

- To create a constant, use the define() function.
  - define(name, value, case-insensitive)
    - case-insensitive – default is false

```php
<?php
     #creating constant
     define('pi',3.14);

     #finding area of a circle
     $radius = 5;
     $area = pi * $radius ** 2;
     echo $area;
 ?>
```

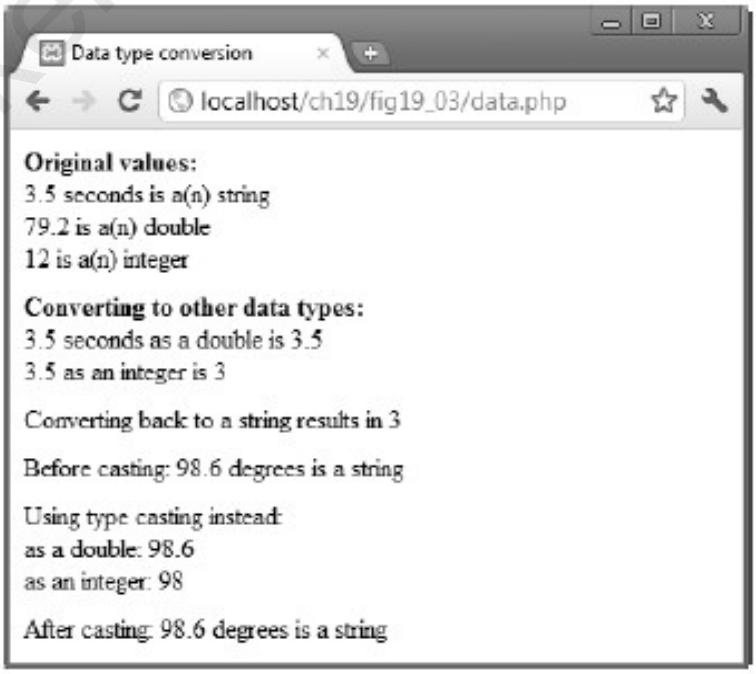


Converting Between data types

```
<!DOCTYPE html>

<!-- Fig. 19.3: data.php -->
<!-- Data type conversion. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Data type conversion</title>
      <style type = "text/css">
         p      { margin: 0; }
         .head  { margin-top: 10px; font-weight: bold; }
         .space { margin-top: 10px; }
      </style>
   </head>
   <body>
      <?php
         // declare a string, double and integer
         $testString = "3.5 seconds";
         $testDouble = 79.2;
         $testInteger = 12;
      ?><!-- end PHP script -->


      <!-- print each variable's value and type -->
      <p class = "head">Original values:</p>
      <?php
         print( "<p>$testString is a(n) " . gettype( $testString )
            . "</p>" );
         print( "<p>$testDouble is a(n) " . gettype( $testDouble )
            . "</p>" );
         print( "<p>$testInteger is a(n) " . gettype( $testInteger )
            . "</p>" );
      ?><!-- end PHP script -->
      <p class = "head">Converting to other data types:</p>
      <?php
         // call function settype to convert variable
         // testString to different data types
         print( "<p>$testString " );
         settype( $testString, "double" );
         print( " as a double is $testString</p>" );
         print( "<p>$testString " );
         settype( $testString, "integer" );
         print( " as an integer is $testString</p>" );
         settype( $testString, "string" );
         print( "<p class = 'space'>Converting back to a string results in
            $testString</p>" );

```

```
47          // use type casting to cast variables to a different type
48          $data = "98.6 degrees";
49          print( "<p class = 'space'>Before casting: $data is a " .
50             gettype( $data ) . "</p>" );
51          print( "<p class = 'space'>Using type casting instead:</p>
52             <p>as a double: " . (double) $data . "</p>" .
53             "<p>as an integer: " . (integer) $data . "</p>";
```

```
54             print( "<p class = 'space'>After casting: $data is a " .
55                gettype( $data ) . "</p>" );
56          ?><!-- end PHP script -->
57       </body>
58    </html>
```



PHP Operators

- Operators are used to perform operations on variables and values.
- PHP divides the operators in the following groups:
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Increment/Decrement operators
  - Logical operators
  - String operators
  - Array operators
  - Conditional assignment operators

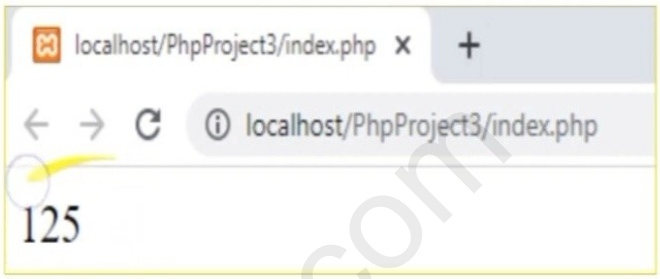
Arithmetic operators

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

Example

- Exponentiation Operator

```php
<?php
    $x = 5;
    $y = 3;
    $z = $x ** $y;
    echo $z;
?>
```

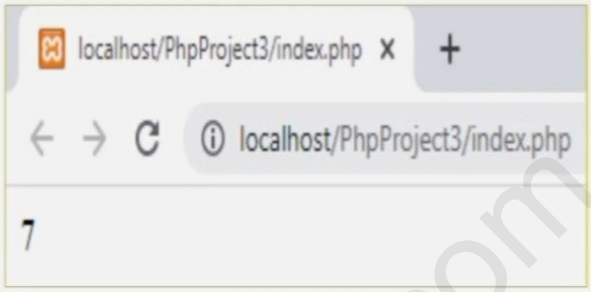


PHP Assignment Operators

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

Example: Addition Assignment

```php
<?php
    $x = 5;
    $x += 2;        // $x = $x + 2
    echo $x;
?>
```



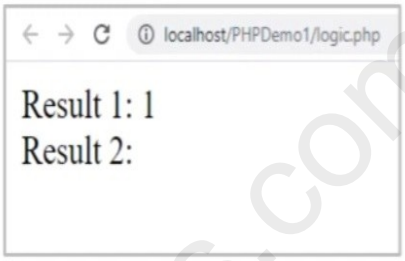
PHP Comparison Operators

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type

Example 1

=== comparison (Same type and same value)

```
<?php
      $a = 10;
      $b = 10;
      echo "Result 1: " . ($a===$b) . "<br>";

      $a = 10;
      $b = "10";
      echo "Result 2: " . ($a===$b) . "<br>";
 ?>
```



Example 2

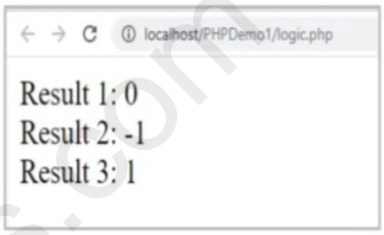
- <> operator

```
<?php
      $a = 10;
      $b = 5;

      echo "Result 1: " . ($a<>$b) . "<br>";
      echo "Result 2: " . ($a!=$b) . "<br>";
 ?>
```

Example 3

- <=> spaceship operator - return 0 or -1 or +1
  - Compare the order of numbers used for sorting

```php
<?php
    $a = 10;
    $b = 10;

    echo "Result 1: " . ($a<=>$b) . "<br>";

    $a = 5;
    $b = 10;
    echo "Result 2: " . ($a<=>$b) . "<br>";

    $a = 10;
    $b = 5;
    echo "Result 3: " . ($a<=>$b) . "<br>";
?>
```

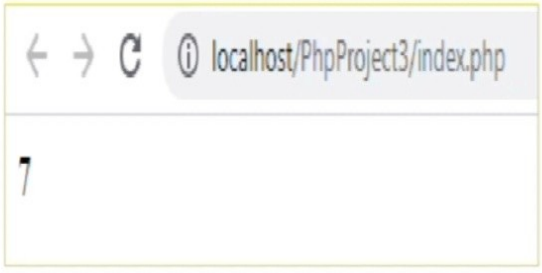


PHP Increment / Decrement Operators

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

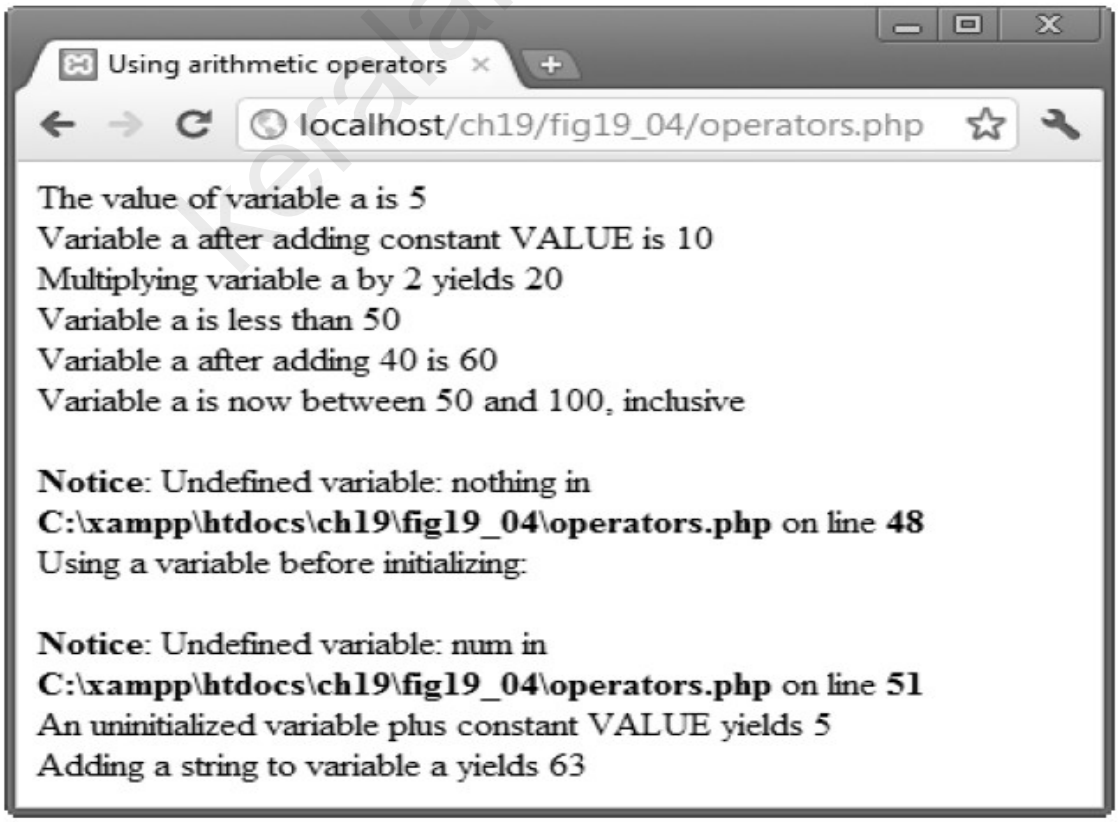
Example:

```php
<?php
    $x = 5;
    $x++;      //$x increments by 1
    ++$x;      //$x increments by 1
    echo $x;
?>
```



```
<!DOCTYPE html>

<!-- Fig. 19.4: operators.php -->
<!-- Using arithmetic operators. -->
<html>
   <head>
      <meta charset = "utf-8">
      <style type = "text/css">
      p { margin: 0; }
      </style>
      <title>Using arithmetic operators</title>
   </head>
   <body>
      <?php
         $a = 5;
         print( "<p>The value of variable a is $a</p>" );

         // define constant VALUE
         define( "VALUE", 5 );

         // add constant VALUE to variable $a
         $a = $a + VALUE;
         print( "<p>Variable a after adding constant VALUE is $a</p>" );


         // multiply variable $a by 2
         $a *= 2;
         print( "<p>Multiplying variable a by 2 yields $a</p>" );

         // test if variable $a is less than 50
         if ( $a < 50 )
            print( "<p>Variable a is less than 50</p>" );

         // add 40 to variable $a
         $a += 40;
         print( "<p>Variable a after adding 40 is $a</p>" );

         // test if variable $a is 50 or less
         if ( $a < 51 )
            print( "<p>Variable a is still 50 or less</p>" );
         elseif ( $a < 101 ) // $a >= 51 and <= 100
            print( "<p>Variable a is now between 50 and 100,
               inclusive</p>" );
         else // $a > 100
            print( "<p>Variable a is now greater than 100</p>" );

         // print an uninitialized variable
         print( "<p>Using a variable before initializing:
            $nothing</p>" ); // nothing evaluates to ""

         // add constant VALUE to an uninitialized variable
         $test = $num + VALUE; // num evaluates to 0
         print( "<p>An uninitialized variable plus constant
            VALUE yields $test</p>" );
```

```
54
55          // add a string to an integer
56          $str = "3 dollars";
57          $a += $str;
58          print( "<p>Adding a string to variable a yields $a</p>" );
59       ?><!-- end PHP script -->
60    </body>
61 </html>
```



Operator	Type	Associativity
`new` `clone`	constructor copy an object	none
`[]`	subscript	left to right
`++` `--`	increment decrement	none

`~`
`-`
`@`
`(type)`
	bitwise not
	unary negative
	error control
	cast
		right to left

`instanceof`		none
`!`	not	right to left

`*`
`/`
`%`
	multiplication
	division
	modulus
		left to right

`+`
`-`
`.`
	addition
	subtraction
	concatenation
		left to right

`<<` `>>`	bitwise shift left bitwise shift right	left to right

`<`
`>`
`<=`
`>=`
	less than
	greater than
	less than or equal
	greater than or equal
		none

`==`
`!=`
`===`
`!==`
	equal
	not equal
	identical
	not identical
		none

`&`	bitwise AND	left to right
`^`	bitwise XOR	left to right
`|`	bitwise OR	left to right
`&&`	logical AND	left to right
`||`	logical OR	left to right
`?:`	ternary conditional	left to right

Fig. 19.6 | PHP operator precedence and associativity. (Part 1 of 2.)

Operator	Type	Associativity
=	assignment	right to left
+=	addition assignment	
-=	subtraction assignment	
*=	multiplication assignment	
/=	division assignment	
%=	modulus assignment	
&=	bitwise AND assignment	
|=	bitwise OR assignment	
^=	bitwise exclusive OR assignment	
.=	concatenation assignment	
<<=	bitwise shift left assignment	
>>=	bitwise shift right assignment	
=>	assign value to a named key	
and	logical AND	left to right
xor	exclusive OR	left to right
or	logical OR	left to right
,	list	left to right

Fig. 19.6 | PHP operator precedence and associativity. (Part 2 of 2.)

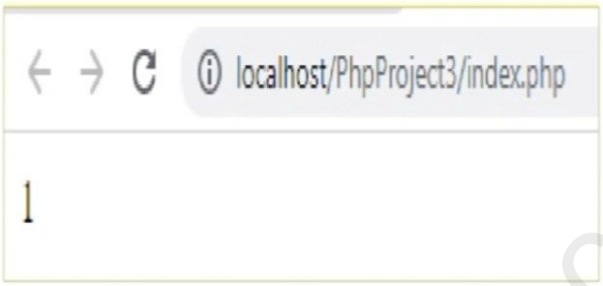
Logical operators

TABLE 12.5 Logical Operators

Operator	Name	Returns True If...	Example	Result
||	Or	Left or right is true.	`true || false`	`true`
or	Or	Left or right is true.	`true or false`	`true`
xor	Xor	Left or right is true, but not both.	`true xor true`	`false`
&&	And	Left and right are true.	`true && false`	`false`
and	And	Left and right are true.	`true and false`	`false`
!	Not	The single operand is not true.	`! true`	`false`

Example

```
<?php
    $a = 10;
    $b = 4;
    $c = 15;
    echo $a>$b and $b<$c;
?>
```



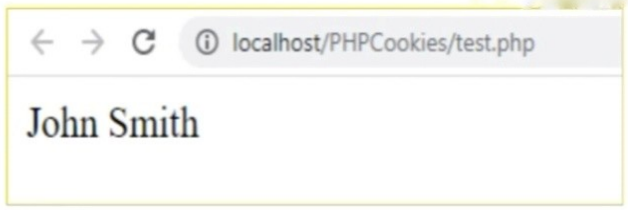
PHP String Operators

- PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

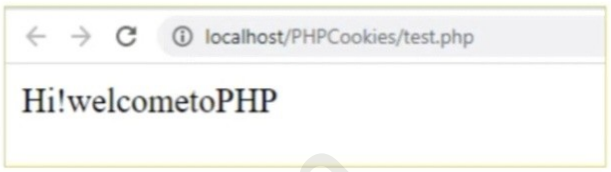
Example

```php
<?php
    $firstname='John';
    $secondname='Smith';
    echo $firstname .' '. $secondname;
?>
```



Concatenation using comma (,) operator

```php
<?php
    echo 'Hi!','welcome','to','PHP';
?>
```



PHP Array Operators

- The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

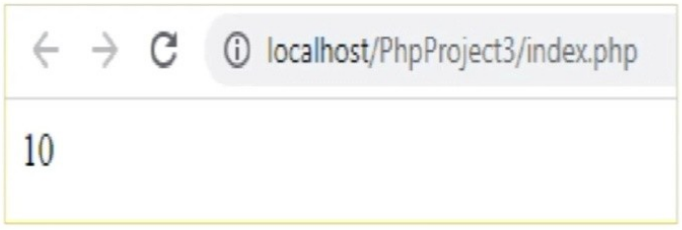
PHP Conditional Assignment Operators

- The PHP conditional assignment operators are used to set a value depending on conditions:

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE

Example

```php
<?php
    $a = 10;
    $b = 4;
    $result = $a>$b ? $a : $c;
    echo $result;
?>
```



PHP Control Statements

- PHP supports the following control statements
  - Conditional Statements
  - Loop Statements

Conditional Statements

In PHP we have the following conditional statements:

- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif...else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

The if Statement

- The if statement executes some code if one condition is true.

Syntax

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

Example: -

```
<?php
    $t = 21;
    if ($t < "20") {
      echo "Have a good day!";
    }
?>
```

The if...else Statement

- The if...else statement executes some code if a condition is true and another code if that condition is false.

Syntax

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

```
<?php
    $t = 10;
    if ($t < "20") {
       echo "Have a good day!";
     } else {
     echo "Have a good night!";
     }
?>
```

The if...elseif...else Statement

- The if...elseif...else statement executes different codes for more than two conditions.

```
Syntax
if (condition) {
  code to be executed if this condition is true;
} elseif (condition) {
  code to be executed if first condition is false and this condition is true;
} else {
  code to be executed if all conditions are false;
}
```

The PHP switch Statement

- Use the switch statement to select one of many blocks of code to be executed.

```
Syntax
    switch (n) {
      case label1:
        code to be executed if n=label1;
        break;
      case label2:
        code to be executed if n=label2;
        break;
      case label3:
        code to be executed if n=label3;
        break;
        ...
      default:
        code to be executed if n is different from all labels;
    }
```

Example

```php
<?php
    $favcolor = "red";
    switch ($favcolor) {
        case "red":
               echo "Your favorite color is red!";
               break;
        case "blue":
               echo "Your favorite color is blue!";
               break;
        case "green":
                echo "Your favorite color is green!";
               break;
        default:
               echo "Your favorite color is neither red, blue, nor green!";
 }
?>
```


PHP Loops

In PHP, we have the following loop types:

- while - loops through a block of code as long as the specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array

The PHP while Loop

- The while loop executes a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
  code to be executed;
}
```

```php
<?php
   $x = 1;
   while($x <= 5) {
       echo "The number is: $x <br>";
       $x++;
   }
?>
```

do...while Loop

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do {
  code to be executed;
} while (condition is true);
```

```php
<?php
   $x = 1;
   do {
       echo "The number is: $x <br>";
       $x++;
   } while ($x <= 5);
?>
```

The PHP for Loop

- The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
 code to be executed for each iteration;
}
```

```php
<?php
  for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
  }
?>
```


Initializing and Manipulating Arrays

- PHP provides the capability to store data in arrays.
- Arrays are divided into elements that behave as individual variables. Array names, like other variables, begin with the $ symbol.
- Individual array elements are accessed by following the array's variable name with an index enclosed in square brackets ([]).
- Assigning a value to an element where the index is omitted *appends a new element to the end of the array*

Some Array-Related Constructs and Functions

- **count() and sizeof()—Each of these functions counts the number of elements in an** array; sizeof() is an alias of count(). Given the array $colors = array("blue", "black", "red", "green"); both count($colors); and sizeof($colors); return a value of 4.
- **each() and list()—These functions (well, list() is a language construct that *looks*** like a function) usually appear together, in the context of stepping through an array and returning its keys and values. You saw an example of this previously, where we steppe through the $c array and printed its contents.
- **foreach()—This control structure (which looks like a function) is used to step through an** array, assigning the value of an element to a given variable.

- **reset()—This function rewinds the pointer to the beginning of an array, as in this** example: reset($character); This function proves useful when you are performing multiple manipulations on an array, such as sorting, extracting values, and so forth.
- **array_push()—This function adds one or more elements to the end of an existing array,** as in this example: array_push($existingArray, "element 1", "element 2", "element 3");
- **array_pop()—This function removes (and returns) the last element of an existing array,** as in this example: $last_element = array_pop($existingArray);
- **array_unshift()—This function adds one or more elements to the beginning of an**
- existing array, as in this example: array_unshift($existingArray, "element 1", "element 2", "element 3");

- **array_shift()—This function removes (and returns) the first element of an existing** array, as in this example, where the value of the element in the first position of $existingArray is assigned to the variable $first_element: $first_element = array_shift($existingArray);
- **array_merge()—This function combines two or more existing arrays, as in this example:** $newArray = array_merge($array1, $array2);
- **array_keys()—This function returns an array containing all the key names within a** given array, as in this example: $keysArray = array_keys($existingArray);
- **array_values()—This function returns an array containing all the values within a** given array, as in this example: $valuesArray = array_values($existingArray);
- **shuffle()—This function randomizes the elements of a given array. The syntax of this** function is simply as follows: shuffle($existingArray);

- Explode()

Explode a string into substring and store it in array

Explode(delemiter,string);

Example:

$str="April in paris";

$words=explode(" ",$str);

Output

$word=("April","in","paris")

- Implode()

Convert an array of strings to a single string,seperating th parts with a specified string

Example:

$str=implode(":",$words);

$str="April :in :paris"

Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```


Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

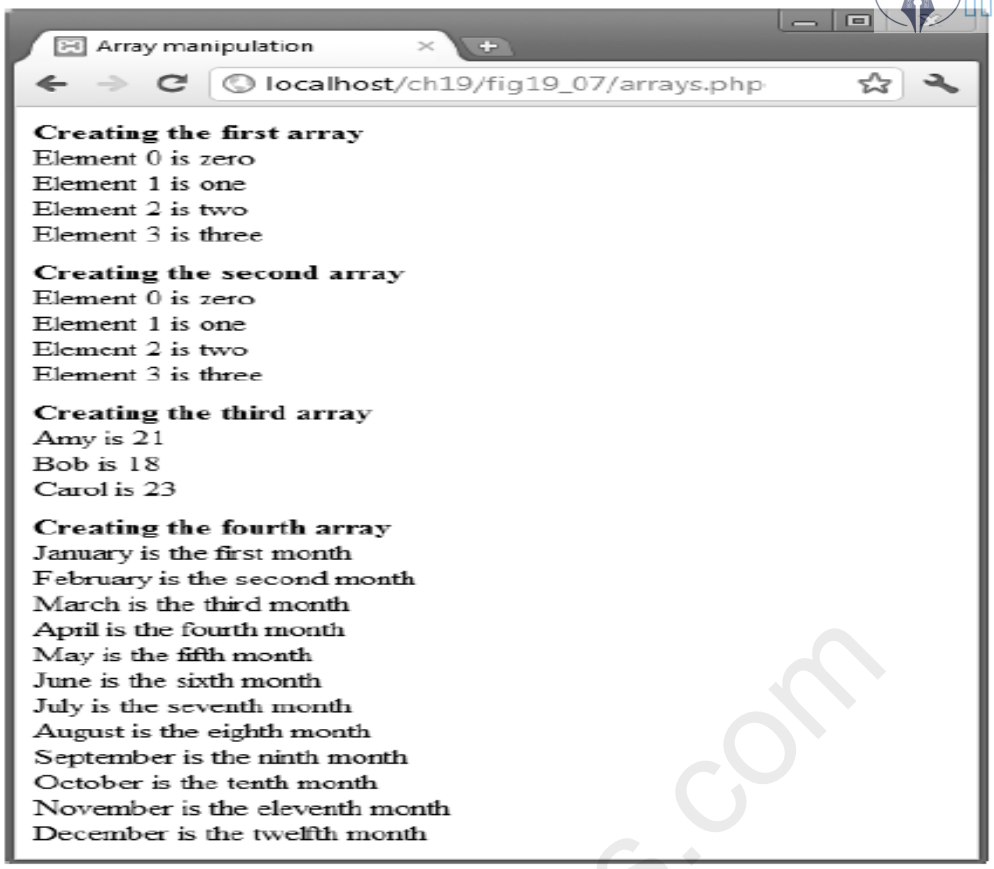
or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37",
"Joe"=>"43");

foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>
```

```
<!DOCTYPE html>

<!-- Fig. 19.7: arrays.php -->
<!-- Array manipulation. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Array manipulation</title>
      <style type = "text/css">
         p    { margin: 0; }
         .head { margin-top: 10px; font-weight: bold; }
      </style>
   </head>
   <body>
      <?php
         // create array first
         print( "<p class = 'head'>Creating the first array</p>" );
         $first[ 0 ] = "zero";
         $first[ 1 ] = "one";
         $first[ 2 ] = "two";
         $first[] = "three";

         // print each element's index and value
         for ( $i = 0; $i < count( $first ); ++$i )
            print( "Element $i is $first[$i]</p>" );

         print( "<p class = 'head'>Creating the second array</p>" );




         // call function array to create array second
         $second = array( "zero", "one", "two", "three" );

         for ( $i = 0; $i < count( $second ); ++$i )
            print( "Element $i is $second[$i]</p>" );

         print( "<p class = 'head'>Creating the third array</p>" );

         // assign values to entries using nonnumeric indices
         $third[ "Amy" ] = 21;
         $third[ "Bob" ] = 18;
         $third[ "Carol" ] = 23;

         // iterate through the array elements and print each
         // element's name and value
         for ( reset( $third ); $element = key( $third ); next( $third ) )
            print( "<p>$element is $third[$element]</p>" );

         print( "<p class = 'head'>Creating the fourth array</p>" );

         // call function array to create array fourth using
         // string indices
         $fourth = array(
            "January"   => "first",   "February" => "second",
            "March"     => "third",   "April"    => "fourth",
            "May"       => "fifth",   "June"     => "sixth",
            "July"      => "seventh", "August"   => "eighth",
            "September" => "ninth",   "October"  => "tenth",
            "November"  => "eleventh","December" => "twelfth" );

         // print each element's name and value
         foreach ( $fourth as $element => $value )
            print( "<p>$element is the $value month</p>" );
      ?><!-- end PHP script -->
   </body>
</html>
```



- Line 30 demonstrates a second method of initializing arrays.
- Function **array creates** an array that contains the arguments passed to it.
- The first item in the argument list is stored as the first array element (recall that the first element's index is 0), the second item is stored as the second array element and so on. Lines 32–33 display the array's contents.
- In addition to integer indices, arrays can have float or nonnumeric indices (lines 38– 40). An array with noninteger indices is called an **associative array.**
- **For example, indices** Amy, Bob and Carol are assigned the values 21, 18 and 23, respectively.

- PHP provides functions for **iterating through the elements of an array (line 44).**
- **Each** array has a built-in **internal pointer, which points to the array element currently being referenced.**
- Function **reset sets the internal pointer to the first array element. Function key** returns the index of the element currently referenced by the internal pointer, and function
- **next moves the internal pointer to the next element and returns the element.**
- The array $fourth is also associative. To override the automatic numeric indexing performed by function array, you can use operator =>, as demonstrated in lines 51–57.

- The value to the left of the operator is the array index and the value to the right is the element's value.
- The **foreach control statement (lines 60–61) is specifically designed for iterating** through arrays, especially associative arrays, because it does not assume that the array has consecutive integer indices that start at 0.
- The foreach statement starts with the array to iterate through, followed by the keyword **as, followed by two variables—the first is** assigned the index of the element, and the second is assigned the value of that index.

Functions

Defining a Function

You can define your own functions using the function statement:

```
function some_function($argument1, $argument2)
{
//function code here
}
```


LISTING 13.2 Declaring and Calling a Function

```
<?php
function bighello()
{
     echo "<h1>HELLO!</h1>";
}
bighello();
?>
```

Example

```
1: <?php
2: function printBR($txt)
3: {
4:      echo $txt."<br>";
5: }
6: printBR("This is a line.");
7: printBR("This is a new line.");
8: printBR("This is yet another line.");
9: ?>
```

Returning values from user-defined function

LISTING 13.4 A Function That Returns a Value

```
1: <?php
2: function addNums($firstnum, $secondnum)
3: {
4:       $result = $firstnum + $secondnum;
5:       return $result;
6: }
7: echo addNums(3,5);
8: //will print "8"
9: ?>
```

Passing variable references to the functions

LISTING 13.12 Passing an Argument to a Function by Value

```
1: <?php
2: function addFive($num)
3: {
4:      $num += 5;
5: }
6: $orignum = 10;
7: addFive($orignum);
8: echo $orignum;
9: ?>
```

LISTING 13.13 Using a Function Definition to Pass an Argument to a Function by Reference

```
1: <?php
2: function addFive(&$num)
3: {
4:      $num += 5;
5: }
6: $orignum = 10;
7: addFive($orignum);
8: echo $orignum;
9: ?>
```

Understanding Variable Scope

- A variable declared within a function remains local to that function. In other words, it is not available outside the function or within other functions.

LISTING 13.5 Variable Scope: A Variable Declared Within a Function Is Unavailable Outside the Function

```
<?php
function test()
{
     $testvariable = "this is a test variable";
}
echo "test variable: ".$testvariable."<br>";
?>
```

FIGURE 13.2
Output of `scopetest.php`.

Accessing variables with the global statement

LISTING 13.6 Variables Defined Outside Functions Are Inaccessible from Within a Function by Default

```
1: <?php
2: $life = 42;
3: function meaningOfLife()
4: {
5:      echo "The meaning of life is ".$life";
6: }
7: meaningOfLife();
8: ?>
```

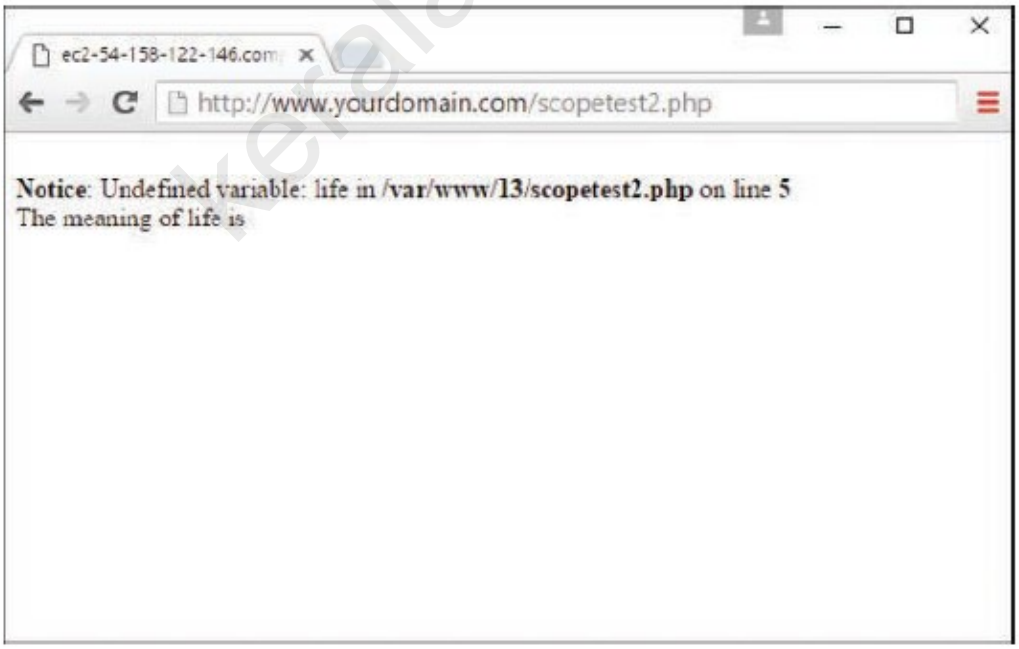


FIGURE 13.3
Attempting to reference a variable from outside the scope of a function.

LISTING 13.7 Accessing Global Variables with the `global` Statement

```
1: <?php
2: $life=42;
3: function meaningOfLife()
4: {
5:      global $life;
6:      echo "The meaning of life is ".$life";
7: }
8: meaningOfLife();
9: ?>
```

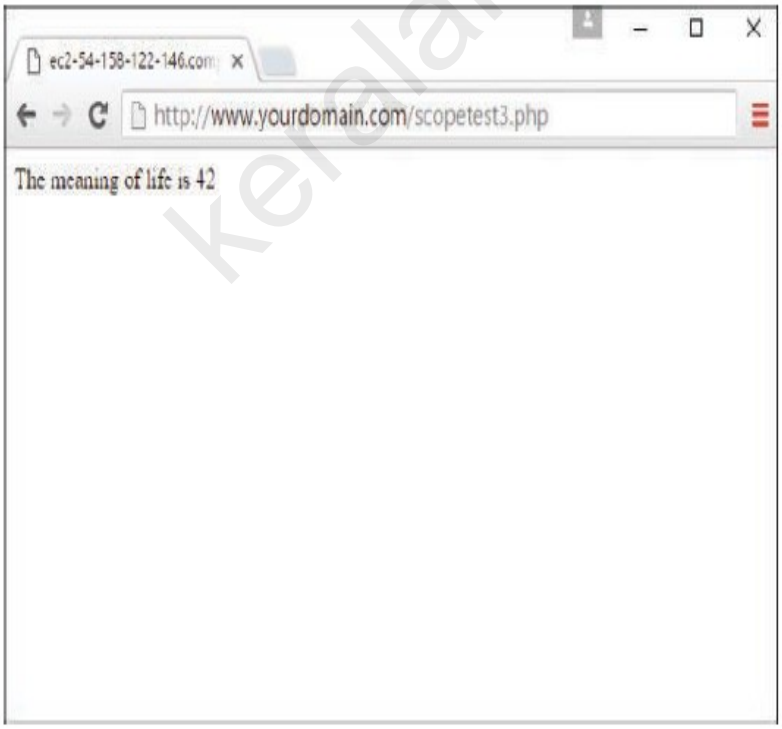



FIGURE 13.4
Successfully accessing a global variable from within a function using the `global` statement.

Objects

- Creating an object is simple; you just declare it to be in existence:

```
class myClass {
//code will go here
}
```

- Now that you have a class, you can create a new instance of an object:

```
$object1 = new myClass();
```


LISTING 13.15 Proof That Your Object Exists

```
1:  <?php
2:  class myClass {
3:      //code will go here
4:  }
5:  $object1 = new myClass();
6:  echo "\$object1 is an ".gettype($object1).".<br>";
7:
8:  if (is_object($object1)) {
9:      echo "Really! I swear \$object1 is an object!";
10: }
11: ?>
```

Properties of Objects

The variables declared inside an object are called *properties*. It is standard practice to declare your variables at the top of the class. These properties can be values, arrays, or even other objects. The following snippet uses simple scalar variables inside the class, prefaced with the `public` keyword:

```
class myCar {
     public $color = "silver";
     public $make = "Mazda";
     public $model = "Protege5";
}
```


LISTING 13.16 Showing Object Properties

```
1:  <?php
2:  class myCar {
3:     public $color = "blue";
4:     public $make = "Jeep";
5:     public $model = "Renegade";
6: }
7: $car = new myCar();
8: echo "I drive a: " . $car->color . " ".$car->make . " " . $car->model;
9: ?>
```

If you save this code as `objproperties.php`, place it in your document root, and access it with your web browser, you will see the following on your screen:

```
I drive a: blue Jeep Renegade
```

LISTING 13.17 Changing Object Properties

```
1:  <?php
2:  class myCar {
3:     public $color = "blue";
4:     public $make = "Jeep";
5:     public $model = "Renegade";
6:  }
7:  $car = new myCar();
8:  $car->color = "red";
9:  $car->make = "Porsche";
10: $car->model = "Boxster";
11: echo "I drive a: " . $car->color . " " . $car->make . " ". $car->model;
12: ?>
```

If you save this code as `objproperties2.php`, place it in your document root, and access it with your web browser, you will see the following on your screen:

```
I drive a: red Porsche Boxster
```


Object Methods

LISTING 13.18 A Class with a Method

```
<?php
class myClass {
     public function sayHello() {
          echo "HELLO!";
     }
}
$object1 = new myClass();
$object1->sayHello();
?>
```

Although it is not the most thrilling example of action, if you save this code as `helloclass.php`, place it in your document root, and access it with your web browser, you will see the following on your screen:

```
HELLO!
```

A method looks and acts like a normal function but is defined within the framework of a class. The -> operator is used to call the object method in the context of your script. Had there been any variables stored in the object, the method would have been capable of accessing them for its own purposes, as illustrated in Listing 13.19.

LISTING 13.19 Accessing Class Properties Within a Method

```
1:  <?php
2:  class myClass {
3:     public $name = "Jimbo";
4:     public function sayHello() {
5:          echo "HELLO! My name is " . $this->name;
6:     }
7:  }
8:  $object1 = new myClass();
9:  $object1->sayHello();
10: ?>
```

If you save this code as `helloclass2.php`, place it in your document root, and access it with your web browser, you will see the following on your screen:

```
HELLO! My name is Jimbo
```

The special variable `$this` is used to refer to the currently instantiated object as you see on line 5. Any time an object refers to itself, you must use the `$this` variable. Using the `$this` variable in conjunction with the -> operator enables you to access any property or method in a class, within the class itself.

LISTING 13.20 Changing the Value of a Property from Within a Method

```
1: <?php
2: class myClass {
3:    public $name = "Jimbo";
4:    public function setName($n) {
5:         $this->name = $n;
6:    }
```

```
7:    public function sayHello() {
8:         echo "HELLO! My name is ".$this->name;
9:    }
10: }
11: $object1 = new myClass();
12: $object1->setName("Julie");
13: $object1->sayHello();
14: ?>
```

If you save this code as `helloclass3.php`, place it in your document root, and access it with your web browser, you will see the following on your screen:

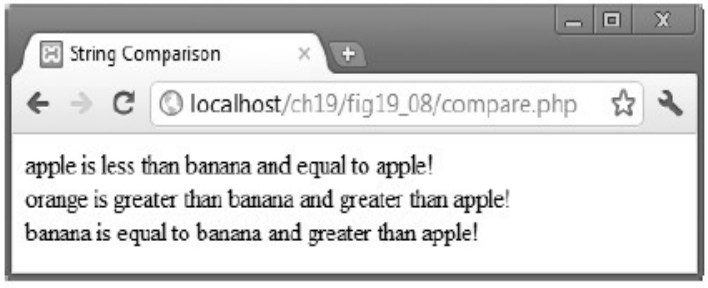
```
HELLO! My name is Julie
```

Why? Because in lines 4–6, a new function called `setName()` was created. When it is called in line 12, it changes the value of `$name` to `Julie`. Therefore, when the `sayHello()` function is called in line 13 and it looks for `$this->name`, it uses `Julie`, which is the new value that was just set by the `setName()` function. In other words, an object can modify its own property—in this case, the `$name` variable.

String Comparisons

- Many string-processing tasks can be accomplished by using the **equality and comparison** operators,
- Fig. 19.8. Line 16 declares and initializes array $fruits. Lines 19–38 iterate through each element in the $fruits array.
- Lines 23 and 25 call function **strcmp to compare two strings. The function returns** -1 if the first string alphabetically precedes the second string, 0 if the strings are equal, and 1 if the first string alphabetically follows the second. Lines 23–28 compare each element in the $fruits array to the string "banana", printing whether each is greater than, less than or equal to the string.
- Relational operators (==, !=, <, <=, > and >=) can also be used to compare strings. Lines 32–37 use relational operators to compare each element of the array to the string "apple".

```
<!DOCTYPE html>

<!-- Fig. 19.8: compare.php -->
<!-- Using the string-comparison operators. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>String Comparison</title>
      <style type = "text/css">
         p { margin: 0; }
      </style>
   </head>
   <body>
      <?php
         // create array fruits
         $fruits = array( "apple", "orange", "banana" );

         // iterate through each array element
         for ( $i = 0; $i < count( $fruits ); ++$i )
         {
            // call function strcmp to compare the array element
            // to string "banana"
            if ( strcmp( $fruits[ $i ], "banana" ) < 0 )
               print( "<p>" . $fruits[ $i ] . " is less than banana " );

            elseif ( strcmp( $fruits[ $i ], "banana" ) > 0 )
               print( "<p>" . $fruits[ $i ] . " is greater than banana ")
            else
               print( "<p>" . $fruits[ $i ] . " is equal to banana " );

            // use relational operators to compare each element
            // to string "apple"
            if ( $fruits[ $i ] < "apple" )
               print( "and less than apple!</p>" );
            elseif ( $fruits[ $i ] > "apple" )
               print( "and greater than apple!</p>" );
            elseif ( $fruits[ $i ] == "apple" )
               print( "and equal to apple!</p>" );
         } // end for
      ?><!-- end PHP script -->
   </body>
</html>
```



String Processing with Regular Expressions

- PHP can process text easily and efficiently, enabling straightforward searching, substitution, extraction and concatenation of strings. Text manipulation is usually done with **regular expressions—a series of characters that serve as *pattern-matching templates (or search*** criteria) in strings, text files and databases.
- Function **preg_match uses regular expressions** to search a string for a specified pattern using **Perl-compatible regular expressions (PCRE).**

```
<!DOCTYPE html>

<!-- Fig. 19.9: expression.php -->
<!-- Regular expressions. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Regular expressions</title>
      <style type = "text/css">
         p { margin: 0; }
      </style>
   </head>

   <body>
      <?php
         $search = "Now is the time";
         print( "<p>Test string is: '$search'</p>" );

         // call preg_match to search for pattern 'Now' in variable search
         if ( preg_match( "/Now/", $search ) )
            print( "<p>'Now' was found.</p>" );

         // search for pattern 'Now' in the beginning of the string
         if ( preg_match( "/^Now/", $search ) )
            print( "<p>'Now' found at beginning of the line.</p>" );

         // search for pattern 'Now' at the end of the string
         if ( !preg_match( "/Now$/", $search ) )
            print( "<p>'Now' was not found at the end of the line.</p>" );
```

```
29
30          // search for any word ending in 'ow'
31          if ( preg_match( "/\b([a-zA-Z]*ow)\b/i", $search, $match ) )
32             print( "<p>Word found ending in 'ow': " .
33                $match[ 1 ] . "</p>" );
34
35          // search for any words beginning with 't'
36          print( "<p>Words beginning with 't' found: " );
37
38          while ( preg_match( "/\b(t[[:alpha:]]+)\b/", $search, $match ) )
39          {
40             print( $match[ 1 ] . " " );
41
42             // remove the first occurrence of a word beginning
43             // with 't' to find other instances in the string
44             $search = preg_replace("/" . $match[ 1 ] . "/", "", $search);
45          } // end while
46
47          print( "</p>" );
48       ?><!-- end PHP script -->
49    </body>
50 </html>
```

Searching for Expressions

- Line 15 assigns the string "Now is the time" to variable $search. The condition in line 19 calls function preg_match to search for the **literal characters "Now" inside variable** $search.
- If the pattern is found, preg_match returns the length of the matched string— which evaluates to true in a boolean context —and line 20 prints a message indicating that the pattern was found.
- We use single quotes (') inside the string in the print statement to emphasize the search pattern. *Anything enclosed in single quotes is not interpolated, unless the single quotes are nested in a double-quoted string literal*

- Function preg_match takes two arguments—a regular-expression pattern to search for and the string to search.
- The regular expression must be enclosed in delimiters—typically a forward slash (/) is placed at the beginning and end of the regular-expression pattern.
- By default, preg_match performs a *case-sensitive pattern matches. To perform caseinsensitive pattern matches you simply place the letter i after the regular-expression pattern's* closing delimiter, as in "/\b([a-zA-Z]*ow)\b/i" (line 31).

Representing Patterns

- In addition to literal characters, regular expressions can include **metacharacters, such as ^,** $ and ., that specify patterns. The **caret (^) metacharacter matches the beginning of a** string (line 23), while the **dollar sign ($) matches the end of a string (line 27). The period (.) metacharacter matches any single character except newlines, but can be made to match** newlines with the s modifier.
- Line 23 searches for the pattern "Now" at the beginning of $search. Line 27 searches for "Now" at the end of $search. Note that Now$ is *not a variable*— it's a *pattern that uses $ to search for the characters "Now" at the end of a string.*

- Line 31, which contains a bracket expression, searches (from left to right) for the first word ending with the letters ow. **Bracket expressions are lists of characters enclosed in** square brackets ([]) that match any single character from the list.
- Ranges can be specified by supplying the beginning and the end of the range separated by a **dash (-). For instance,** the bracket expression [a-z] matches any *lowercase letter and [A-Z] matches any uppercase* letter.
- In this example, we combine the two to create an expression that matches *any letter.*
- The \b before and after the parentheses indicates the beginning and end of a word, respectively— in other words, we're attempting to match whole words.

- The expression [a-zA-Z]*ow inside the parentheses (line 31) represents any word ending in ow. The **quantifier * matches the preceding pattern zero or more times.**
- **Thus,** [a-zA-Z]*ow matches any number of letters followed by the literal characters ow. Quantifiers are used in regular expressions to denote how often a particular character or set of characters can appear in a match.

Quantifier	Matches
{*n*}	Exactly *n* times
{*m*,*n*}	Between *m* and *n* times, inclusive
{*n*,}	*n* or more times

Fig. 19.10 | Some regular expression quantifiers. (Part 1 of 2.)

Quantifier	Matches
+	One or more times (same as {1,})
*	Zero or more times (same as {0,})
?	Zero or one time (same as {0,1})

Fig. 19.10 | Some regular expression quantifiers. (Part 2 of 2.)

Finding Matches

- The optional third argument to function preg_match is an array that stores matches to the regular expression. When the expression is broken down into parenthetical sub-expressions, preg_match stores the first encountered instance of each expression in this array, starting from the leftmost parenthesis.
- the statement in line 31 is the first parenthetical pattern, Now is stored in variable $match[1]
- Lines 38–45 use a while statement and the **preg_replace function to find all the words in the string that begin** with t.

Character Classes

- The pattern in line 38, /\b(t[[:alpha:]]+)\b/i, matches any word beginning with the character t followed by one or more letters.
- The pattern uses the **character class [[:alpha:]]** to recognize any letter—this is equivalent to the [a-zA-Z].

Character class	Description
`alnum`	Alphanumeric characters (i.e., letters [a-zA-Z] or digits [0-9])
`alpha`	Word characters (i.e., letters [a-zA-Z])
`digit`	Digits
`space`	White space
`lower`	Lowercase letters
`upper`	Uppercase letters

Fig. 19.11 | Some regular expression character classes.

- Character classes are enclosed by the delimiters [: and :]. When this expression is placed in another set of brackets, such as [[:alpha:]] in line 38, it's a regular expression matching a single character that's a member of the class. A bracketed expression containing
- two or more adjacent character classes in the class delimiters represents those character sets combined. For example, the expression [[:upper:][:lower:]]* represents all strings of uppercase and lowercase letters in any order,
- while [[:upper:]][[:lower:]]* matches strings with a single uppercase letter followed by any number of lowercase characters.
- The expression ([[:upper:]][[:lower:]])* represents all strings that alternate between uppercase and lowercase characters (starting with uppercase and ending with lowercase).

Finding Multiple Instances of a Pattern

- The quantifier + matches one or more consecutive instances of the preceding expression. The result of the match is stored in $match[1]. Once a match is found, we print it in line 40.
- We then remove it from the string in line 44, using function **preg_replace. This function** takes three arguments—the pattern to match, a string to replace the matched string and the string to search. The modified string is returned. Here, we search for the word that
- we matched with the regular expression, replace the word with an empty string, then assign the result back to $search. This allows us to match any other words beginning with the character t in the string and print them to the screen.

Form Processing and Business Logic

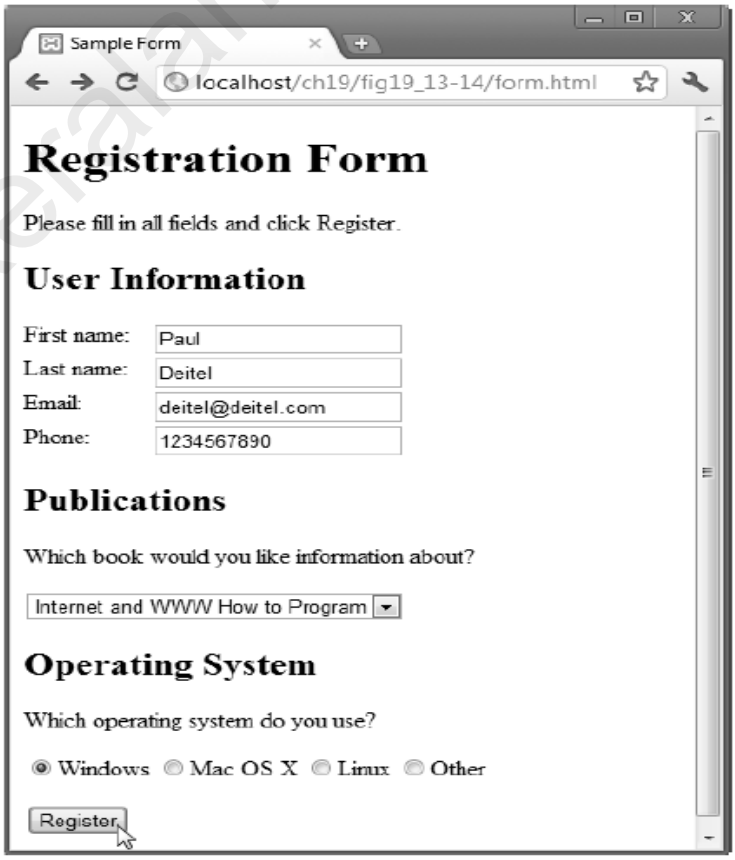
Superglobal Arrays

- Superglobal arrays are associative arrays predefined by PHP that hold variables acquired from user input, the environment or the web server, and are accessible in any variable scope.
- Superglobal arrays are useful for verifying user input. The arrays $_GET and $_POST retrieve information sent to the server by HTTP get and post requests, respectively, making it possible for a script to have access to this data when it loads another page
- For_example, if data entered by a user into a form is posted to a script, the $_POST array will contain all of this information in the new script. Thus, any information entered into the form can be accessed easily from a confirmation page, or a page that verifies whether fields have been entered correctly.

```
<!DOCTYPE html>

<!-- Fig. 19.13: form.html -->
<!-- HTML form for gathering user input. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Sample Form</title>
      <style type = "text/css">
         label  { width: 5em; float: left; }
      </style>
   </head>
   <body>
      <h1>Registration Form</h1>
      <p>Please fill in all fields and click Register.</p>

      <!-- post form data to form.php -->
      <form method = "post" action = "form.php">
         <h2>User Information</h2>

         <!-- create four text boxes for user input -->
         <div><label>First name:</label>
            <input type = "text" name = "fname"></div>
         <div><label>Last name:</label>
            <input type = "text" name = "lname"></div>

         <div><label>Email:</label>
            <input type = "text" name = "email"></div>
         <div><label>Phone:</label>
            <input type = "text" name = "phone"
               placeholder = "(555) 555-5555"></div>
         </div>

         <h2>Publications</h2>
         <p>Which book would you like information about?</p>

         <!-- create drop-down list containing book names -->
         <select name = "book">
            <option>Internet and WWW How to Program</option>
```

```
39            <option>C++ How to Program</option>
40            <option>Java How to Program</option>
41            <option>Visual Basic How to Program</option>
42         </select>
43
44         <h2>Operating System</h2>
45         <p>Which operating system do you use?</p>
46
47         <!-- create five radio buttons -->
48         <p><input type = "radio" name = "os" value = "Windows"
49               checked>Windows
50            <input type = "radio" name = "os" value = "Mac OS X">Mac OS X
51            <input type = "radio" name = "os" value = "Linux">Linux
52            <input type = "radio" name = "os" value = "Other">Other</p>
53
54         <!-- create a submit button -->
55         <p><input type = "submit" name = "submit" value = "Register"></p>
56      </form>
57   </body>
58 </html>
```

The form is filled out with an incorrect phone number

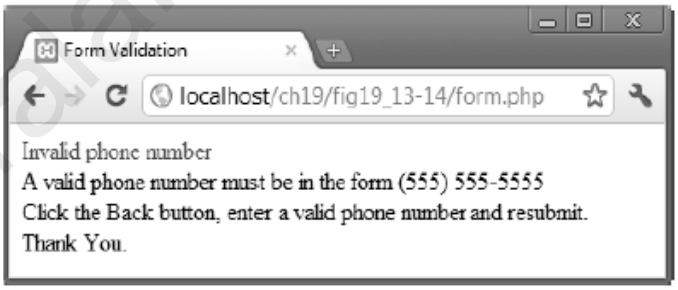


- The form's action attribute (line 18) indicates that when the user clicks the **Register** button, the form data will be posted to form.php (Fig. 19.14) for processing.
- Using method = "post" appends form data to the browser request that contains the protocol (i.e., HTTP) and the URL of the requested resource (specified by the action attribute).
- Scripts located on the web server's machine can access the form data sent as part of the request.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.14: form.php -->
4  <!-- Process information sent from form.html. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Form Validation</title>
9        <style type = "text/css">
10          p       { margin: 0px; }
11          .error  { color: red }
12          p.head  { font-weight: bold; margin-top: 10px; }
13       </style>
14    </head>
15    <body>
16       <?php
17          // determine whether phone number is valid and print
18          // an error message if not
19          if (!preg_match( "/^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$/",
20             $_POST["phone"]))
21          {
22             print( "<p class = 'error'>Invalid phone number</p>
23                <p>A valid phone number must be in the form
24                (555) 555-5555</p><p>Click the Back button,
25                enter a valid phone number and resubmit.</p>
26                <p>Thank You.</p></body></html>" );
27             die(); // terminate script execution
28          }
```

```
29        ?><!-- end PHP script -->
30        <p>Hi <?php print( $_POST["fname"] ); ?>. Thank you for
31           completing the survey. You have been added to the

32           <?php print( $_POST["book"] ); ?>mailing list.</p>
33        <p class = "head">The following information has been saved
34           in our database:</p>
35        <p>Name: <?php print( $_POST["fname"] );
36           print( $_POST["lname"] ); ?></p>
37        <p>Email: <?php print( "$email" ); ?></p>
38        <p>Phone: <?php print( "$phone" ); ?></p>
39        <p>OS: <?php print( $_POST["os"] ); ?></p>
40        <p class = "head">This is only a sample form.
41           You have not been added to a mailing list.</p>
42     </body>
43  </html>
```

a) Submitting the form in Fig. 19.13 redirects the user to `form.php`, which gives appropriate instructions if the phone number is in an incorrect format



b) The results of form.php after the user submits the form in Fig. 19.13 with a phone number in a valid format

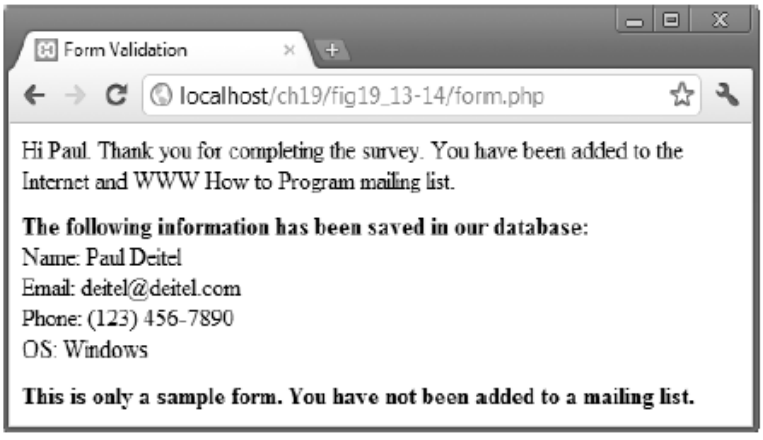


Fig. 19.14 | Process information sent from `form.html`. (Part 2 of 2.)