Web Technology
Unit 4
Server Side Scripting:PHP

By Dr Prasanna B T
SJCE,JSSSTU,Mysuru

Client Script :

- Script downloaded and Executes on the client browser
- The clients never get to see the code, just the HTML output from the script
- Client cannot access web server resources

Server script:

- In a server-side script, it is executed on the web server
- The server sends the JavaScript (that the user could look at), but you have no guarantee that the script will even execute

- In contrast, server-side source code remains hidden from the client as it is processed on the server
- Server scripts cannot manipulate the HTML or DOM of a page in the client browser as is possible with client scripts
- a server script can access resources on the web server whereas the client cannot

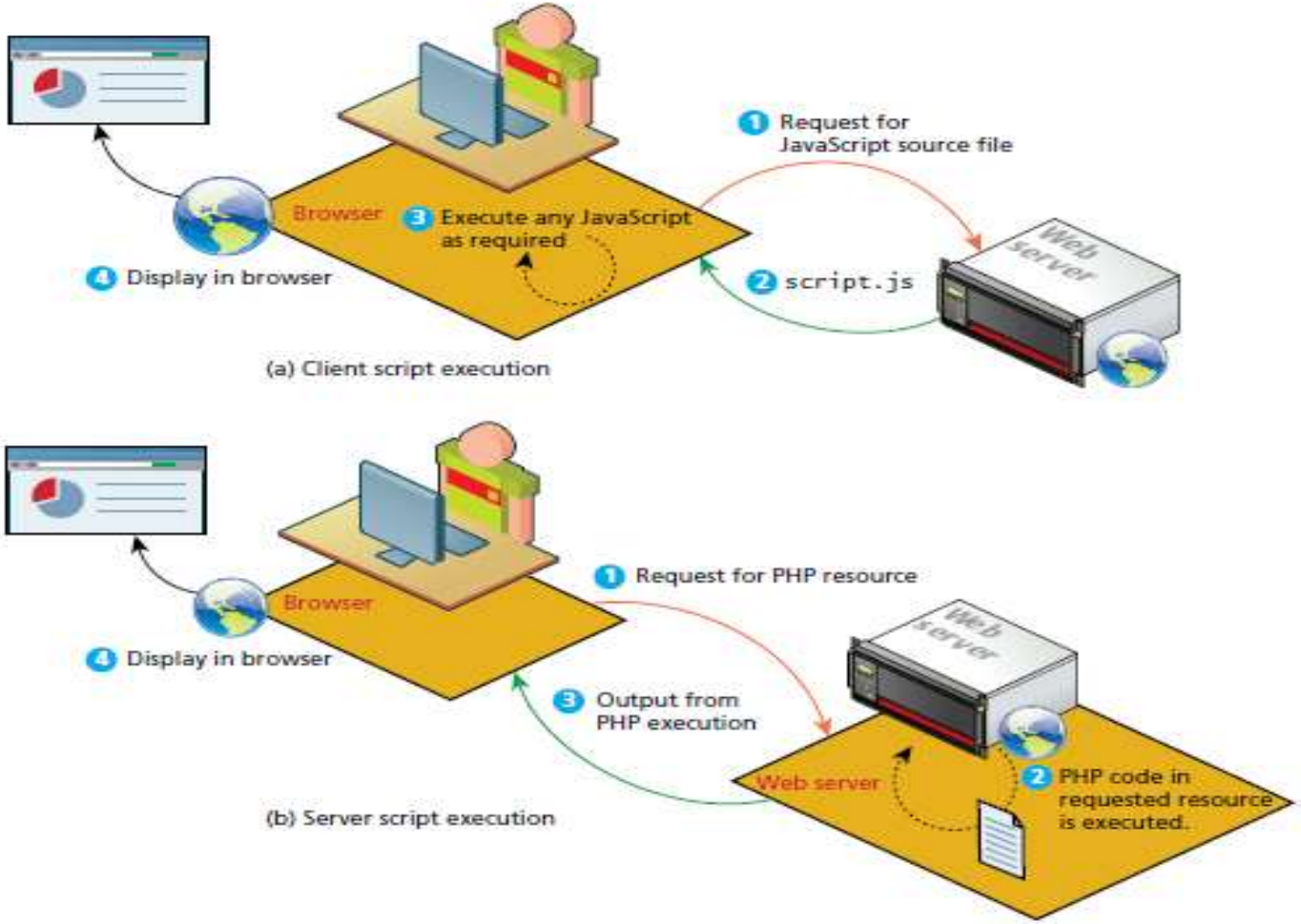


FIGURE Comparison of (a) client script execution and (b) server script execution

- A server-side script can access any resources made available to it by the server.
- These resources can be categorized as data storage resources, web services, and software applications

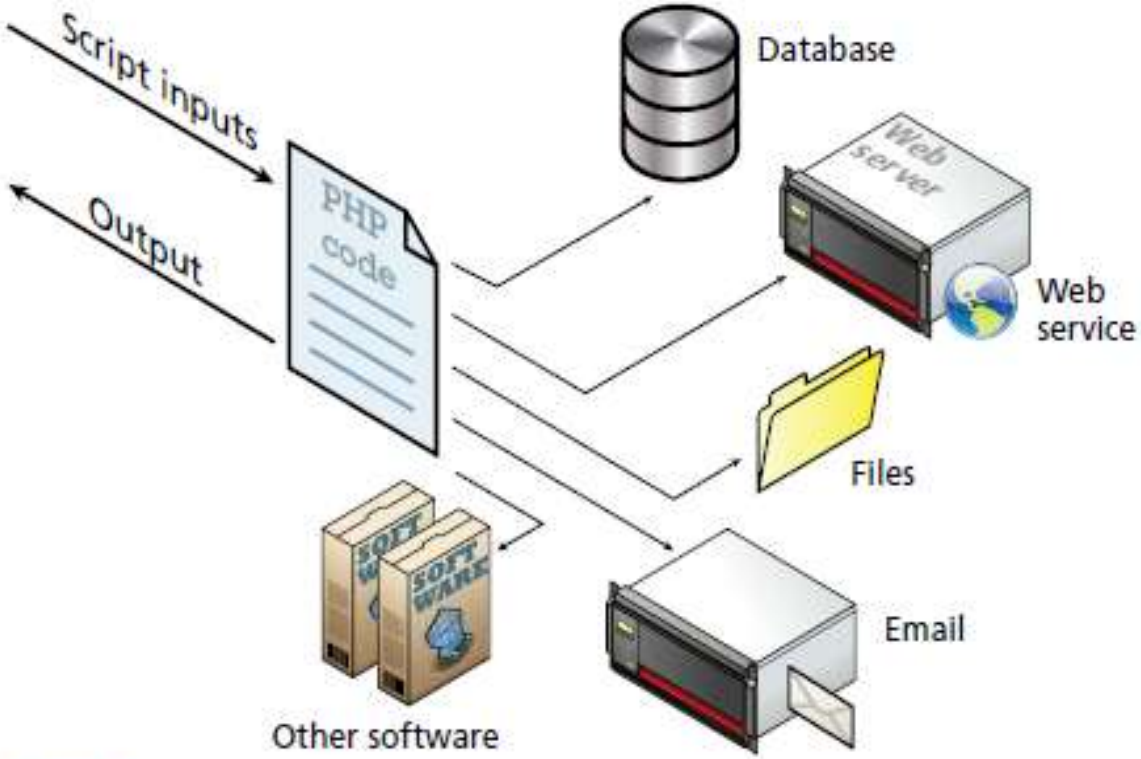


FIGURE Server scripts have access to many resources.

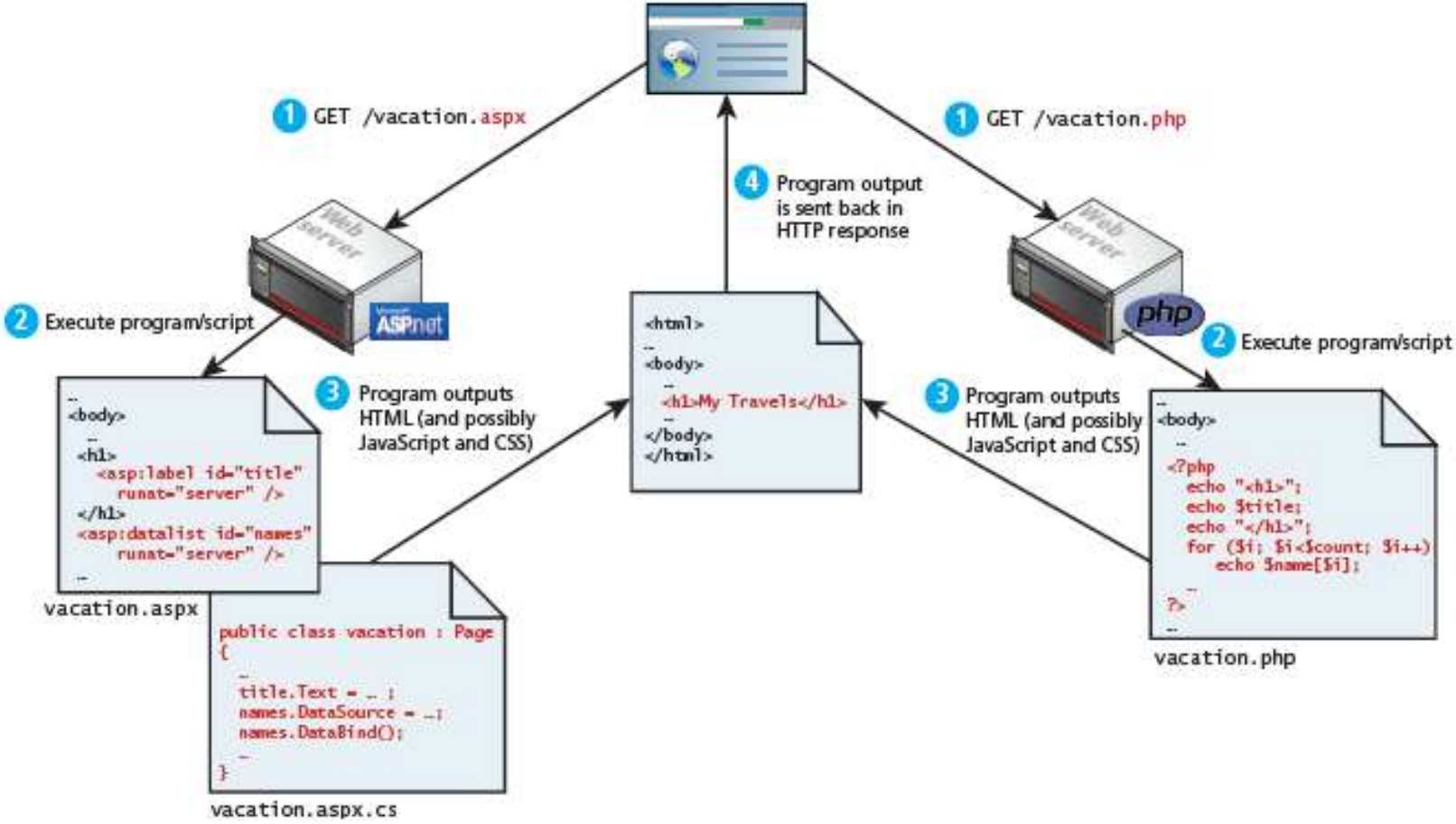


FIGURE Web development technologies

- ASP (Active Server Pages). This was Microsoft's first server-side technology (also called ASP Classic). Like PHP, ASP code (using the VBScript programming language) can be embedded within the HTML; though it supported classes and *some* object-oriented features, most developers did not make use of these features. ASP programming code is interpreted at run time, hence it can be slow in comparison to other technologies.
- ASP.NET. This replaced Microsoft's older ASP technology. ASP.NET is part of Microsoft's .NET Framework and can use any .NET programming language (though C# is the most commonly used). ASP.NET uses an explicitly object-oriented approach that typically takes longer to learn than ASP or PHP, and is often used in larger corporate web application systems. It also uses special markup called web server controls that encapsulate common web functionality such as database-driven lists, form validation, and user registration wizards. A recent extension called ASP.NET MVC makes use of the Model-View-Controller design pattern (this pattern will be covered in Chapter 14). ASP.NET pages are compiled into an intermediary file format called MSIL that is analogous to Java's byte-code. ASP.NET then uses a JIT (Just-In-Time) compiler to compile the MSIL into machine executable code so its performance can be excellent. However, ASP.NET is essentially limited to Windows servers.

- JSP (Java Server Pages). JSP uses Java as its programming language and like ASP.NET it uses an explicit object-oriented approach and is used in large enterprise web systems and is integrated into the J2EE environment. Since JSP uses the Java Runtime Engine, it also uses a JIT compiler for fast execution time and is cross-platform. While JSP's usage in the web as a whole is small, it has a substantial market share in the intranet environment, as well as with very large and busy sites.
- Node.js. This is a more recent server environment that uses JavaScript on the server side, thus allowing developers already familiar with JavaScript to use just a single language for both client-side and server-side development. Unlike the other development technologies listed here, node.js is also its own web server software, thus eliminating the need for Apache, IIS, or some other web server software.

- Perl. Until the development and popularization of ASP, PHP, and JSP, Perl was the language typically used for early server-side web development. As a language, it excels in the manipulation of text. It was commonly used in conjunction with the Common Gateway Interface (CGI), an early standard API for communication between applications and web server software.
- PHP. Like ASP, PHP is a dynamically typed language that can be embedded directly within the HTML, though it now supports most common object-oriented features, such as classes and inheritance. By default, PHP pages are compiled into an intermediary representation called opcodes that are analogous to Java's byte-code or the .NET Framework's MSIL. Originally, PHP stood for *personal home pages*, although it now is a recursive acronym that means *PHP: Hypertext Processor*.

- Python. This terse, object-oriented programming language has many uses, including being used to create web applications. It is also used in a variety of web development frameworks such as Django and Pyramid.
- Ruby on Rails. This is a web development framework that uses the Ruby programming language. Like ASP.NET and JSP, Ruby on Rails emphasizes the use of common software development approaches, in particular the MVC design pattern. It integrates features such as templates and engines that aim to reduce the amount of development work required in the creation of a new site.

All of these technologies share one thing in common: using programming logic, they generate HTML and possibly CSS and JavaScript on the server and send it back to the requesting browser, as shown in Figure .

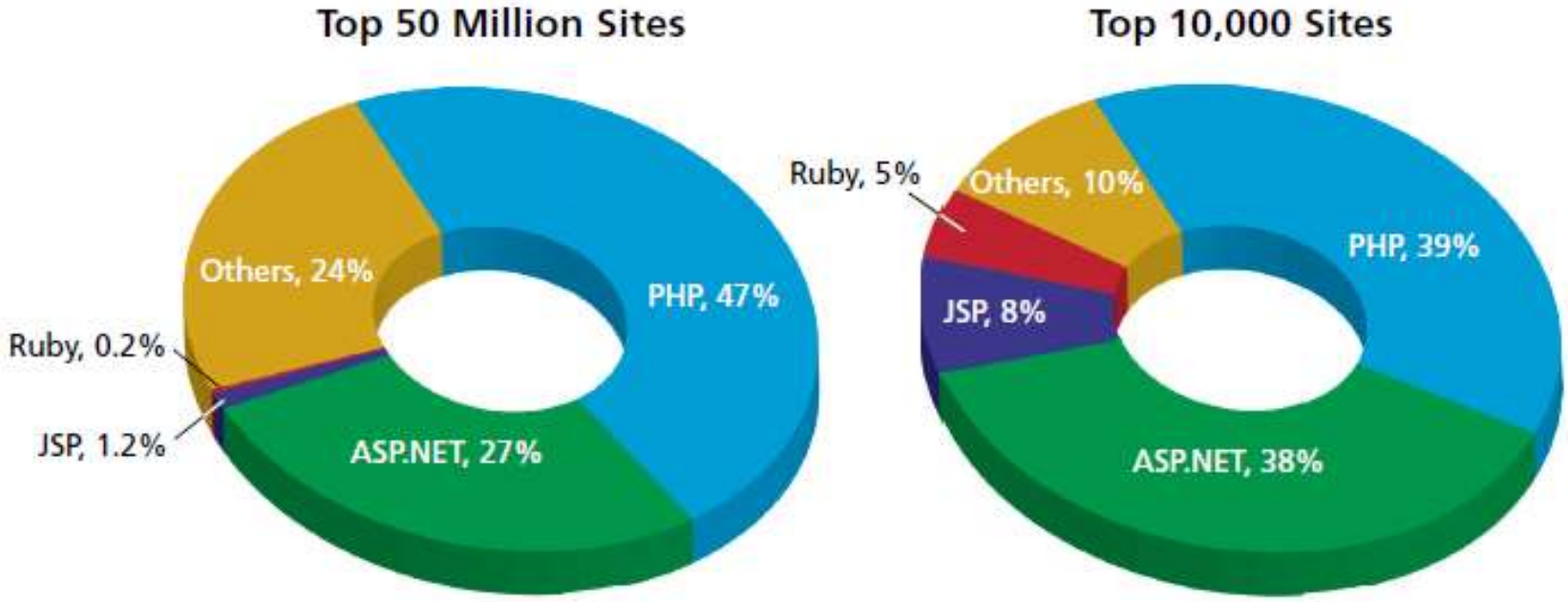


FIGURE 1 Market share of web development environments (data courtesy of BuiltWith.com)

```
<?php
$user = "Randy";
?>
<!DOCTYPE html>
<html>
<body>
<h1>Welcome <?php echo $user; ?></h1>
<p>
The server time is
<?php
echo "<strong>";
echo date("H:i:s");
echo "</strong>";
?>
</p>
</body>
</html>
```

LISTING PHP tags

```
<!DOCTYPE html>
<html>
<body>
<h1>Welcome Randy</h1>
<p>
The server time is <strong>02:59:09</strong>
</p>
</body>
</html>
```

LISTING HTML Output for the listing above

What is PHP?

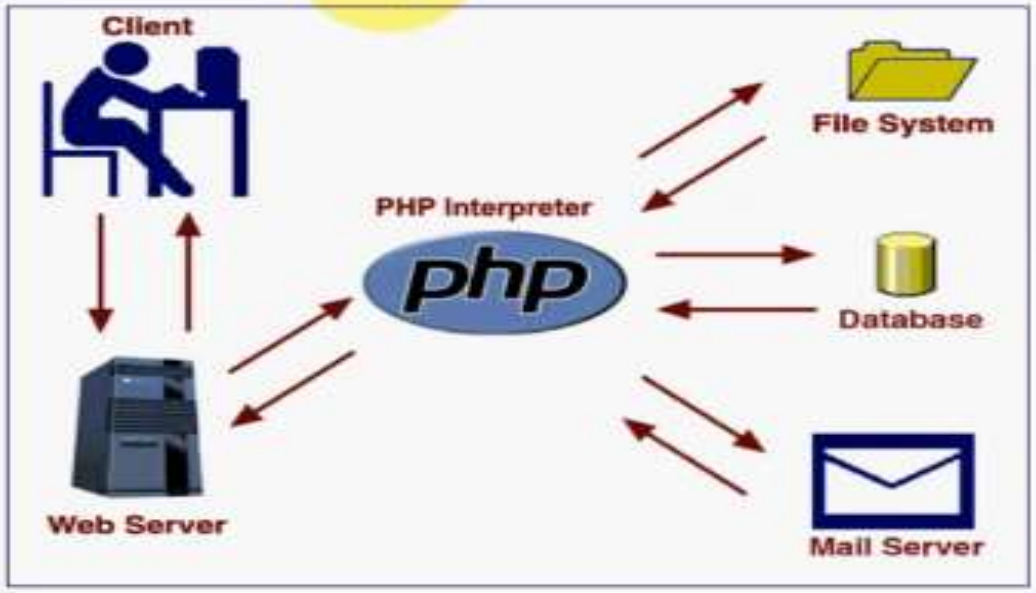
- PHP == 'Hypertext Preprocessor'
- Open-source, server-side scripting language
- Used to generate dynamic web-pages
- PHP scripts reside between reserved PHP tags
  - This allows the programmer to embed PHP scripts within HTML pages

What is PHP?

- Interpreted language, scripts are parsed at run-time rather than compiled beforehand
- Executed on the server-side
- Source-code not visible by client
  - 'View Source' in browsers does not display the PHP code
- Various built-in functions allow for fast development
- Compatible with many popular databases

- Goto php.net for detailed information about php

How PHP Works ?



Programming languages used in most popular websites

Website	Popularity (unique visitors) [1][unreliable source?]	Front-end (Client-side)	Back-end (Server-side)	Database	Notes
Google.com[2]	1,000,000,000	JavaScript	C, C++, Go,[3] Java, Python	BigTable[4]	The most used search engine in the wo
Facebook.com	880,000,000	JavaScript	Hack, PHP, C++, Java, Python, Erlang, D,[5] Xhp[6]	MySQL,[7] HBase	The most visited social networking site
YouTube.com	800,000,000	Flash, JavaScript	C/C++, Python, Java[8]	MySQL, BigTable	The most visited video sharing site
Yahoo	600,000,000	JavaScript	PHP	MySQL	Yahoo is presently[when?] transitioning t node.js[citation needed]
Live.com	490,000,000	JavaScript	ASP.NET	Microsoft SQL Server	
MSN.com	440,000,000	JavaScript	ASP.NET	Microsoft SQL Server	An email client, for simple use. Mostly k "messenger"
Wikipedia.org	410,000,000	JavaScript	PHP	MySQL, MariaDB[9]	"MediaWiki" is programmed in PHP; fre encyclopedia
Blogger	340,000,000	JavaScript	Python	BigTable	
Bing	230,000,000	JavaScript	ASP.NET	Microsoft SQL Server	
Twitter.com	160,000,000	JavaScript	C++, Java, Scala, Ruby on Rails[10]	MySQL[11]	140 characters social network
Wordpress.com	130,000,000	JavaScript	PHP	MySQL	
Amazon.com	110,000,000	JavaScript	Java, J2EE, C++, Perl		Popular internet shopping site
eBay.com	88,000,000	JavaScript	Java	Oracle Database	Online auction house

What does PHP code look like?

- Structurally similar to C/C++
- Supports procedural and object-oriented paradigm (to some degree)
- All PHP statements end with a semi-colon
- Each PHP script must be enclosed in the reserved PHP tag

```
<?php
...
?>
```


<?php
echo "Hello World" ;
?>

Comments in PHP

- Standard C, C++, and shell comment symbols

```
// C++ and Java-style comment

# Shell-style comments

/* C-style comments
     These can span multiple lines */
```

- **Single-line comments.** Lines that begin with a # are comment lines and will not be executed.
- **Multiline (block) comments.** Each PHP script and each function within it are ideal places to include a large comment block. These comments begin with a /* and encompass everything that is encountered until a closing */ tag is found. These tags cannot be nested.

  A comment block above a function or at the start of a file is a good place to write, in normal language, what this function does. By using the /** tag to open the comment instead of the standard /*, you are identifying blocks of comment that can later be parsed for inclusion in generated documents.
- **End-of-line comments.** Comments need not always be large blocks of natural language. Sometimes a variable needs a little blurb to tell the developer what it's for, or a complex portion of code needs a few comments to help the programmer understand the logic. Whenever // is encountered in code, everything up to the end of the line is considered a comment. These comments are sometimes preferable to the block comments because they do not interfere with one another, but are unable to span multiple lines of code.

```php
<?php
// this is a hello world code
#this is a hello world code
/*this is a hello world code*/
echo "Hello World" ;

?>
```

```php
<?php
// this is a hello world code
#this is a hello world code
/*this is a hello world code*/
echo "Hello <br> <b>World</b>" ;

?>
```

Variables in PHP

- PHP variables must begin with a "$" sign
- Case-sensitive ($Boo != $Boo != $BOo)
- Global and locally-scoped variables
  - Global variables can be used anywhere
  - Local variables restricted to a function or class
- Certain variable names reserved by PHP
  - Form variables ($_POST, $_GET)
  - Server variables ($_SERVER)
  - Etc.

Variable usage

```
<?php
$foo = 25;          // Numerical variable
$bar = "Hello";     // String variable

$foo = ($foo * 7);// Multiplies foo by 7
$bar = ($bar * 7);// Invalid expression
?>
```

```php
<?php
$value=25;
$name="ProgrammingKnowledge";

$value=($value*8);
?>
```

Data Type	Description
`Boolean`	A logical true or false value
`Integer`	Whole numbers
`Float`	Decimal numbers
`String`	Letters
`Array`	A collection of data of any type (covered in the next chapter)
`Object`	Instances of classes

TABLE PHP Data Types

PHP allows variable names to also be specified at run time. This type of variable is sometimes referred to as a "variable variable" and can be convenient at times. For instance, imagine you have a set of variables named as follows:

```
<?php

$artist1 = "picasso";
$artist2 = "raphael";
$artist3 = "cezanne";
$artist4 = "rembrandt";
$artist5 = "giotto";

?>
```

If you wanted to output each of these variables within a loop, you can do so by programmatically constructing the variable name within curly brackets, as shown in the following loop:

```
for ($i = 1; $i <= 5; $i++) {
    echo ${"artist". $i};
    echo "<br/>";
}
```

Echo

- The PHP command '**echo**' is used to output the parameters passed to it
  - The typical usage for this is to send data to the client's web-browser
- Syntax
  - void **echo** (string arg***1*** [, string arg***n***...])
  - In practice, arguments are not passed in parentheses since **echo** is a language construct rather than an actual function

```php
<?php
$value=25;
$name="ProgrammingKnowledge";

$value=($value*8);

echo "$name";
echo "$name";
?>
```

Writing to Output

Remember that PHP pages are programs that output HTML. To output something that will be seen by the browser, you can use the `echo()` function.

```
echo ("hello");
```

There is also an equivalent shortcut version that does not require the parentheses.

```
echo "hello";
```

Strings can easily be appended together using the concatenate operator, which is the period (.) symbol. Consider the following code:

```
$username = "Ricardo";
echo "Hello". $username;
```

```php
<?php

$firstName = "Pablo";
$lastName = "Picasso";

/*
  Example one:
  These two lines are equivalent. Notice that you can reference PHP
  variables within a string literal defined with double quotes.

  The resulting output for both lines is:

    <em>Pablo Picasso</em>

*/
echo "<em>" . $firstName . " ". $lastName. "</em>";
echo "<em> $firstName $lastName </em>";
```

```
/*
  Example two:
  These two lines are also equivalent. Notice that you can use
  either the single quote symbol or double quote symbol for string
  literals.
*/
echo "<h1>";
echo '<h1>';

/*
  Example three:
  These two lines are also equivalent. In the second example, the
  escape character (the backslash) is used to embed a double quote
  within a string literal defined within double quotes.
*/
echo '<img src="23.jpg" >';
echo "<img src=\"23.jpg\" >";

?>
```

LISTING PHP quote usage and concatenation approaches

```php
<?php

$id = 23;
$firstName = "Pablo";
$lastName = "Picasso";

echo "<img src='23.jpg' alt='". $firstName . " ". $lastName . "' >";
echo "<img src='$id.jpg' alt='$firstName  $lastName' >";
echo "<img src=\"$id.jpg\" alt=\"$firstName  $lastName\" >";
echo '<img src="' . $id . '.jpg" alt="' . $firstName . ' ' .
     $lastName . '" >';
echo '<a href="artist.php?id=' . $id .'">' . $firstName . ' ' .
     $lastName . '</a>';


?>
```

LISTING More complicated concatenation examples


1
echo "<img src='23.jpg' alt='" . $firstName . " " . $lastName . "' >";
outputs
<img src='23.jpg' alt='Pablo Picasso' >
2
echo "<img src='$id.jpg' alt='$firstName $lastName' >";
<img src='23.jpg' alt='Pablo Picasso' >
3
echo "<img src=\"$id.jpg\" alt=\"$firstName $lastName\" >";
<img src="23.jpg" alt="Pablo Picasso" >

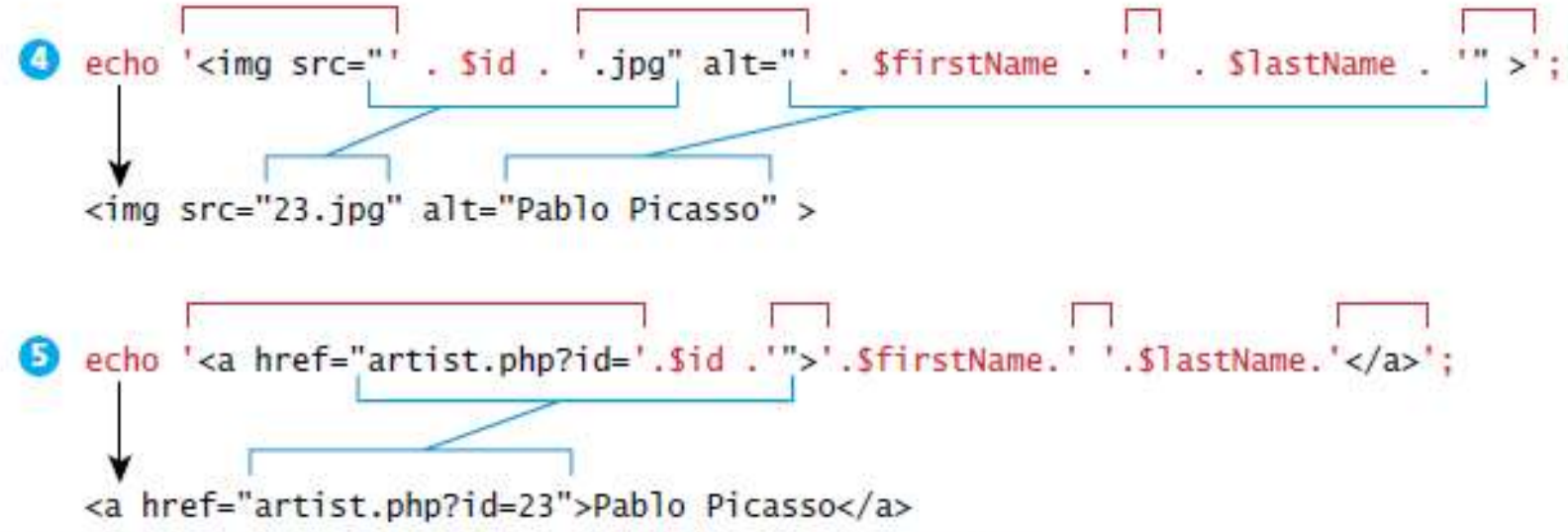


FIGURE More complicated concatenation examples explained

printf

As the examples in Listing 8.6 illustrate, while `echo` is quite simple, more complex output can get confusing. As an alternative, you can use the `printf()` function. This function is derived from the same-named function in the C programming language and includes variations to print to string and files (`sprintf`, `fprintf`). The function takes at least one parameter, which is a string, and that string optionally references parameters, which are then integrated into the first string by placeholder substitution.[4] The `printf()` function also allows a developer to apply special formatting, for instance, specific date/time formats or number of decimal places.

Figure illustrates the relationship between the first parameter string, its placeholders and subsequent parameters, precision, and output.

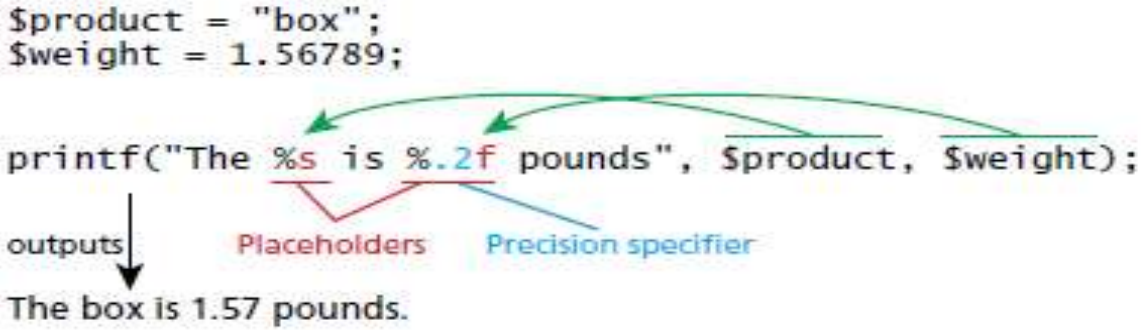


FIGURE Illustration of components in a printf statement and output

Each placeholder requires the percent (%) symbol in the first parameter string followed by a type specifier. Common type specifiers are `b` for binary, `d` for signed integer, `f` for float, `o` for octal, and `x` for hexadecimal. Precision is achieved in the string with a period (`.`) followed by a number specifying how many digits should be displayed for floating-point numbers.

Program controls

if . . . else

The syntax for conditionals in PHP is almost identical to that of JavaScript. In this syntax the condition to test is contained within () brackets with the body contained in {} blocks. Optional `else if` statements can follow, with an `else` ending the branch. Listing uses a conditional to set a greeting variable, depending on the hour of the day.

```
// if statement with condition
if ( $hourOfDay > 6 && $hourOfDay < 12 ) {
   $greeting = "Good Morning";
}
else if ($hourOfDay == 12) {   // optional else if
   $greeting = "Good Noon Time";
}
else {                         // optional else branch
   $greeting = "Good Afternoon or Evening";
}
```

LISTING Conditional statement using if . . . else

```php
<?php if ($userStatus == "loggedin") {  ?>
   <a href="account.php">Account</a>
   <a href="logout.php">Logout</a>
<?php }  else { ?>
   <a href="login.php">Login</a>
   <a href="register.php">Register</a>
<?php } ?>


<?php
   // equivalent to the above conditional
   if ($userStatus == "loggedin") {
      echo '<a href="account.php">Account</a> ';
      echo '<a href="logout.php">Logout</a>';
   }
   else {
      echo '<a href="login.php">Login</a> ';
      echo '<a href="register.php">Register</a>';
   }
?>
```

LISTING Combining PHP and HTML in the same script

Switch

```
switch ($artType) {
   case "PT":
      $output = "Painting";
      break;
   case "SC":
      $output = "Sculpture";
      break;
   default:
      $output = "Other";
}

// equivalent
if ($artType == "PT")
   $output = "Painting";
else if ($artType == "SC")
   $output = "Sculpture";
else
   $output = "Other";
```

LISTING Conditional statement using switch

NOTE

Be careful with mixing types when using the `switch` statement: if the variable being compared has an integer value, but a case value is a string, then there will be type conversions that will create some unexpected results. For instance, the following example will output `"Painting"` because it first converts the `"PT"` to an integer (since `$code` currently contains an integer value), which is equal to the integer 0 (zero).

```
$code = 0;
switch($code) {
  case "PT":
    echo "Painting";
    break;
  case 1:
    echo "Sculpture";
    break;
  default:
    echo "Other";
}
```

While do while

```php
$count = 0;
while ($count < 10)
{
    echo $count;
    $count++;
}

$count = 0;
do
{
    echo $count;
    $count++;
} while ($count < 10);
```

LISTING . while loops

for

```
for ($count=0; $count < 10; $count++)
{
   echo $count;
}
```

LISTING for loops

Alternate Syntax for Control Structures

PHP has an alternative syntax for most of its control structures (namely, the `if`, `while`, `for`, `foreach`, and `switch` statements). In this alternate syntax (shown in Listing), the colon (`:`) replaces the opening curly bracket, while the closing brace is replaced with `endif;`, `endwhile;`, `endfor;`, `endforeach;`, or `endswitch;`. While this may seem strange and unnecessary, it can actually improve the readability of your PHP code when it intermixes PHP and markup within a control structure, as was seen in Listing .

```
<?php if ($userStatus == "loggedin") :  ?>
   <a href="account.php">Account</a>
   <a href="logout.php">Logout</a>
<?php else : ?>
   <a href="login.php">Login</a>
   <a href="register.php">Register</a>
<?php endif; ?>
```

LISTING : Alternate syntax for control structures

Include Files

PHP does have one important facility that is generally unlike other nonweb programming languages, namely the ability to include or insert content from one file into another.[5] Almost every PHP page beyond simple practice exercises makes use of this include facility. Include files provide a mechanism for reusing both markup and PHP code, as shown in Figure 8.14.

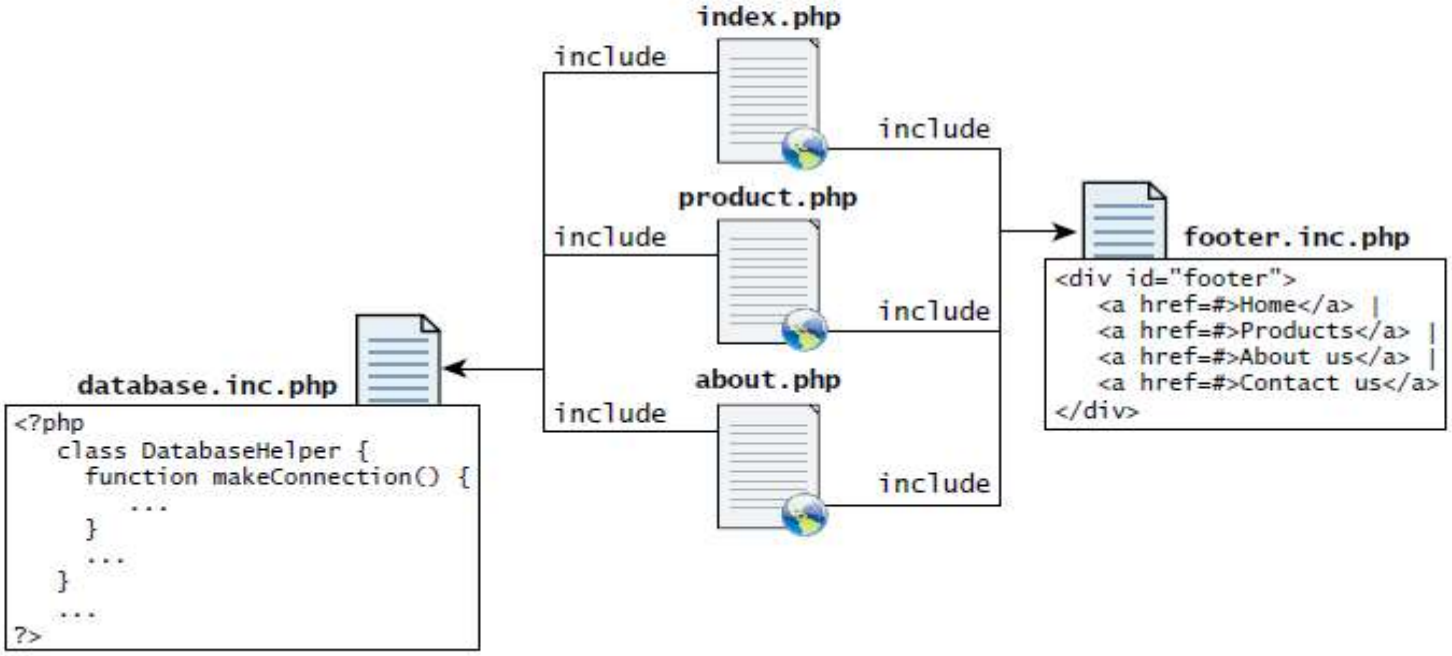


FIGURE Include files

function

```php
/**
* This function returns a nicely formatted string using the current
* system time.
*/
function getNiceTime() {
   return date("H:i:s");
}
```

LISTING The definition of a function to return the current time as a string

```php
/**
* This function outputs the footer menu
*/
function outputFooterMenu() {
   echo '<div id="footer">';
   echo '<a href=#>Home</a> | <a href=#>Products</a> | ';
   echo '<a href=#>About us</a> | <a href=#>Contact us</a>';
   echo '</div>';
}
```

LISTING The definition of a function without a return value

Calling a Function

Now that you have defined a function, you are able to use it whenever you want to. To call a function you must use its name with the () brackets. Since `getNiceTime()` returns a string, you can assign that return value to a variable, or echo that return value directly, as shown below.

```
$output = getNiceTime();
echo getNiceTime();
```

If the function doesn't return a value, you can just call the function:

```
outputFooterMenu();
```

parameters

Parameters

It is more common to define functions with parameters, since functions are more powerful and reusable when their output depends on the input they get. Parameters are the mechanism by which values are passed into functions, and there are some complexities that allow us to have multiple parameters, default values, and to pass objects by reference instead of value.

```php
/**
* This function returns a nicely formatted string using the current
* system time. The showSeconds parameter controls whether or not to
* include the seconds in the returned string.
*/
function getNiceTime($showSeconds) {
  if ($showSeconds==true)
    return date("H:i:s");
  else
    return date("H:i");
}
```

LISTING : A function to return the current time as a string with an integer parameter

Thus to call our function, you can now do it in two ways:

```
echo getNiceTime(1);    // this will print seconds
echo getNiceTime(0);   //  will not print seconds
```

```php
/**
* This function returns a nicely formatted string using the current
* system time. The showSeconds parameter controls whether or not
* to show the seconds.
*/
function getNiceTime($showSeconds=1){
   if ($showSeconds==true)
      return date("H:i:s");
   else
      return date("H:i");
}
```

LISTING . A function to return the current time with a parameter that includes a default

Now if you were to call the function with no values, the $showSeconds parameter would take on the default value, which we have set to 1, and return the string with seconds. If you do include a value in your function call, the default will be overridden by whatever that value was. Either way you now have a single function that can be called with or without values passed.

```php
function changeParameter($arg) {
   $arg += 300;
   echo "<br/>arg=" . $arg;
}

$initial = 15;
echo "<br/>initial=" . $initial;    // output: initial=15
changeParameter($initial);          // output: arg=315
echo "<br/>initial=" . $initial;    // output: initial=15
```

LISTING Passing a parameter by value

```php
function changeParameter(&$arg) {
   $arg += 300;
   echo "<br/>arg=". $arg;
}

$initial = 15;
echo "<br/>initial=" . $initial;    // output: initial=15
changeParameter($initial);          // output: arg=315
echo "<br/>initial=" . $initial;    // output: initial=315
```

LISTING Passing a parameter by reference

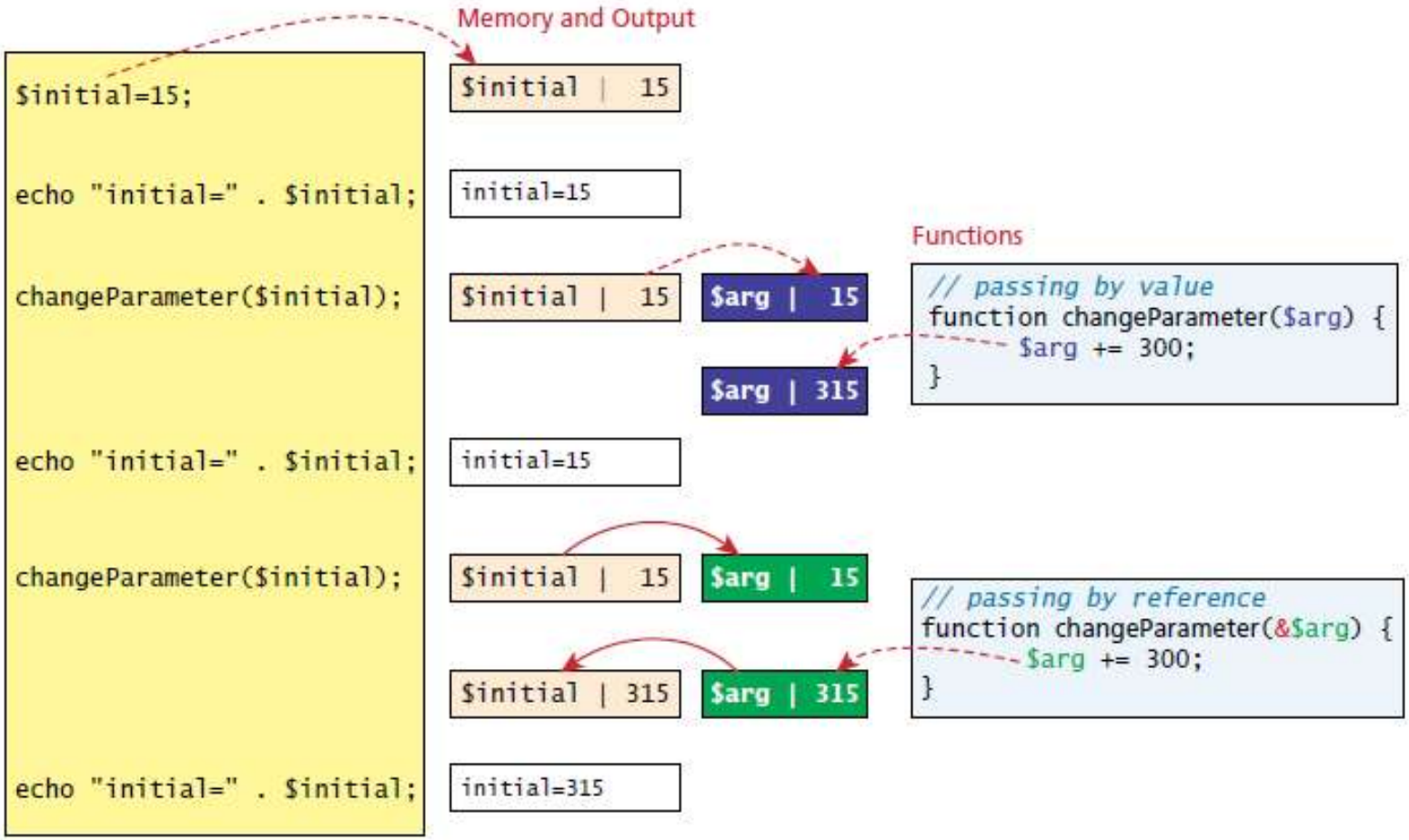


FIGURE Pass by value versus pass by reference

Variable scope within functions

```php
$count= 56;

function testScope() {
    echo $count;     // outputs 0 or generates run-time warning/error
}
testScope();
echo $count;         // outputs 56
```

```php
$count= 56;

function testScope() {
    global $count;
    echo $count;     // outputs 56
}

testScope();
echo $count;         // outputs 56
```

LISTING . Using the global keyword

Review Questions

1. In the LAMP stack, what software is responsible for responding to HTTP requests?
2. Describe one alternative to the LAMP stack.
3. Identify and briefly describe at least four different server-side development technologies.
4. Describe the difference between the multi-threaded and multi-process setup of PHP in Apache.
5. Describe the steps taken by the Zend Engine when it receives a PHP request.
6. What does it mean that PHP is dynamically typed?
7. What are server-side include files? Why are they important in PHP?
8. Can we have two functions with the same name in PHP? Why or why not?
9. How do we define default function parameters in PHP?
10. How are parameters passed by reference different than those passed by value?

Arrays

For some PHP developers, arrays are easy to understand, but for others they are a challenge. To help visualize what is happening, one should become familiar with the concept of keys and associated values. Figure illustrates a PHP array with five strings containing day abbreviations.

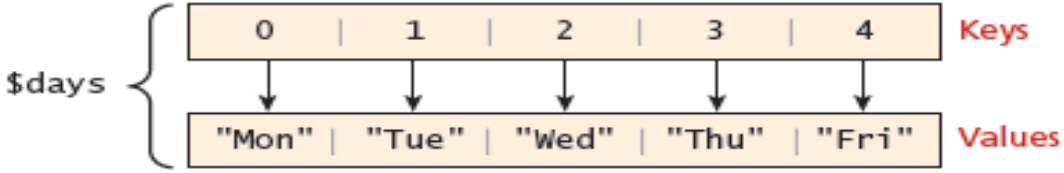


FIGURE . Visualization of a key-value array

Array keys in most programming languages are limited to integers, start at 0, and go up by 1. In PHP, keys *must* be either integers or strings and need not be sequential. This means you cannot use an array or object as a key (doing so will generate an error).

One should be especially careful about mixing the types of the keys for an array since PHP performs cast operations on the keys that are not integers or strings. You cannot have key "1" distinct from key 1 or 1.5, since all three will be cast to the integer key 1.

Array values, unlike keys, are not restricted to integers and strings. They can be any object, type, or primitive supported in PHP. You can even have objects of your own types, so long as the keys in the array are integers and strings.

Defining and Accessing an Array

Let us begin by considering the simplest array, which associates each value inside of it with an integer index (starting at 0). The following declares an empty array named `days`:

```
$days = array();
```

To define the contents of an array as strings for the days of the week as shown in Figure , you declare it with a comma-delimited list of values inside the ( ) braces using either of two following syntaxes:

```
$days = array("Mon","Tue","Wed","Thu","Fri");
$days = ["Mon","Tue","Wed","Thu","Fri"];     // alternate syntax
```

In these examples, because no keys are explicitly defined for the array, the default key values are 0, 1, 2, . . . , n. Notice that you do not have to provide a size for the array: arrays are dynamically sized as elements are added to them.

Elements within a PHP array are accessed in a manner similar to other programming languages, that is, using the familiar square bracket notation. The code example below echoes the value of our `$days` array for the `key=1`, which results in output of `Tue`.

```
echo "Value at index 1 is ". $days[1];    // index starts at zero
```

You could also define the array elements individually using this same square bracket notation:

```
$days = array();
$days[0] = "Mon";
$days[1] = "Tue";
$days[2] = "Wed";

// also alternate approach
$daysB = array();
$daysB[] = "Mon";
$daysB[] = "Tue";
$daysB[] = "Wed";
```

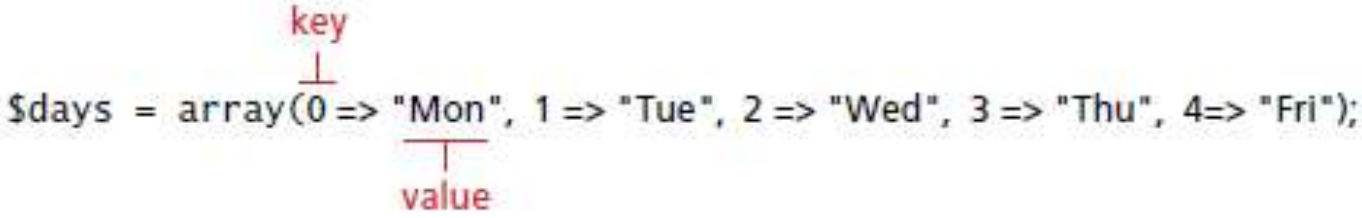
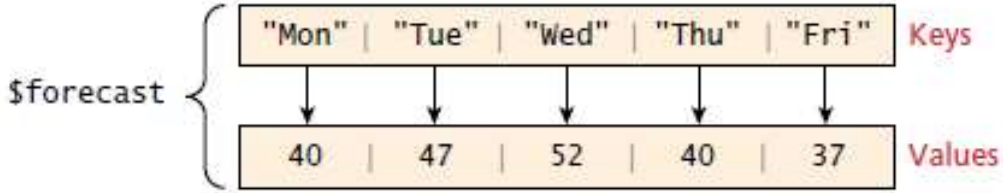


FIGURE Explicitly assigning keys to array elements

```
$forecast = array("Mon" => 40, "Tue" => 47, "Wed" => 52, "Thu" => 40, "Fri" => 37);
```

key

value




```
echo $forecast["Tue"];  // outputs 47
echo $forecast["Thu"];  // outputs 40
```

FIGURE Array with strings as keys and integers as values

Multidimensional Arrays

PHP also supports multidimensional arrays. Recall that the values for an array can be any PHP object, which includes other arrays. Listing ' illustrates the creation of two different multidimensional arrays (each one contains two dimensions).

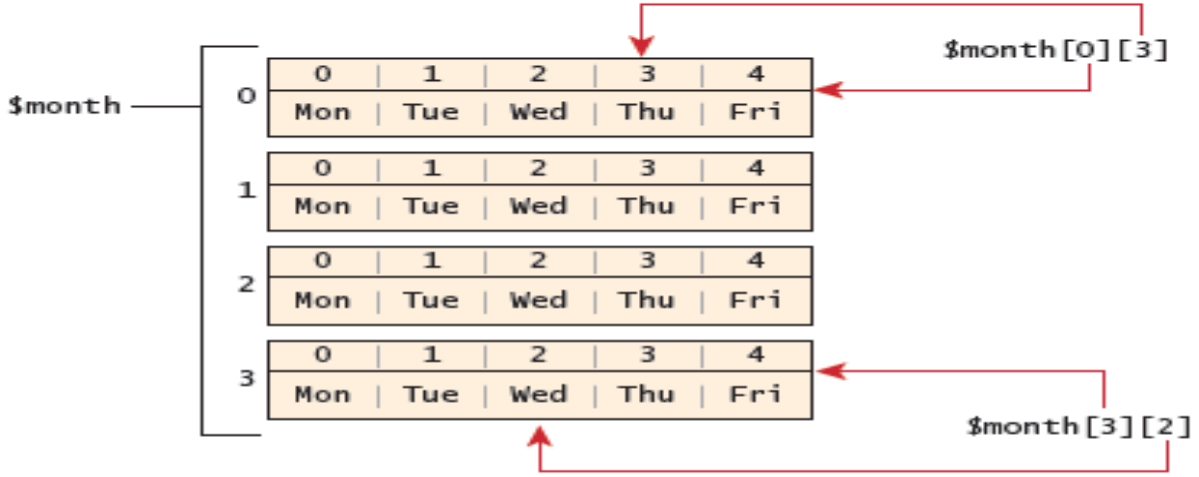
```
$month = array
  (
  array("Mon","Tue","Wed","Thu","Fri"),
  array("Mon","Tue","Wed","Thu","Fri"),
  array("Mon","Tue","Wed","Thu","Fri"),
  array("Mon","Tue","Wed","Thu","Fri")
  );

echo $month[0][3];    // outputs Thu
```

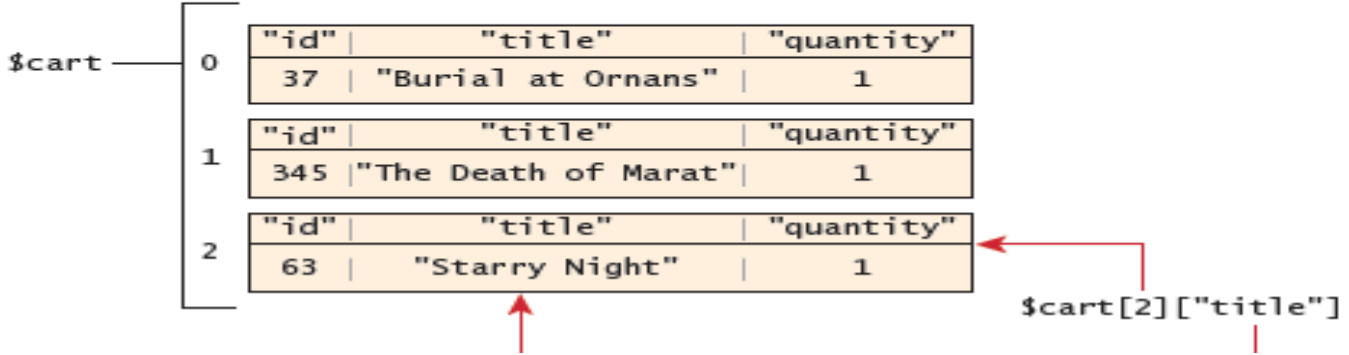
```
$cart = array();
$cart[] = array("id" => 37, "title" => "Burial at Ornans",
                "quantity" => 1);
$cart[] = array("id" => 345, "title" => "The Death of Marat",
                "quantity" => 1);
$cart[] = array("id" => 63, "title" => "Starry Night", "quantity" => 1);

echo $cart[2]["title"];  // outputs Starry Night
```

LISTING Multidimensional arrays

Iterating through an Array

One of the most common programming tasks that you will perform with an array is to iterate through its contents. Listing 9.2 illustrates how to iterate and output the





```php
// while loop
$i=0;
while ($i < count($days)) {
    echo $days[$i] . "<br>";
    $i++;
}


// do while loop
$i=0;
do {
    echo $days[$i] . "<br>";
    $i++;
} while ($i < count($days));


// for loop
for ($i=0; $i<count($days); $i++) {
    echo $days[$i] . "<br>";
}
```

LISTING . Iterating through an array using while, do while, and for loops

Checking If a Value Exists

Since array keys need not be sequential, and need not be integers, you may run into a scenario where you want to check if a value has been set for a particular key. As with undefined null variables, values for keys that do not exist are also undefined. To check if a value exists for a key, you can therefore use the `isset()` function, which returns true if a value has been set, and false otherwise. Listing . defines an array with noninteger indexes, and shows the result of asking `isset()` on several indexes.

```
$oddKeys = array (1 => "hello", 3 => "world", 5 => "!");
if (isset($oddKeys[0])) {
   // The code below will never be reached since $oddKeys[0] is not set!
   echo "there is something set for key 0";
}
if (isset($oddKeys[1])) {
   // This code will run since a key/value pair was defined for key 1
   echo "there is something set for key 1, namely ". $oddKeys[1];
}
```

LISTING . Illustrating nonsequential keys and usage of isset( )

Array Sorting

One of the major advantages of using a mature language like PHP is its built-in functions. There are many built-in sort functions, which sort by key or by value. To sort the `$days` array by its values you would simply use:

```
sort($days);
```

As the values are all strings, the resulting array would be:

```
Array ([0] => Fri [1] => Mon [2] => Sat [3] => Sun [4] => Thu
       [5] => Tue [6] => Wed)
```

However, such a sort loses the association between the values and the keys! A better sort, one that would have kept keys and values associated together, is:

```
asort($days);
```

The resulting array in this case is:

```
Array ([4] => Fri [0] => Mon [5] => Sat [6] => Sun [3] => Thu
       [1] => Tue [2] => Wed)
```

More Array Operations

In addition to the powerful sort functions, there are other convenient functions you can use on arrays. It does not make sense to reinvent the wheel when valid, efficient functions have already been written for you. While we will not go into detail about each one, here is a brief description of some key array functions:

- **`array_keys($someArray)`**: This method returns an indexed array with the values being the *keys* of `$someArray`.

  For example, `print_r(array_keys($days))` outputs

  ```
  Array ( [0] => 0 [1] => 1 [2] => 2 [3] => 3 [4] => 4 )
  ```

- **`array_values($someArray)`**: Complementing the above `array_keys()` function, this function returns an indexed array with the values being the *values* of `$someArray`.

  For example, `print_r(array_values($days))` outputs

  ```
  Array ( [0] => Mon [1] => Tue [2] => Wed [3] => Thu [4] => Fri )
  ```

- **`array_rand($someArray, $num=1)`**: Often in games or widgets you want to select a random element in an array. This function returns as many random keys as are requested. If you only want one, the key itself is returned; otherwise, an array of keys is returned.

  For example, `print_r(array_rand($days,2))` might output:

  ```
  Array (3, 0)
  ```

- **`array_reverse($someArray)`**: This method returns `$someArray` in reverse order. The passed `$someArray` is left untouched.

For example, `print_r(array_reverse($days))` outputs:

```
Array ( [0] => Fri [1] => Thu [2] => Wed [3] => Tue [4] => Mon )
```

- **`array_walk($someArray, $callback, $optionalParam)`**: This method is extremely powerful. It allows you to call a method (`$callback`), for each value in `$someArray`. The `$callback` function typically takes two parameters, the value first, and the key second. An example that simply prints the value of each element in the array is shown below.

```
$someA = array("hello", "world");
array_walk($someA, "doPrint");
function doPrint($value,$key){
  echo $key . ": " . $value;
}
```

- **`in_array($needle, $haystack)`**: This method lets you search array `$haystack` for a value (`$needle`). It returns `true` if it is found, and `false` otherwise.
- **`shuffle($someArray)`**: This method shuffles `$someArray`. Any existing keys are removed and `$someArray` is now an indexed array if it wasn't already.

Superglobal Arrays

PHP uses special predefined associative arrays called superglobal variables that allow the programmer to easily access HTTP headers, query string parameters, and other commonly needed information

Name	Description
$GLOBALS	Array for storing data that needs superglobal scope
$_COOKIES	Array of cookie data passed to page via HTTP request
$_ENV	Array of server environment data
$_FILES	Array of file items uploaded to the server
$_GET	Array of query string data passed to the server via the URL
$_POST	Array of query string data passed to the server via the HTTP header
$_REQUEST	Array containing the contents of $_GET, $_POST, and $_COOKIES
$_SESSION	Array that contains session data
$_SERVER	Array containing information about the request and the server

TABLE Sugerglobal Variables

$_GET and $_POST Superglobal Arrays

The $_GET and $_POST arrays are the most important superglobal variables in PHP since they allow the programmer to access data sent by the client in a query string.

an HTML form (or an HTML link) allows a client to send data to the server. That data is formatted such that each value is associated with a name defined in the form. If the form was submitted using an HTTP GET request, then the resulting URL will contain the data in the query string. PHP will populate the superglobal $_GET array using the contents of this query string in the URL.

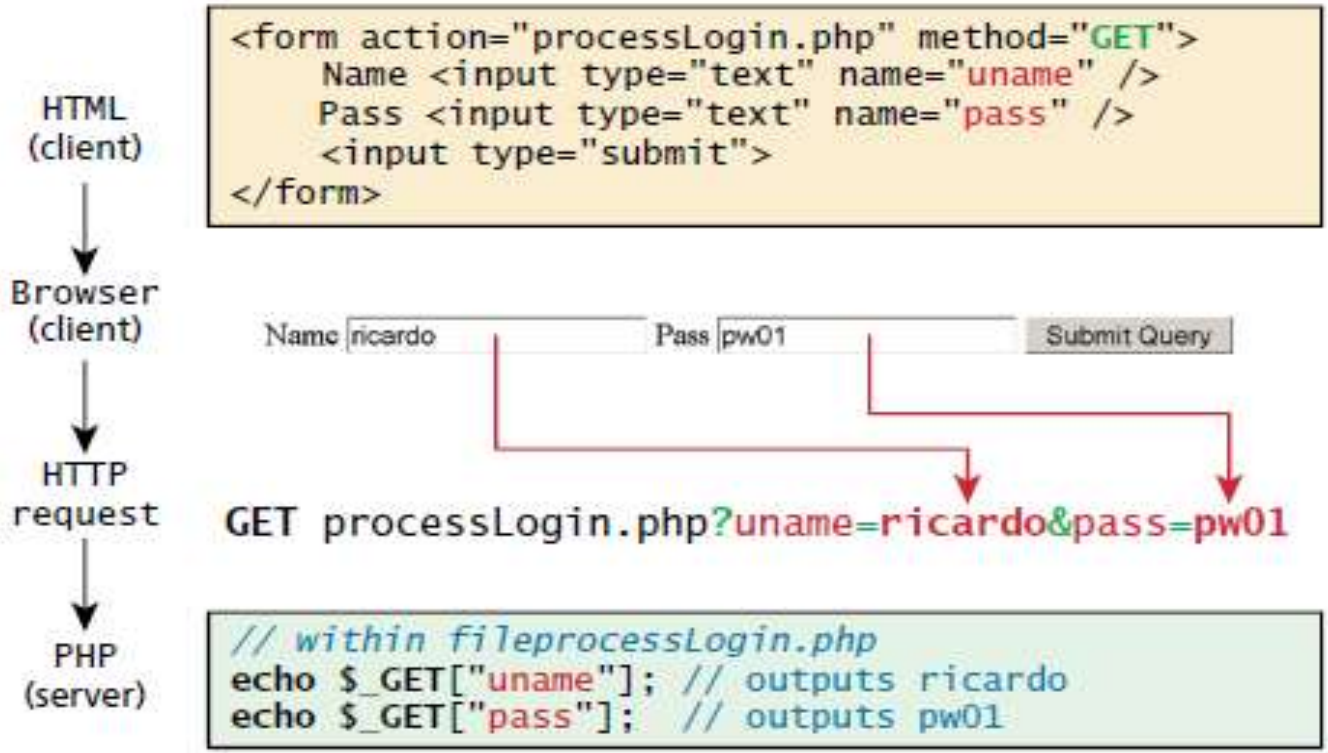


FIGURE Illustration of flow from HTML, to request, to PHP's $_GET array

If the form was sent using HTTP POST, then the values would not be visible in the URL, but will be sent through HTTP POST request body. From the PHP programmer's perspective, almost nothing changes from a GET data post except that those values and keys are now stored in the $_POST array. This mechanism greatly simplifies accessing the data posted by the user, since you need not parse the query string or the POST request headers.

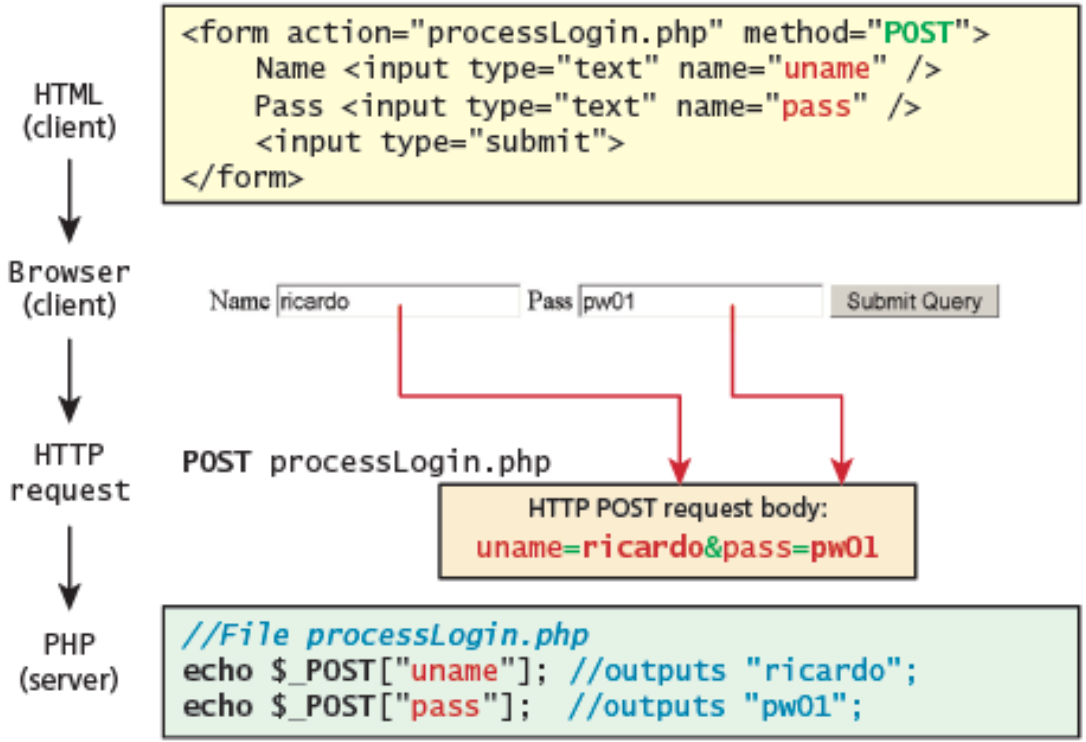


FIGURE Data flow from HTML form through HTTP request to PHP's $_POST array

$_SERVER Array

The $_SERVER associative array contains a variety of information. It contains some of the information contained within HTTP request headers sent by the client. It also contains many configuration options for PHP itself, as shown in Figure . . .

To use the $_SERVER array, you simply refer to the relevant case-sensitive key name:

```
echo $_SERVER["SERVER_NAME"] . "<br/>";
echo $_SERVER["SERVER_SOFTWARE"] . "<br/>";
echo $_SERVER["REMOTE_ADDR"] . "<br/>";
```

It is worth noting that because the entries in this array are created by the web server, not every key listed in the PHP documentation will necessarily be available. A complete list of keys contained within this array is listed in the online PHP documentation, but we will cover some of the critical ones here. They can be classified into keys containing request header information and keys with information about the server settings (which is often configured in the php.ini file).

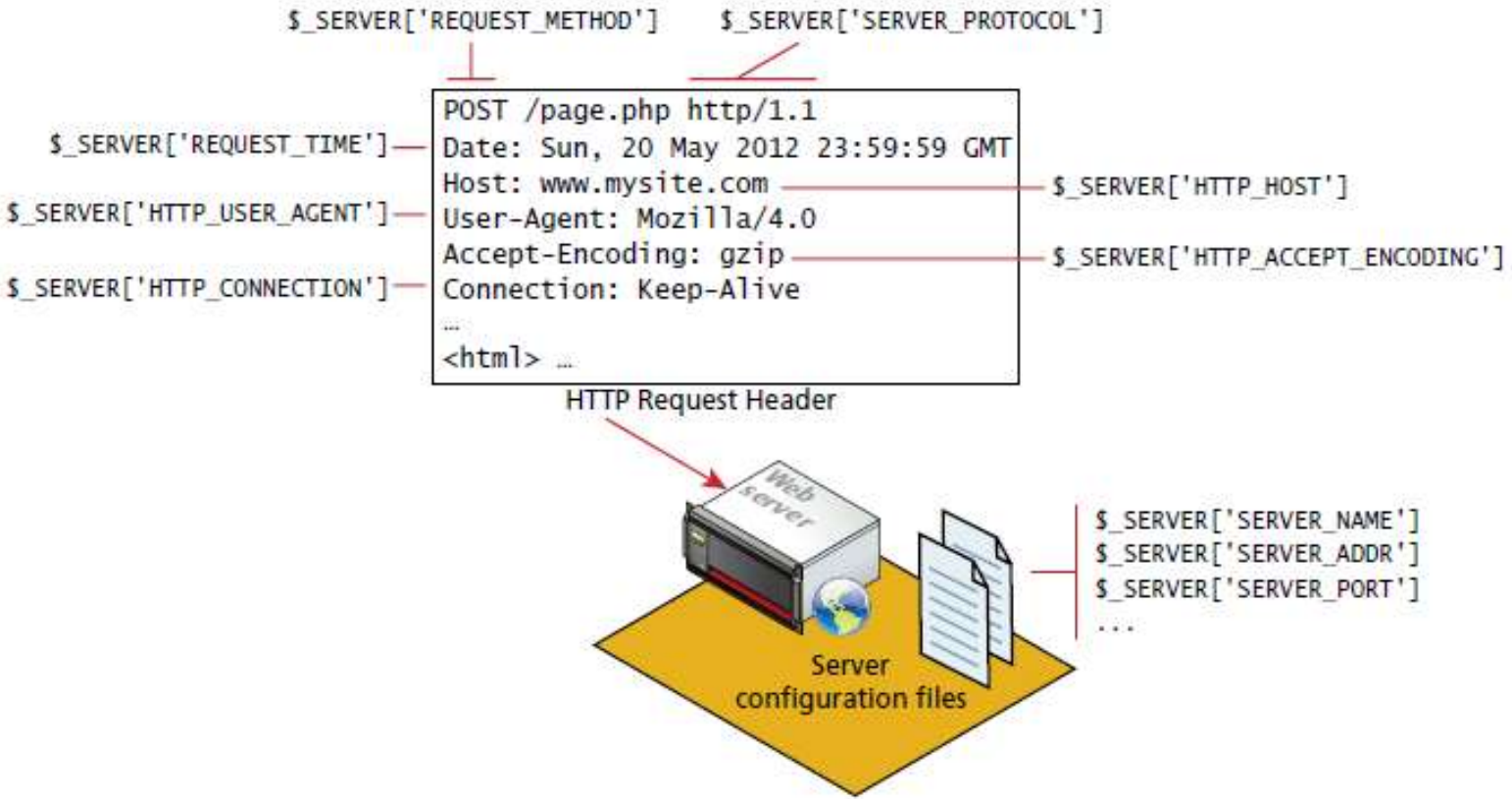


FIGURE Relationship between request headers, the server, and the $_SERVER array

Server Information Keys

SERVER_NAME is a key in the $_SERVER array that contains the name of the site that was requested. If you are running multiple hosts on the same code base, this can be a useful piece of information. SERVER_ADDR is a complementary key telling us the IP of the server. Either of these keys can be used in a conditional to output extra HTML to identify a development server, for example.

DOCUMENT_ROOT tells us the file location from which you are currently running your script. Since you are often moving code from development to production, this key can be used to great effect to create scripts that do not rely on a particular location to run correctly. This key complements the SCRIPT_NAME key that identifies the actual script being executed.

Request Header Information Keys

Recall that the web server responds to HTTP requests, and that each request contains a request header. These keys provide programmatic access to the data in the request header.

The `REQUEST_METHOD` key returns the request method that was used to access the page: that is, `GET`, `HEAD`, `POST`, `PUT`.

The `REMOTE_ADDR` key returns the IP address of the requestor, which can be a useful value to use in your web applications. In real-world sites these IP addresses are often stored to provide an audit trail of which IP made which requests, especially on sensitive matters like finance and personal information. In an online poll, for example, you might limit each IP address to a single vote. Although these can be forged, the technical competence required is high, thus in practice one can usually assume that this field is accurate.

One of the most commonly used request headers is the **user-agent** header, which contains the operating system and browser that the client is using. This header value can be accessed using the key `HTTP_USER_AGENT`. The user-agent string as posted in the header is cryptic, containing information that is semicolon-delimited and may be hard to decipher. PHP has included a comprehensive (but slow) method to help you debug these headers into useful information.

illustrates a script that accesses and echoes the user-agent header information.

```
<?php
echo $_SERVER['HTTP_USER_AGENT'];

$browser = get_browser($_SERVER['HTTP_USER_AGENT'], true);
print_r($browser);
?>
```

LISTING Accessing the user-agent string in the HTTP headers

Listing . shows an example of context-dependent output that outputs a message to clients that came to this page from the search page, a message that is not shown to clients that came from any other link. This allows us to output a link back to the search page, but only when the user arrived from the search page.

```
$previousPage = $_SERVER['HTTP_REFERER'];
// Check to see if referer was our search page
if (strpos("search.php",$previousPage) != 0) {
   echo "<a href='search.php'>Back to search</a>";
}
// Rest of HTML output
```

LISTING . Using the HTTP_REFERER header to provide context-dependent output

$_FILES Array

The $_FILES associative array contains items that have been uploaded to the current script. the <input type="file"> element is used to create the user interface for uploading a file from the client to the server. The user interface is only one part of the uploading process. A server script must process the upload file(s) in some way; the $_FILES array helps in this process.

HTML Required for File Uploads

To allow users to upload files, there are some specific things you must do:

- First, you must ensure that the HTML form uses the HTTP `POST` method, since transmitting a file through the URL is not possible.
- Second, you must add the `enctype="multipart/form-data"` attribute to the HTML form that is performing the upload so that the HTTP request can

  submit multiple pieces of data (namely, the HTTP post body, and the HTTP file attachment itself).
- Finally you must include an input type of `file` in your form. This will show up with a browse button beside it so the user can select a file from their computer to be uploaded. A simple form demonstrating a very straightforward file upload to the server is shown in Listing .

```
<form enctype='multipart/form-data' method='post'>
   <input type='file' name='file1' id='file1' />
   <input type='submit' />
</form>
```

LISTING HTML for a form that allows an upload

Handling the File Upload in PHP

The corresponding PHP file responsible for handling the upload (as specified in the HTML form's `action` attribute) will utilize the superglobal `$_FILES` array.[4] This array will contain a key=value pair for each file uploaded in the post. The key for each element will be the `name` attribute from the HTML form, while the value will be an array containing information about the file as well as the file itself. The keys in that array are the `name`, `type`, `tmp_name`, `error`, and `size`.

Figure illustrates the process of uploading a file to the server and how the corresponding upload information is contained in the `$_FILES` array. The values for each of the keys, in general, are described below.

- **`name`** is a string containing the full file name used on the client machine, including any file extension. It does not include the file path on the client's machine.
- **`type`** defines the `MIME` type of the file. This value is provided by the client browser and is therefore not a reliable field.
- **`tmp_name`** is the full path to the location on your server where the file is being temporarily stored. The file will cease to exist upon termination of the script, so it should be copied to another location if storage is required.
- **`error`** is an integer that encodes many possible errors and is set to `UPLOAD_ERR_OK` (integer value 0) if the file was uploaded successfully.
- **`size`** is an integer representing the size in bytes of the uploaded file.

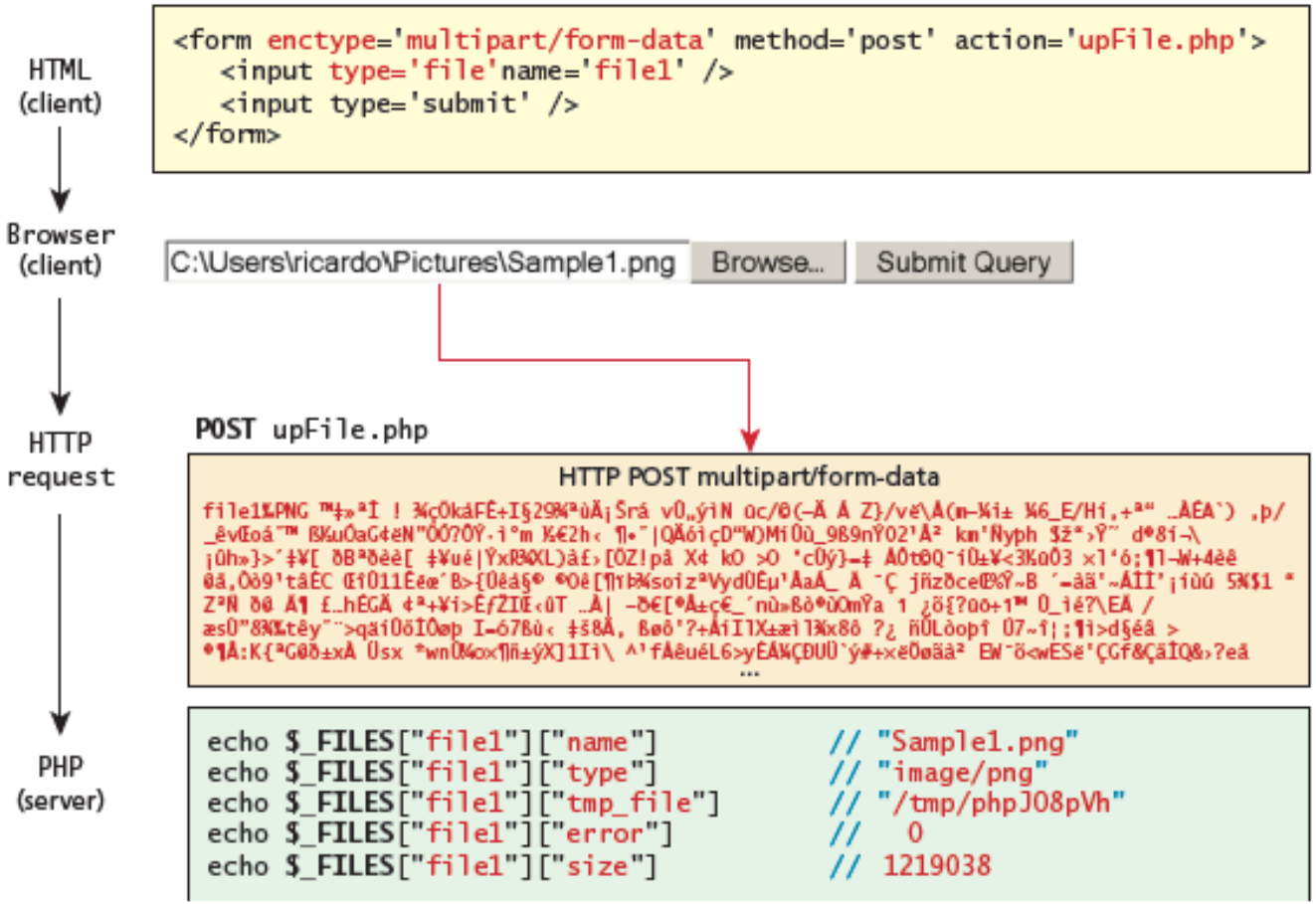


FIGURE Data flow from HTML form through POST to PHP $_FILES array

Checking for Errors

For every uploaded file, there is an error value associated with it in the `$_FILES` array. The error values are specified using constant values, which resolve to integers. The value for a successful upload is `UPLOAD_ERR_OK`, and should be looked for before proceeding any further. The full list of errors is provided in Table and shows that there are many causes for bad file uploads.

A proper file upload script will therefore check each uploaded file by checking the various error codes as shown in Listing .

Error Code	Integer	Meaning
UPLOAD_ERR_OK	0	Upload was successful.
UPLOAD_ERR_INI_SIZE	1	The uploaded file exceeds the `upload_max_filesize` directive in php.ini.
UPLOAD_ERR_FORM_SIZE	2	The uploaded file exceeds the `max_file_size` directive that was specified in the HTML form.
UPLOAD_ERR_PARTIAL	3	The file was only partially uploaded.
UPLOAD_ERR_NO_FILE	4	No file was uploaded. Not always an error, since the user may have simply not chosen a file for this field.
UPLOAD_ERR_NO_TMP_DIR	6	Missing the temporary folder.
UPLOAD_ERR_CANT_WRITE	7	Failed to write to disk.
UPLOAD_ERR_EXTENSION	8	A PHP extension stopped the upload.

TABLE Error Codes in PHP for File Upload Taken from php.net.[6]

```
foreach ($_FILES as $fileKey => $fileArray) {
   if ($fileArray["error"] != UPLOAD_ERR_OK) { // error
      echo "Error: " . $fileKey . " has error" . $fileArray["error"]
           . "<br>";
   }
   else {   // no error
      echo $fileKey . "Uploaded successfully ";
   }
}
```

LISTING Checking each file uploaded for errors

File Size Restrictions

Some scripts limit the file size of each upload. There are many reasons to do so, and ideally you would prevent the file from even being transmitted in the first place if it is too large. There are three main mechanisms for maintaining uploaded file size restrictions: via HTML in the input form, via JavaScript in the input form, and via PHP coding.

The first of these mechanisms is to add a hidden input field before any other input fields in your HTML form with a name of MAX_FILE_SIZE. This technique allows your php.ini maximum file size to be large, while letting some forms override that large limit with a smaller one. . It should be noted that though this mechanism is set up in the HTML form, it is only available to use when your server-side environment is using PHP.

```
<form enctype='multipart/form-data' method='post'>
   <input type="hidden" name="MAX_FILE_SIZE" value="1000000" />
   <input type='file' name='file1' />
   <input type='submit' />
</form>
```

LISTING Limiting upload file size via HTML

The more complete client-side mechanism to prevent a file from uploading if it is too big is to prevalidate the form using JavaScript. Such a script, to be added to a handler for the form, is shown in Listing . .

```
<script>
var file = document.getElementById('file1');
var max_size = document.getElementById("max_file_size").value;
if (file.files && file.files.length ==1){
   if (file.files[0].size > max_size) {
      alert("The file must be less than " + (max_size/1024) + "KB");
      e.preventDefault();
   }
}
</script>
```

LISTING Limiting upload file size via JavaScript

The third (and essential) mechanism for limiting the uploaded file size is to add a simple check on the server side (just in case JavaScript was turned off or the user modified the MAX_FILE_SIZE hidden field). This technique checks the file size on the server by simply checking the size field in the $_FILES array. Listing . shows an example of such a check.

```
$max_file_size = 10000000;
foreach($_FILES as $fileKey => $fileArray) {
   if ($fileArray["size"] > $max_file_size) {
      echo "Error: " . $fileKey . " is too big";
   }
   printf("%s is %.2f KB", $fileKey, $fileArray["size"]/1024);
}
```

LISTING Limiting upload file size via PHP

Limiting the Type of File Upload

Even if the upload was successful and the size was within the appropriate limits, you may still have a problem. What if you wanted the user to upload an image and they uploaded a Microsoft Word document? You might also want to limit the uploaded image to certain image types, such as jpg and png, while disallowing bmp and others. To accomplish this type of checking you typically examine the file extension and the type field. Listing ' ' shows sample code to check the file extension of a file, and also to compare the type to valid image types.

```
$validExt = array("jpg", "png");
$validMime = array("image/jpeg","image/png");
foreach($_FILES as $fileKey => $fileArray ){
   $extension = end(explode(".", $fileArray["name"]));
   if (in_array($fileArray["type"],$validMime) &&
         in_array($extension, $validExt)) {
      echo "all is well. Extension and mime types valid";
   }
   else {
      echo $fileKey." Has an invalid mime type or extension";
   }
}
```

LISTING PHP code to look for valid mime types and file extensions

Moving the File

With all of our checking completed, you may now finally want to move the temporary file to a permanent location on your server. Typically, you make use of the PHP function `move_uploaded_file()`, which takes in the temporary file location and the file's final destination. This function will only work if the source file exists and if the destination location is writable by the web server (Apache). If there is a problem the function will return false, and a warning may be output. Listing illustrates

```
$fileToMove = $_FILES['file1']['tmp_name'];
$destination = "./upload/" . $_FILES["file1"]["name"];
if (move_uploaded_file($fileToMove,$destination)) {
   echo "The file was uploaded and moved successfully!";
}
else {
   echo "there was a problem moving the file";
}
```

LISTING Using move_uploaded_file() function

Reading/Writing Files

Before the age of the ubiquitous database, software relied on storing and accessing data in files. In web development, the ability to read and write to text files remains an important technical competency. Even if your site uses a database for storing its information, the fact that the PHP file functions can read/write from a file or from an external website (i.e., from a URL) means that file system functions still have relevance even in the age of database-driven websites.

There are two basic techniques for read/writing files in PHP:

- **Stream access**. In this technique, our code will read just a small portion of the file at a time. While this does require more careful programming, it is the most memory-efficient approach when reading very large files.
- **All-In-Memory access**. In this technique, we can read the entire file into memory (i.e., into a PHP variable). While not appropriate for large files, it does make processing of the file extremely easy.

NOTE

When reading a file from an external site, you should be aware that your script will not proceed until the remote website responds to the request. If you do not control the other website, you should be cautious about relevant intellectual property restrictions on the data you are retrieving.

Stream Access

To those of you familiar with functions like `fopen()`, `fclose()`, and `fgets()` from the C programming language, this first technique will be second nature to you. In the C-style file access you separate the acts of opening, reading, and closing a file.

The function `fopen()` takes a file location or URL and access mode as parameters. The returned value is a **stream resource**, which you can then read sequentially. Some of the common modes are "r" for read, "rw" for read and write, and "c," which creates a new file for writing.

Once the file is opened, you can read from it in several ways. To read a single line, use the `fgets()` function, which will return false if there is no more data, and if it reads a line it will advance the stream forward to the next one so you can use the `===` check to see if you have reached the end of the file. To read an arbitrary amount of data (typically for binary files), use `fread()` and for reading a single character use `fgetsc()`. Finally, when finished processing the file you must close it using `fclose()`. Listing . · illustrates a script using `fopen()`, `fgets()`, and `fclose()` to read a file and echo it out (replacing new lines with
 tags).

```
$f = fopen("sample.txt", "r");
$ln = 0;
while ($line = fgets($f)) {
   $ln++;
   printf("%2d: ", $ln);
   echo $line . "<br>";
}
fclose($f);
```

LISTING Opening, reading lines, and closing a file

To write data to a file, you can employ the `fwrite()` function in much the same way as `fgets()`, passing the file handle and the string to write. However, as you do more and more processing in PHP, you may find yourself wanting to read or write entire files at once. In support of these situations there are simpler techniques, which we will now explore.

In-Memory File Access

While the previous approach to reading/writing files gives you complete control, the programming requires more care in dealing with the streams, file handles, and other low-level issues. The alternative simpler approach is much easier to use, at the cost of relinquishing fine-grained control. The functions shown in Table provide a simpler alternative to the processing of a file in PHP.

Function	Description
`file()`	Reads the entire file into an array, with each array element corresponding to one line in the file
`file_get_contents`	Reads the entire file into a string variable
`file_put_contents`	Writes the contents of a string variable out to a file

TABLE . In-Memory File Functions

The `file_get_contents()` and `file_put_contents()` functions allow you to read or write an entire file in one function call. To read an entire file into a variable you can simply use:

```
$fileAsString = file_get_contents(FILENAME);
```

To write the contents of a string `$writeme` to a file, you use

```
file_put_contents(FILENAME, $writeme);
```

Review Questions

1. What are the superglobal arrays in PHP?
2. What function is used to determine if a value was sent via query string?
3. How do we handle arrays of values being posted to the server?
4. Describe the relationship between keys and indexes in arrays.
5. How does one iterate through all keys and values of an array?
6. Are arrays sorted by key or by value, or not at all?
7. How would you get a random element from an array?
8. What does `urlencode()` do? How is it "undone"?
9. What information is uploaded along with a file?
10. How do you read or write a file on the server from PHP?
11. List and briefly describe the ways you can limit the types and size of file uploaded.
12. What classes of information are available via the `$_SERVER` superglobal array?
13. Describe why hidden form fields can easily be forged/changed by an end user.

SQL

Although non-SQL options are available, relational databases almost universally use Structured Query Language or, as it is more commonly called, SQL (pronounced *sequel*) as the mechanism for storing and manipulating data. While each DBMS typically adds its own extensions to SQL, the basic syntax for retrieving and modifying data is standardized and similar.

SQL keyword that indicates the type of query (in this case a query to retrieve data)

SQL keyword for specifying the tables

```
SELECT ISBN10, Title FROM Books
```

Fields to retrieve

Table to retrieve from

```
SELECT * FROM Books
```

Wildcard to select all fields

Note: While the wildcard is convenient, especially when testing, for production code it is usually avoided; instead of selecting every field, you should select just the fields you need.

```
select iSbN10, title
FROM BOOKS
ORDER BY title
```

SQL keyword to indicate sort order

Field to sort on

Note: SQL doesn't care if a command is on a single line or multiple lines, nor does it care about the case of keywords or table and field names. Line breaks and keyword capitalization are often used to aid in readability.

```
SELECT ISBN10, Title FROM Books
ORDER BY CopyrightYear DESC, Title ASC
```

Keywords indicating that sorting should be in descending or ascending order (which is the default)

Several sort orders can be specified: in this case the data is sorted first on year, then on title.

```
SELECT isbn10, title FROM books
WHERE copyrightYear > 2010
```

SQL keyword that indicates to return only those records whose data matches the criteria expression

Expressions take form: field *operator* value

```
SELECT isbn10, title FROM books
WHERE category = 'Math' AND copyrightYear = 2014
```

Comparisons with strings require string literals (single or double quote)

FIGURE 11.11 Using the WHERE clause

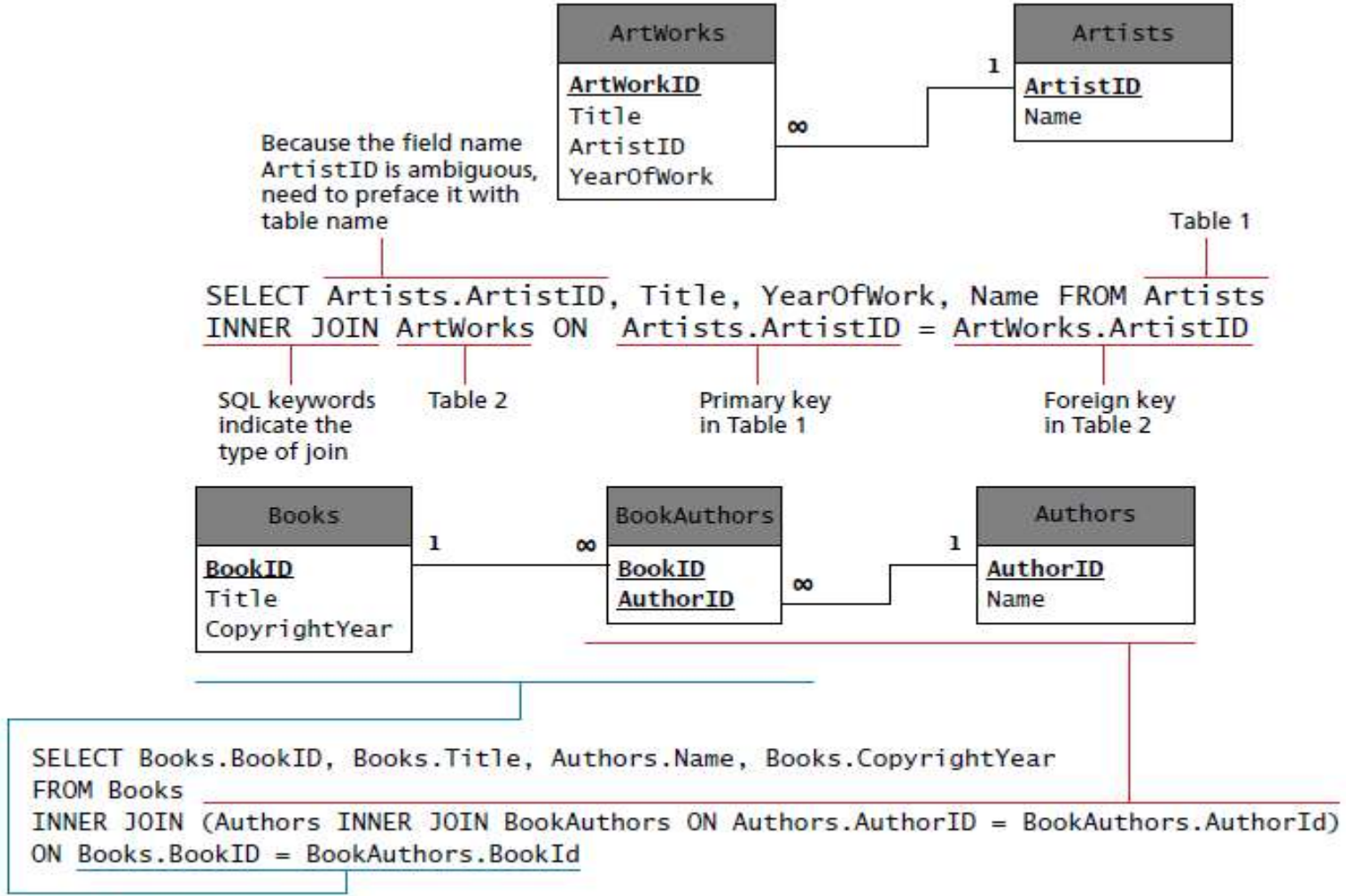


FIGURE SQL SELECT from multiple tables using an INNER JOIN

This aggregate function returns a count of the number of records.

Defines an alias for the calculated value

```
SELECT Count(ArtWorkID) AS NumPaintings
FROM ArtWorks
WHERE YearOfWork > 1900
```

Count number of paintings after year 1900

Note: This SQL statement returns a single record with a single value in it.

NumPaintings
745

```
SELECT Nationality, Count(ArtistID) AS NumArtists
FROM Artists
GROUP BY Nationality
```

SQL keywords to group output by specified fields

Note: This SQL statement returns as many records as there are unique values in the group-by field.

Nationality	NumArtists
Belgium	4
England	15
France	36
Germany	27
Italy	53

FIGURE . Using GROUP BY with aggregate functions

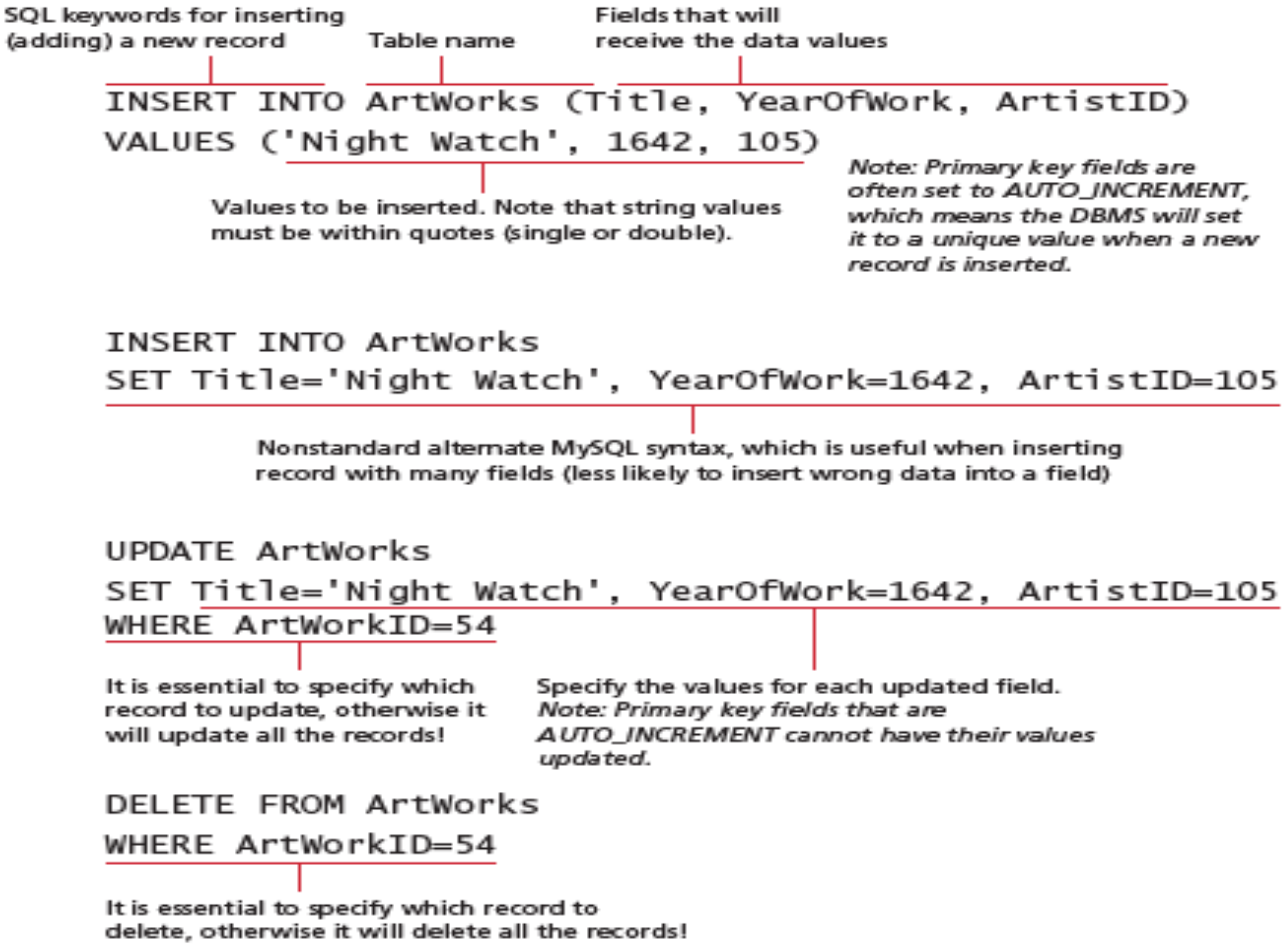


FIGURE · SQL INSERT, UPDATE, and DELETE

Go through for these topics in Text Book Fundamentals of Web technology By Randy Connolly

- Managing MySQL Dataase
- Accessing MySQL in PHP,
- Sample database techniques