Internal Assessment Test 3 – NOV. 2019

Sub:	Web Technologies & its Applications					Sub Code:	15CS71	Branch:	CSE
Date:	18-11-2019	Duration:	90 min's	Max Marks:	50	Sem / Sec:	7 – A, B & C		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1(a)	Briefly explain Arrays in PHP with examples. **PHP Arrays** PHP supports arrays. An array is a data structure that allows the programmer to collect a number of related elements together in a single variable. An array is actually an **ordered map**, which associates each value in the array with a key. \$days Keys 0 | 1 | 2 | 3 | 4 "Mon"| "Tue"| "Wed"| "Thu" | "Fri" Values **FiGURE 9.1** 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. 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.	[10]	CO 3	L2

Defining and Accessing an Array

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 9.1, you declare it with a comma-delimited list of values inside the ( ) braces using either of two following syntax's:

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

no keys are explicitly defined for the array, the default key values are 0, 1, 2, . . . , n.

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";
```

```
$days = array(0 =>"Mon", 1 =>"Tue", 2 =>"Wed", 3 =>"Thu", 4=>"Fri");
0->key
Mon->value
```

Figure 9.2 Explicitly assigning keys to array elements

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

Key->mon

Value->40

echo $forecast["Tue"]; *// outputs 47*

echo $forecast["Thu"]; *// outputs 40*

Consider an array to be a dictionary or hash map. These types of arrays in PHP are generally referred to as **associative arrays**.

to access an element in an associative array, you simply use the key value rather than an index:

echo $forecast["Wed"]; *// this will output 52*

Multidimensional Arrays

PHP also supports multidimensional arrays.

$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 9.1 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.

```
// 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>";
}
```

Code: Iterating through an array using while, do while, and for loops

Output the content of the $days array using the built-in function count()

For an associative array, the foreach loop and illustrated for the $forecast array

```
// foreach: iterating through the values
foreach ($forecast as $value) {
echo $value . "<br>";
}
// foreach: iterating through the values AND the keys
foreach ($forecast as $key => $value) {
echo "day" . $key . "=" . $value;
}
```

Adding and Deleting Elements

In PHP, arrays are dynamic, that is, they can grow or shrink in size. An element can be added to an array simply by using a key/index that hasn't been used, as shown below:

```
$days[5] = "Sat";
```

Since there is no current value for key 5, the array grows by one, with the new key/value pair added to the end of our array. If the key had a value already, the same style of assignment replaces the value at that key. As an alternative to specifying the index, a new element can be added to the end of any array using the following technique:

```
$days[ ] = "Sun";
$days = array("Mon","Tue","Wed","Thu","Fri");
$days[7] = "Sat";
print_r($days);
```

What will be the output of the print_r()? It will show that our array now contains the following:

Array ([0] => Mon [1] => Tue [2] => Wed [3] => Thu [4] => Fri [7] => Sat)

Referencing $days[6], for instance, it will return a NULL value, which is a special PHP value that represents a variable with no value. You can also create "gaps" by explicitly deleting array elements using the unset() function,

```
$days = array("Mon","Tue","Wed","Thu","Fri");
unset($days[2]);
unset($days[3]);
print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )
$days = array_values($days);
print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [2] => Fri )
```

you can remove "gaps" in arrays (which really are just gaps in the index keys) using the array_values() function, which reindexes the array numerically.

Checking If a Value Exists

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.

```
$oddKeys = array (1 =>"hello", 3 =>"world", 5 =>"!");
if (isset($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];
}
```

Array Sorting

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

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_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.			
2(a)	Explain selectors in JQuery with examples. **jQuery Selectors** **Basic Selectors** The four basic selectors were defined, and include the universal selector, class selectors, id selectors, and elements selectors. To review: ▪ $("*") **Universal selector** matches all elements (and is slow). ▪ $("tag") **Element selector** matches all elements with the given element name. ▪ $(".class") **Class selector** matches all elements with the given CSS class. ▪ $("#id") **Id selector** matches all elements with a given HTML id attribute. For example, to select the single element with id="grab"you would write: **var singleElement = $("#grab");** To get a set of all the <a>elements the selector would be: **var allAs = $("a");** These selectors are powerful enough that they can replace the use of **getElementById()** entirely. **Attribute Selector** An **attribute selector** provides a way to select elements by either the presence of an element attribute or by the value of an attribute.	[10]	CO 5	L2

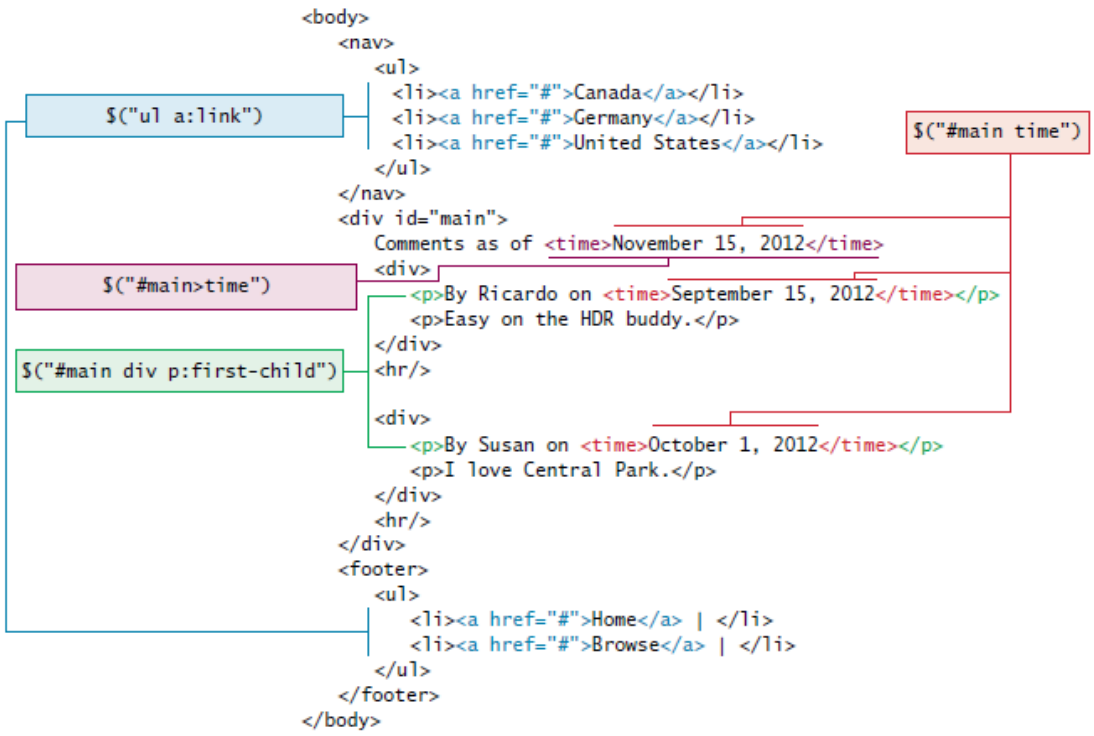


FIGURE 15.4 Illustration of some jQuery selectors and the HTML being selected

A list of sample CSS attribute selectors was given in Chapter 3 (Table 3.4), but to jog your memory with an example, consider a selector to grab all <img>elements with an src attribute beginning with /artist/ as:

```
var artistImages = $("img[src^='/artist/']");
```

Recall that you can select by attribute with square brackets ([attribute]), specify a value with an equals sign ([attribute=value]) and search for a particular value inthe beginning, end, or anywhere inside a string with ^, $, and * symbols respectively.

([attribute^=value], [attribute$=value], [attribute*=value]).

Pseudo-Element Selector

Pseudo-elements are special elements, which are special cases of regular ones. these **pseudo-element selectors** allow you to append to any selector using the colon and one of :link, :visited, :focus, :hover,:active, :checked, :first-child, :first-line, and :first-letter.

These selectors can be used in combination with the selectors presented above, or alone. Selecting all links that have been visited, for example, would be specified with:

```
var visitedLinks = $("a:visited");
```

Contextual Selector

Another powerful CSS selector included in jQuery's selection mechanism is the **contextual selectors** introduced in Chapter 3. These selectors allowed you to specify elements with certain relationships to one another in your CSS. These relationships included descendant (space), child (>), adjacent sibling (+), and general sibling (~).To select all <p>elements inside of elements you would write

```
var para = $("div p");
```

Content Filters

The **content filter** is the only jQuery selector that allows you to append filters to all ofthe selectors you've used thus far and match a particular pattern. You can select elements that have a particular child using :has(), have no children using :empty, or match a particular piece of text with :contains(). Consider the following example:

```
var allWarningText = $("body *:contains('warning')");
```


```
var allWarningText = $("body *:contains('warning')");
```


It will return a list of all the DOM elements with the word *warning* inside of them. You might imagine how we may want to highlight those DOM elements by coloring the background red as shown in Figure 15.5 with one line of code:

```
$("body *:contains('warning')").css("background-color", "#aa0000");
```

Selector	CSS Equivalent	Description
$(:button)	$("button, input[type='button']")	Selects all *buttons*.
$(:checkbox)	$('[type=checkbox]')	Selects all *checkboxes*.
$(:checked)	No equivalent	Selects elements that are checked. This includes radio buttons and checkboxes.
$(:disabled)	No equivalent	Selects form elements that are disabled. These could include <button>, <input>, <optgroup>, <option>, <select>, and <textarea>
$(:enabled)	No equivalent	Opposite of :disabled. It returns all elements where the disabled attribute=false as well as form elements with no disabled attribute.
$(:file)	$('[type=file]')	Selects all elements of type file.
$(:focus)	$(document.activeElement)	The element with focus.
$(:image)	$('[type=image]')	Selects all elements of type Image.
$(:input)	No equivalent	Selects all <input>, <textarea>, <select>, and <button> elements.
$(:password)	$('[type=password]')	Selects all password fields.
$(:radio)	$('[type=radio]')	Selects all radio elements.
$(:reset)	$('[type=reset]')	Selects all the reset buttons.
$(:selected)	No equivalent	Selects all the elements that are currently selected of type <option>. It does not include checkboxes or radio buttons.
$(:submit)	$('[type=submit]')	Selects all submit input elements.
$(:text)	No equivalent	Selects all input elements of type text. $('[type=text]') is almost the same, except that $(:text) includes <input>. fields with no type specified.

TABLE 15.1 jQuery form selectors and their CSS equivalents when applicable

Form Selectors

Since form HTML elements are well known and frequently used to collect and transmit data, there are jQuery selectors written especially for them. These selectors, listed in Table 15.1, allow for quick access to certain types of field as well as fields in certain states. attributes like the href attribute of an <a>tag, the src attribute of an <img>, or the class attribute of most elements. In jQuery we can

	both set and get an attribute value by using the attr() method on any element from a selector. This function takes a parameter to specify which attribute, and the optional second parameter is the value to set it to. If no second parameter is passed, then the return value of the call is the current value of the attribute. Some example usages are: ***// var link is assigned the href attribute of the first <a> tag*** **var link = $("a").attr("href");** ***// change all links in the page to http://funwebdev.com*** **$("a").attr("href","http://funwebdev.com");** ***// change the class for all images on the page to fancy*** **$("img").attr("class","fancy");**			
3(a)	Explain super global array FILES with examples. **$_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. <form enctype='multipart/form-data' method='post'> <input type='file' name='file1' id='file1' /> <input type='submit' /> </form>	[10]	CO 4	L2

Handling the File Upload in PHP

The corresponding PHP file responsible for handling the upload will utilize the superglobal $_FILES array. 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.

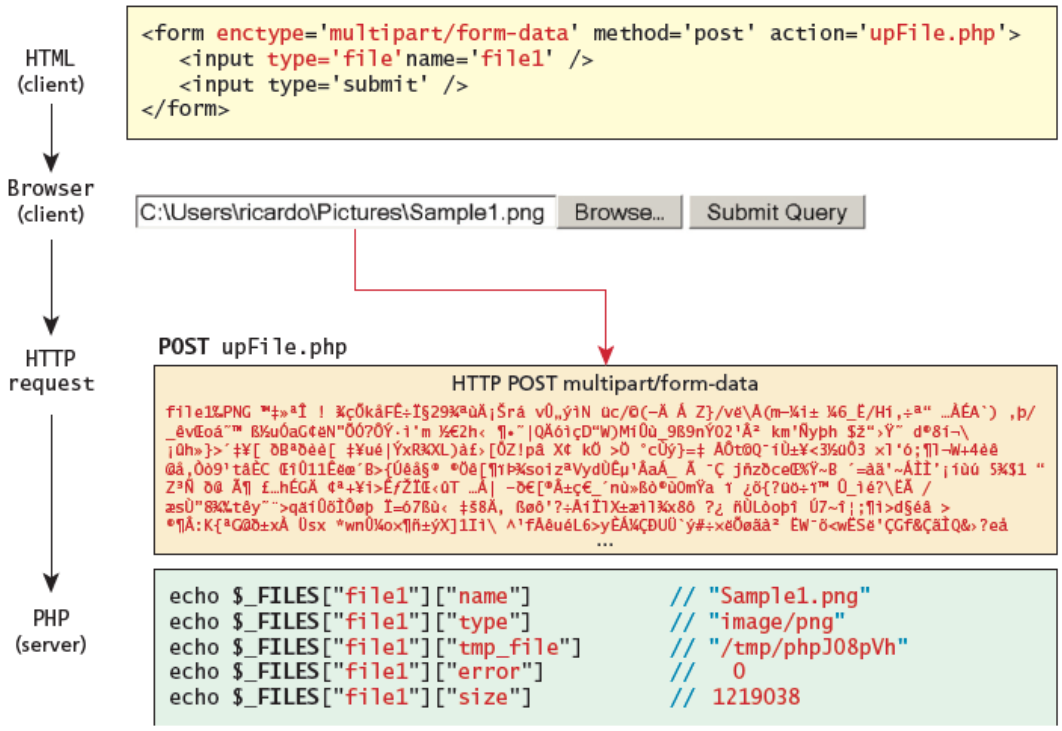


FIGURE 9.12 Data flow from HTML form through POST to PHP $_FILES array

The values for each of the keys 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.

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.

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_PART IAL	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.

```
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 ";
```

```
}
}
```

File Size Restrictions

Some scripts limit the file size of each upload. There are three main mechanisms for maintaining uploaded file size restrictions:

- HTML in the input form
- JavaScript in the input form
- PHP coding

HTML in the input form

This mechanism 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.

```
<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>
```

JavaScript in the input form

The more complete client-side mechanism to prevent a file from uploading if it

is too big is to prevalidate the form using JavaScript.

```
<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>
```

PHP coding

This mechanism is to add a simple check on the server side.

```
$max_file_size = 10000000;
foreach($_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);
}
```

Limiting the Type of File Upload

Even if the upload was successful and the size was within the appropriate limits, 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.

```
$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";
	}
	}
	
	## 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.
	
	$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";
	}			

4(a)	Explain super global array $_GET and $_POST. The $_GET and $_POST arrays are the most important super global variables in PHP since they allow the programmer to access data sent by the client in a query string.	[10]	CO 4	L2

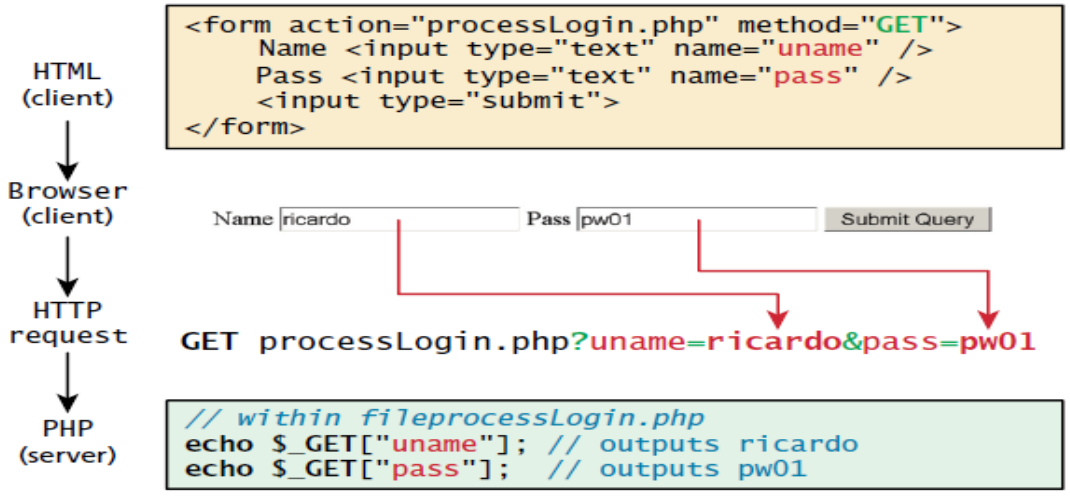


FIGURE 9.5 Illustration of flow from HTML, to request, to PHP's $_GET array

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.

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.

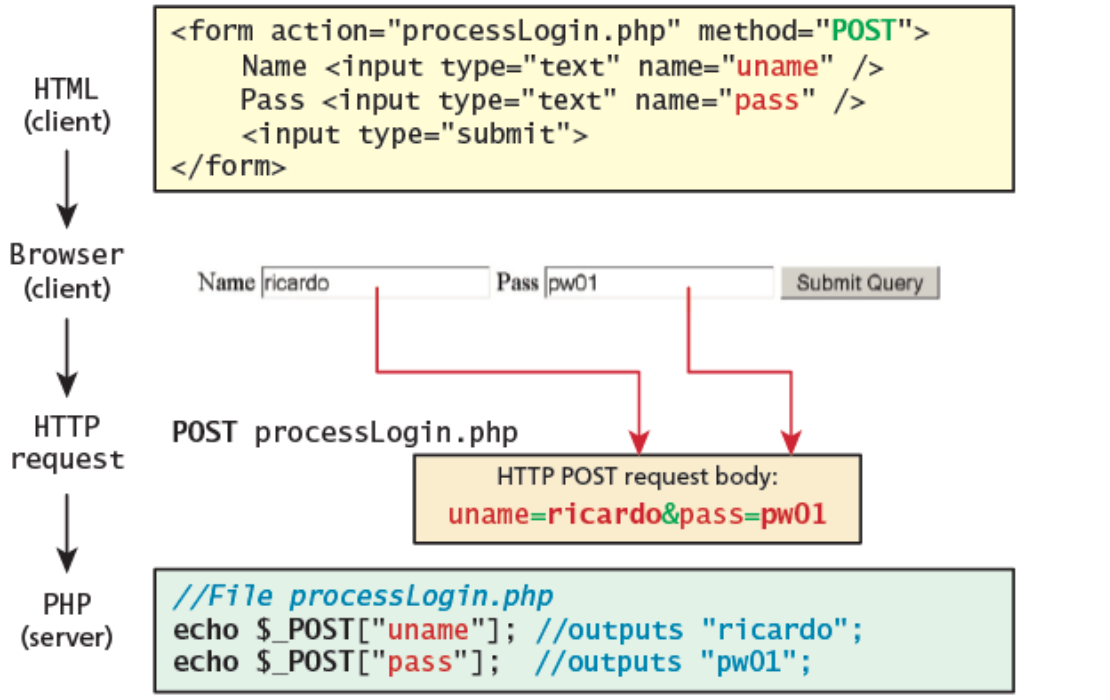


FIGURE 9.6 Data flow from HTML form through HTTP request to PHP's $_POST array

Determining If Any Data Sent

PHP that you will use the same file to handle both the display of a form as well as the form input. For example, a single file is often used to display a login form to the user, and that same file also handles the processing of the submitted form data, as shown in Figure 9.8. In such cases you may want to know whether any form data was submitted at all using either POST or GET.

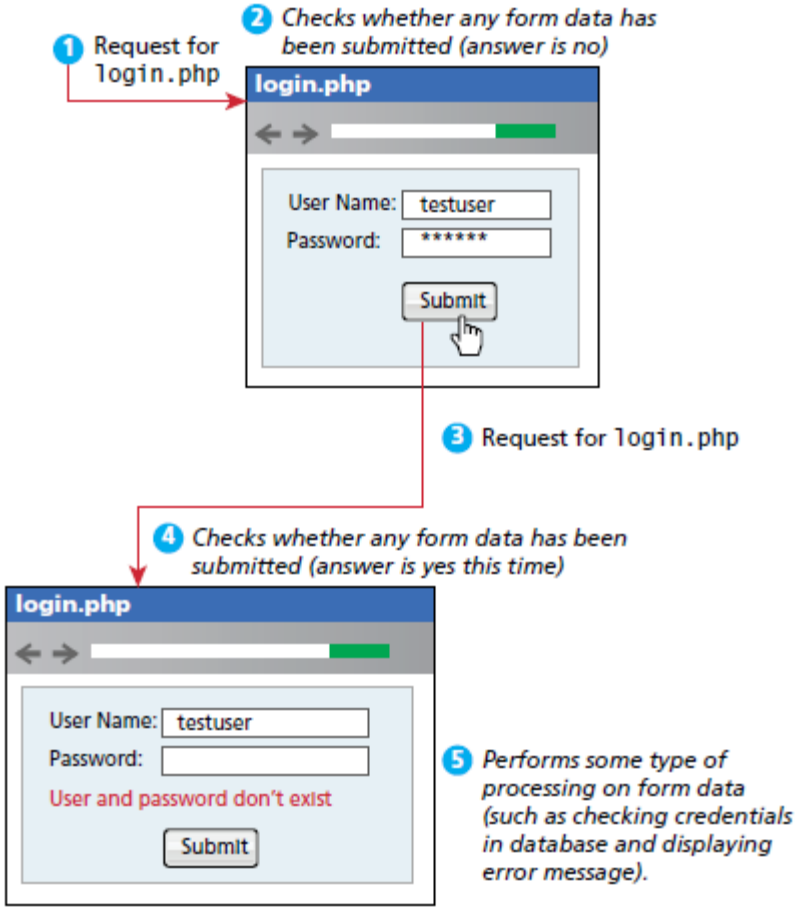


FIGURE 9.8 Form display and processing by the same PHP page

In PHP, there are several techniques to accomplish this task. First, you can determine if you are responding to a POST or GET by checking the $_SERVER['REQUEST_METHOD'] variable.

To check if any of the fields are set. To do this you can use the isset() function in PHP to see if there is anything set for a particular query string parameter.

```
<!DOCTYPE html>
<html>
<body>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
if ( isset($_POST["uname"]) && isset($_POST["pass"]) ) {
```

```
// handle the posted data.
echo "handling user login now ...";
echo "... here we could redirect or authenticate ";
echo " and hide login form or something else";
}
}?>
<h1>Some page that has a login form</h1>
<form action="samplePage.php" method="POST">
Name <input type="text" name="uname"/><br/>
Pass <input type="password" name="pass"/><br/>
<input type="submit">
</form>
</body>
</html>
```

5(a) How cookies and session work? Give examples. [10] CO 2 L2

Cookies

Cookies are a client-side approach for persisting state information. They are name=value pairs that are saved within one or more text files that are managed by the browser. These pairs accompany both server requests and responses within the HTTP header. While cookies cannot contain viruses, third-party tracking cookies have been a source of concern for privacy advocates.

keep track of whether a user has logged into a site.

How Do Cookies Work?

While cookie information is stored and retrieved by the browser, the information in a cookie travels within the HTTP header. Figure 13.6 illustrates how cookies work. There are limitations to the amount of information that can be stored in a cookie (around 4K) and to the number of cookies for a domain (for instance, Internet Explorer 6 limited a domain to 20 cookies).HTTP cookies can also expire. That is, the browser will delete cookies that are beyond their expiry date (which is a configurable property of a cookie). If a cookie does not have an expiry date specified, the browser will delete it when the browser closes (or the next time it accesses the site). For this reason, some commentators will say that there are two types of cookies session cookies and persistent cookies. A **session cookie** has no expiry stated and thus will be deleted at the end of the user browsing session. **Persistent cookies** have an expiry date specified; they will persist in the browser's cookie file until the expiry date occurs, after which they are deleted.

Using Cookies

Like any other web development technology, PHP provides mechanisms for writing and reading cookies. Cookies in PHP are *created* using the **setcookie()** function and are *retrieved* using the $_COOKIES superglobal associative array.Below example illustrates the writing of a persistent cookie in PHP

```
<?php
// add 1 day to the current time for expiry time
$expiryTime = time()+60*60*24;
// create a persistent cookie
$name = "Username";
$value = "Ricardo";
setcookie($name, $value, $expiryTime);
?>
```

The **setcookie()** function also supports several more parameters, which further customize the new cookie. You can examine the online official PHP documentation for more information. The below example illustrates the reading of cookie values. Notice that when we read a cookie, we must also check to ensure that the cookie exists. In PHP, if the cookie has expired (or never existed in the first place), then the client's browser would not send anything, and so the $_COOKIE array would be blank.

```
<?php
if( !isset($_COOKIE['Username']) ) {
//no valid cookie found
}
else {
echo "The username retrieved from the cookie is:";
echo $_COOKIE['Username'];
}
?>
```

Session State

Session state is a server-based state mechanism that lets web applications store and retrieve objects of any type for each unique user session. That is, each browser session has its own session state stored as a serialized file on the server, which is deserialized and loaded into memory as needed for each request

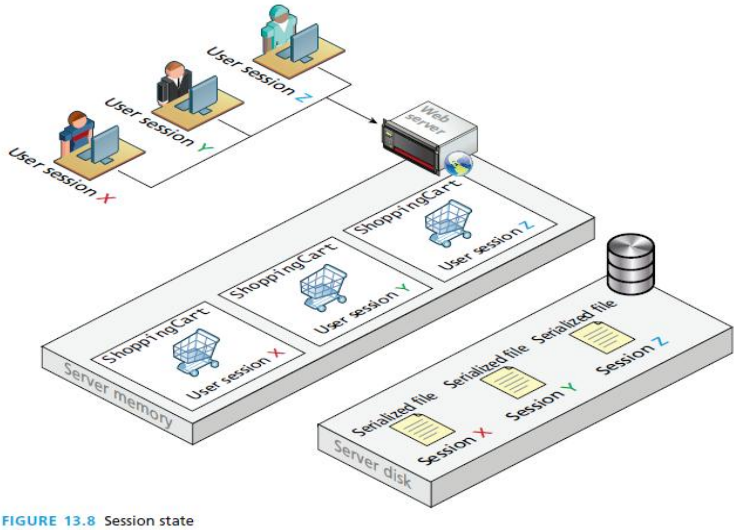


FIGURE 13.8 Session state

Because server storage is a finite resource, objects loaded into memory are released when the request completes, making room for other requests and their session objects. This means there can be more active sessions on disk than in memory at any one time. Session state is ideal for storing

more complex (but not too complex . . . more on that later) objects or data structures that are associated with a user session. The classic example is a shopping cart. While shopping carts could be implemented via cookies or query string parameters, it would be quite complex and cumbersome to do so.

It can be accessed via the $_SESSION variable, but unlike the other superglobals, you have to take additional steps in your own code in order to use the $_SESSION superglobal To use sessions in a script, you must call the session_start() function at the beginning of the script as shown in Listing 13.5.

```
<?php
session_start();
if ( isset($_SESSION['user']) ) {
// User is logged in
}
else {
// No one is logged in (guest)
}
?>
```

In this example, we differentiate a logged-in user from a guest by checking for the existence of the $_SESSION['user'] variable. Session state is typically used for storing information that needs to be preserved across multiple requests by the same user. Since each user session has its own session state collection, it should not be used to store large amounts of information because this will consume very large amounts of server memory as the number of active sessions increase. As well, since session information does eventually time out, one should always check if an item retrieved from session state still exists before using the retrieved object. If the session object does not yet exist (either because it is the first time the user has requested it or because the session has timed out), one might generate an error, redirect to another page, or create the required object using the lazy initialization approach as shown in Listing 13.6. In this example ShoppingCart is a user defined class. Since PHP sessions are serialized into files, one must ensure that any

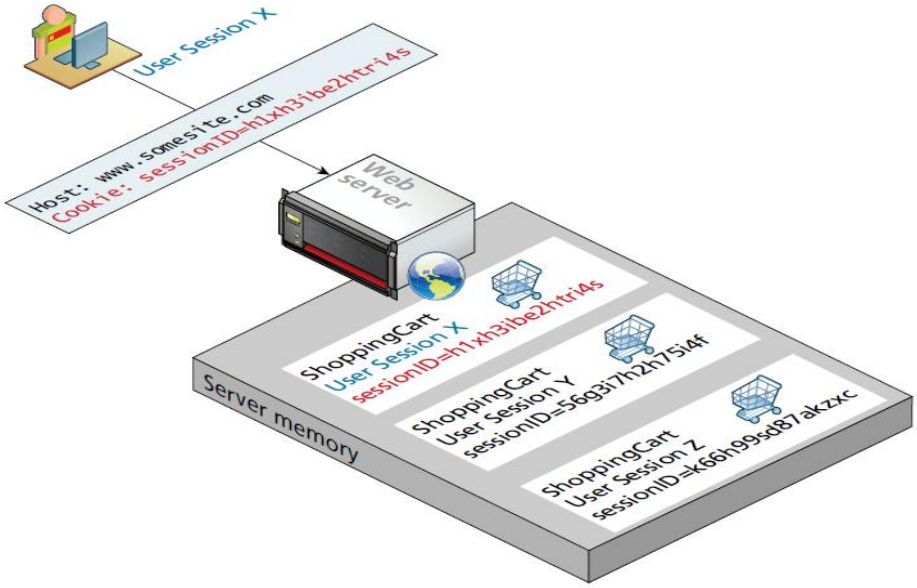
listing 13.5 Accessing session state

```
<?php
include_once("ShoppingCart.class.php");
session_start();
// always check for existence of session object before accessing it
if ( !isset($_SESSION["Cart"]) ) {
//session variables can be strings, arrays, or objects, but
// smaller is better
$_SESSION["Cart"] = new ShoppingCart();
}
$cart = $_SESSION["Cart"];
?>
```

How Does Session State Work?

The first thing to know about session state is that it works within the same HTTP context as any web request. The server needs to be able to identify a given HTTP request with a specific user request. Since HTTP is stateless, some type of user/session identification system is needed.

In PHP, this is a unique 32-byte string that is by default transmitted back and forth between the user and the server via a session cookie as shown in Figure 13.9.



For a brand new session, PHP assigns an initially empty dictionary-style collection that can be used to hold any state values for this session. When the request processing is finished, the session state is saved to some type of state storage mechanism, called a session state provider (discussed in next section).Finally, when a new request is received for an already existing session; the session's dictionary collection is filled with the previously saved session data from the session state provider.

6(a)	How do you achieve encapsulation in PHP. Give examples. **Data Encapsulation** Perhaps the most important advantage to object-oriented design is the possibility of encapsulation, which generally refers to restricting access to an object's internal components. Another way of understanding encapsulation is: it is the hiding of an object's implementation details. A properly encapsulated class will define an interface to the world in the form of its public methods, and leave its data, that is, its properties, hidden (that is, private). This allows the class to control exactly how its data will be used. If a properly encapsulated class makes its properties private, then how do you access them? The typical approach is to write methods for accessing and modifying properties rather than allowing them to be accessed directly. These methods are commonly called getters and setters (or accessors and mutators). Some development environments can even generate getters and setters automatically. A getter to return a variable's value is often very straightforward and should not modify the property. It is normally called without parameters, and returns the property from within the class. For instance: public function getFirstName()	[05]	CO 4	L2

{

return $this->firstName;

}

Setter methods modify properties, and allow extra logic to be added to prevent properties from being set to strange values. For example, we might only set a date property if the setter was passed an acceptable date:

```
public function setBirthDate($birthdate){
   // set variable only if passed a valid date string
   $date = date_create($birthdate);

   if ( ! $date ) {
      $this->birthDate = $this->getEarliestAllowedDate();
   }
   else {
      // if very early date then change it to
      // the earliest allowed date
      if ( $date < $this->getEarliestAllowedDate() ) {
         $date = $this->getEarliestAllowedDate();
      }
      $this->birthDate = $date;
   }
}
```

```php
class Artist {
   const EARLIEST_DATE = 'January 1, 1200';

   private static $artistCount = 0;
   private $firstName;
   private $lastName;
   private $birthDate;
   private $deathDate;
   private $birthCity;

   // notice constructor is using setters instead
   // of accessing properties
   function __construct($firstName, $lastName, $birthCity, $birthDate,
                        $deathDate) {
      $this->setFirstName($firstName);
      $this->setLastName($lastName);
      $this->setBirthCity($birthCity);
      $this->setBirthDate($birthDate);
      $this->setDeathDate($deathDate);
      self::$artistCount++;
   }
   // saving book space by putting each getter on single line
   public function getFirstName() { return $this->firstName; }
   public function getLastName()  { return $this->lastName; }
   public function getBirthCity() { return $this->birthCity; }
   public function getBirthDate() { return $this->birthDate; }
   public function getDeathDate() { return $this->deathDate; }
   public static function getArtistCount() { return self::$artistCount; }
   public function getEarliestAllowedDate () {
      return date_create(self::EARLIEST_DATE);
   }
```

```php
   public function setLastName($lastName)
     { $this->lastName = $lastName; }
   public function setFirstName($firstName)
     { $this->firstName = $firstName; }
   public function setBirthCity($birthCity)
     { $this->birthCity = $birthCity; }

   public function setBirthDate($birthdate) {
      // set variable only if passed a valid date string
      $date = date_create($birthdate);
      if ( ! $date ) {
         $this->birthDate = $this->getEarliestAllowedDate();
      }
      else {
```

```
            // if very early date then change it to earliest allowed date
            if ( $date < $this->getEarliestAllowedDate()  ) {
               $date = $this->getEarliestAllowedDate();
            }
            $this->birthDate = $date;
         }
      }

      public function setDeathDate($deathdate) {
         // set variable only if passed a valid date string
         $date = date_create($deathdate);

         if ( ! $date ) {
            $this->deathDate = $this->getEarliestAllowedDate();
         }
         else {
            // set variable only if later than birth date
            if ($date > $this->getBirthDate()) {
            $this->deathDate = $date;
            }
            else {
               $this->deathDate = $this->getBirthDate();
            }
         }
      }
   }
```

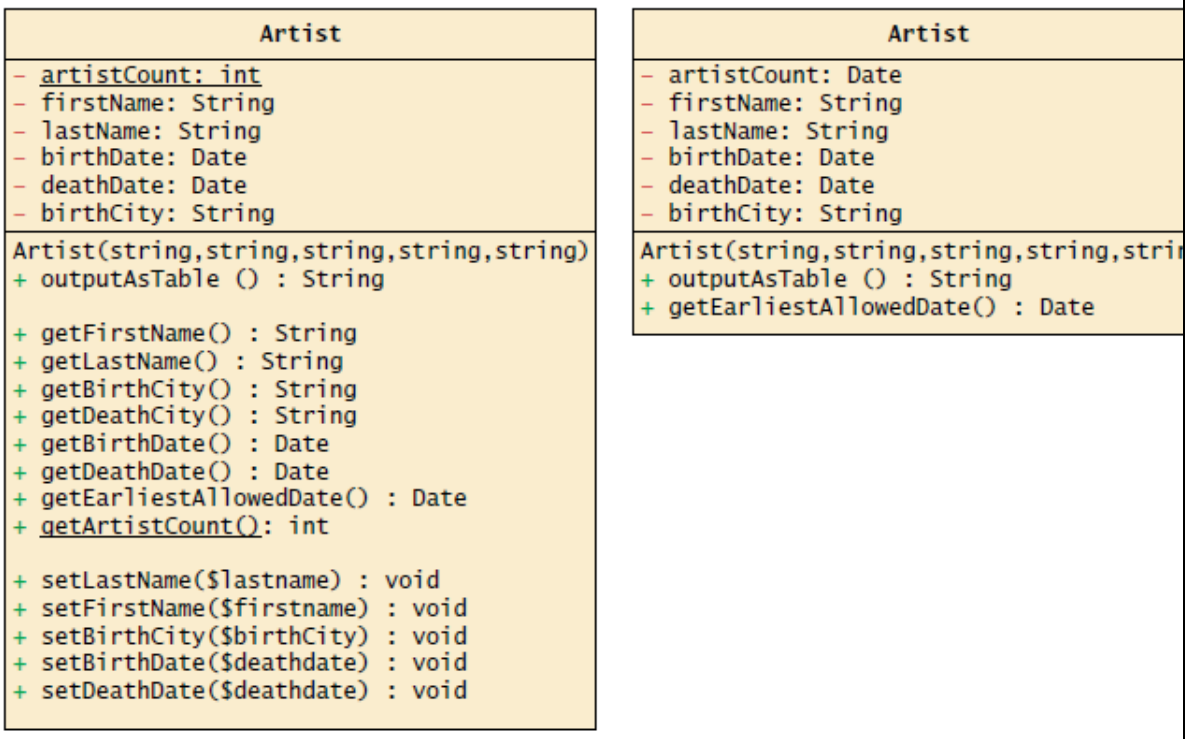


FIGURE 10.8 Class diagrams for fully encapsulated Artist class

The below example demonstrates how the Artist class could be used and

tested.

```
<html>
  <body>
  <h2>Tester for Artist class</h2>

  <?php
  // first must include the class definition
  include 'Artist.class.php';

  // now create one instance of the Artist class
  $picasso = new Artist("Pablo","Picasso","Malaga","Oct 25,1881",
                        "Apr 8,1973");
```

```
  // output some of its fields to test the getters
  echo $picasso->getLastName() . ': ';
  echo date_format($picasso->getBirthDate(),'d M Y') . ' to ';
  echo date_format($picasso->getDeathDate(),'d M Y') . '<hr>';

  // create another instance and test it
  $dali = new Artist("Salvador","Dali","Figures","May 11,1904",
                     "January 23,1989");

  echo $dali->getLastName() . ': ';
  echo date_format($dali->getBirthDate(),'d M Y') . ' to ';
  echo date_format($dali->getDeathDate(),'d M Y'). '<hr>';

  // test the output method
  echo $picasso->outputAsTable();

  // finally test the static method: notice its syntax
  echo '<hr>';
  echo 'Number of Instantiated artists: ' . Artist::getArtistCount();

  ?>
  </body>
  </html>
```

6(b)	Write short notes on JSON with examples. **JSON** **JSON** stands for JavaScript Object Notation; its use is not limited to JavaScript. It provides a more concise format than XML to represent data. It was originally designed to provide a lightweight serialization format to represent objects in JavaScript. While it doesn't have the validation and readability of XML, it has the advantage of generally requiring significantly fewer bytes to represent data than XML, which in the web context is quite significant. Like XML, JSON is a data serialization format. That is, it is used to represent object data in a text format so that it can be transmitted from one computer to another. Many REST web services encode their returned data in the JSON data format instead of XML.	[05]	CO 5	L2

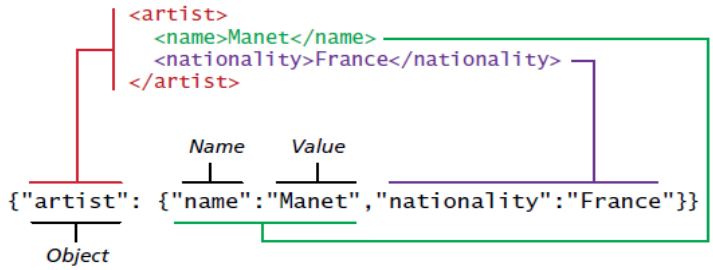


FIGURE 17.6 Sample JSON

Just like XML, JSON data can be nested to represent objects within objects. In general JSON data will have all white space removed to reduce the number of bytes traveling across the network.

```
{
   "paintings": [
      {
         "id":290,
         "title":"Balcony",
         "artist":{
            "name":"Manet",
            "nationality":"France"
         },
         "year":1868,
         "medium":"Oil on canvas"
      },
      {
         "id":192,
         "title":"The Kiss",
         "artist":{
            "name":"Klimt",
            "nationality":"Austria"
         },
         "year":1907,
         "medium":"Oil and gold on canvas"
      },
      {
         "id":139,
         "title":"The Oath of the Horatii",
         "artist":{
            "name":"David",
            "nationality":"France"
         },
         "year":1784,
         "medium":"Oil on canvas"
      }
   ]
}
```

Using JSON in JavaScript

Since the syntax of JSON is the same used for creating objects in JavaScript, it is easy to make use of the JSON format in JavaScript:

```
<script>
var a = {"artist": {"name":"Manet","nationality":"France"}};
alert(a.artist.name + "" + a.artist.nationality);
</script>
```

JSON information will be contained within a string, and the JSON.parse()function can be used to transform the string containing the JSON data into a JavaScript object:

```
var text = '{"artist": {"name":"Manet","nationality":"France"}}';
var a = JSON.parse(text);
alert(a.artist.nationality);
```

The jQuery library also provides a JSON parser that will work with all browsers (the JSON.parse() function is not available on older browsers):

```
var artist = jQuery.parseJSON(text);
```

JavaScript also provides a mechanism to translate a JavaScript object into a JSON string:

```
var text = JSON.stringify(artist);
```

Using JSON in PHP

PHP comes with a JSON extension and as of version 5.2 of PHP; the JSON extension is bundled and compiled into PHP by default. Converting a JSON string into a PHP object is quite straightforward:

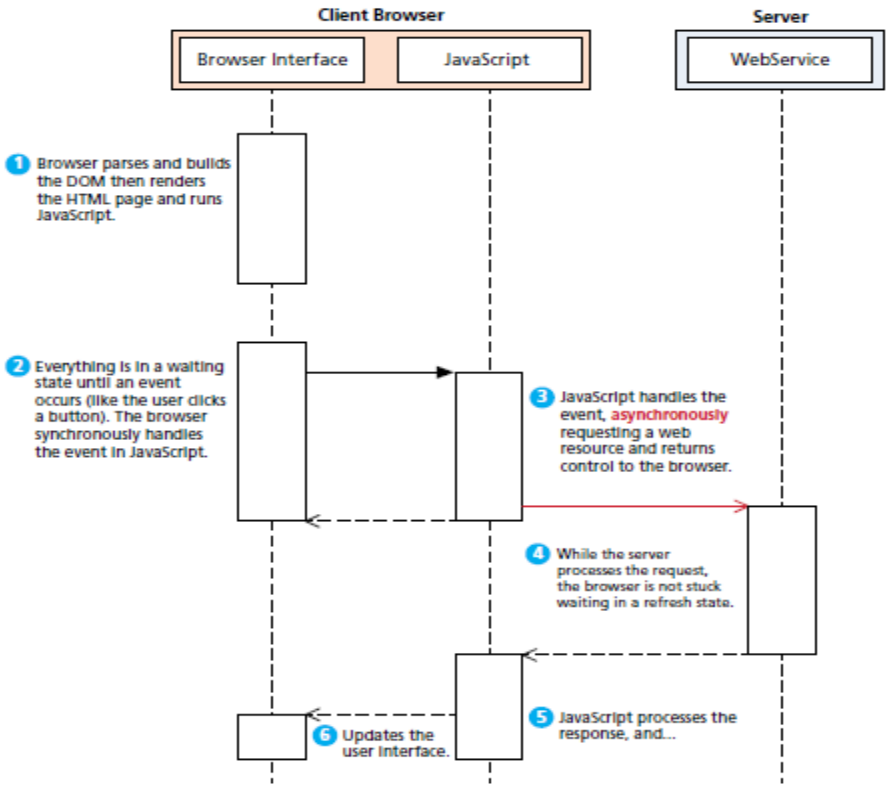
```
<?php
// convert JSON string into PHP object
$text = '{"artist": {"name":"Manet","nationality":"France"}}';
$anObject = json_decode($text);
echo $anObject->artist->nationality;
// convert JSON string into PHP associative array
$anArray = json_decode($text, true);
echo $anArray['artist']['nationality'];
?>
```

json_decode() function can return either a PHP object or an associative array. Since JSON data is often coming from an external source, one should always check for parse errors before using it, which can be done via the json_last_error() function:

```
<?php
// convert JSON string into PHP object
$text = '{"artist": {"name":"Manet","nationality":"France"}}';
$anObject = json_decode($text);
// check for parse errors
if (json_last_error() == JSON_ERROR_NONE) {
echo $anObject->artist->nationality;
}
?>
```

To go the other direction (i.e., to convert a PHP object into a JSON string), you can use the json_encode() function.

```
// convert PHP object into a JSON string
$text = json_encode($anObject);
```

7(a)	Write short notes on AJAX with examples. **AJAX** Asynchronous JavaScript with XML (AJAX) is a term used to describe a paradigm that allows a web browser to send messages back to the server without interrupting the flow of what's being shown in the browser. This makes use of a browser's multi-threaded design and lets one thread handle the browser and interactions while other threads wait for responses to asynchronous requests.  FIGURE 15.8 UML sequence diagram of an AJAX request Responses to asynchronous requests are caught in JavaScript as events. The events can subsequently trigger changes in the user interface or make additional requests. This differs from the typical synchronous requests we have seen thus far, which require the entire web page to refresh in response to a request. Another way to contrast AJAX and synchronous JavaScript is to consider a webpage that displays the current server time as shown in the below figure.	[05]	CO 5	L2

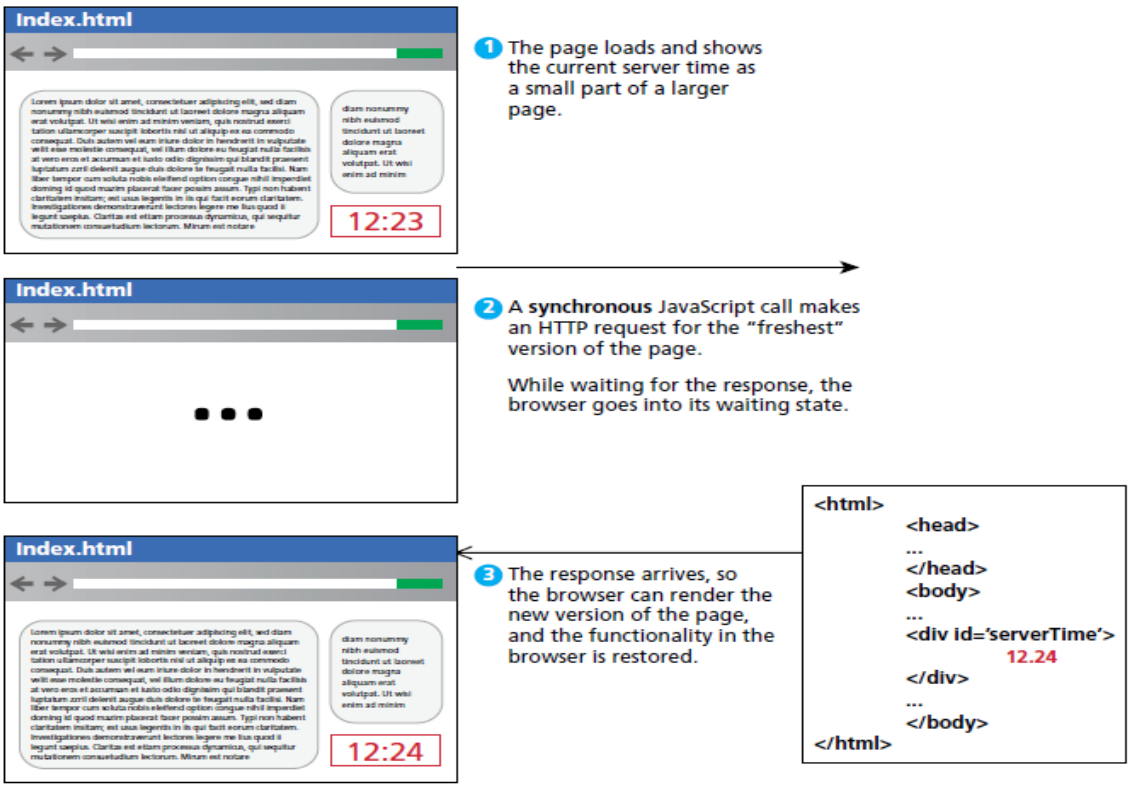


FIGURE 15.9 Illustration of a synchronous implementation of the server time web page.

If implemented synchronously, the entire page has to be refreshed from the server just to update the displayed time. During that refresh, the browser enters a waiting state, so the user experience is interrupted (yes, you could implement a refreshing time using pure JavaScript, but for illustrative purposes, imagine it's essential to see the server's time).

In contrast, consider the very simple asynchronous implementation of the server time, where an AJAX request updates the server time in the background as illustrated in Figure

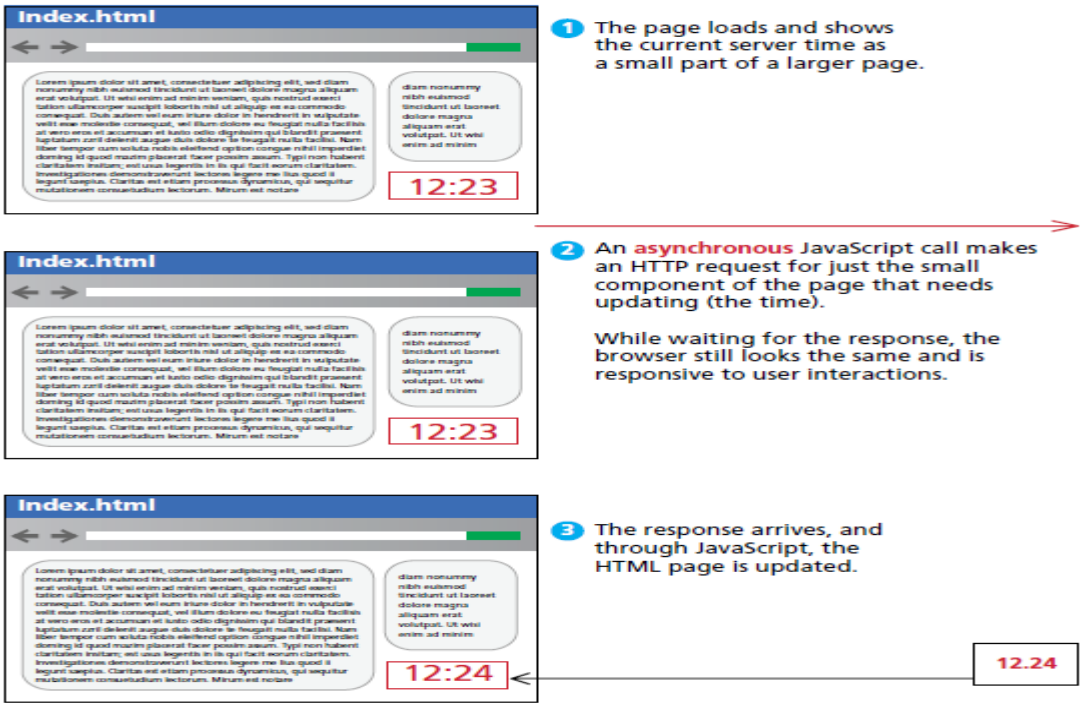


FIGURE 15.10 Illustration of an AJAX implementation of the server time widget

Making Asynchronous Requests

jQuery provides a family of methods to make asynchronous requests. Consider for instance the very simple server time page described above. If the URL currentTime.php returns a single string and you want to load that value asynchronously

into the element, you could write:

```
$("#timeDiv").load("currentTime.php");
```

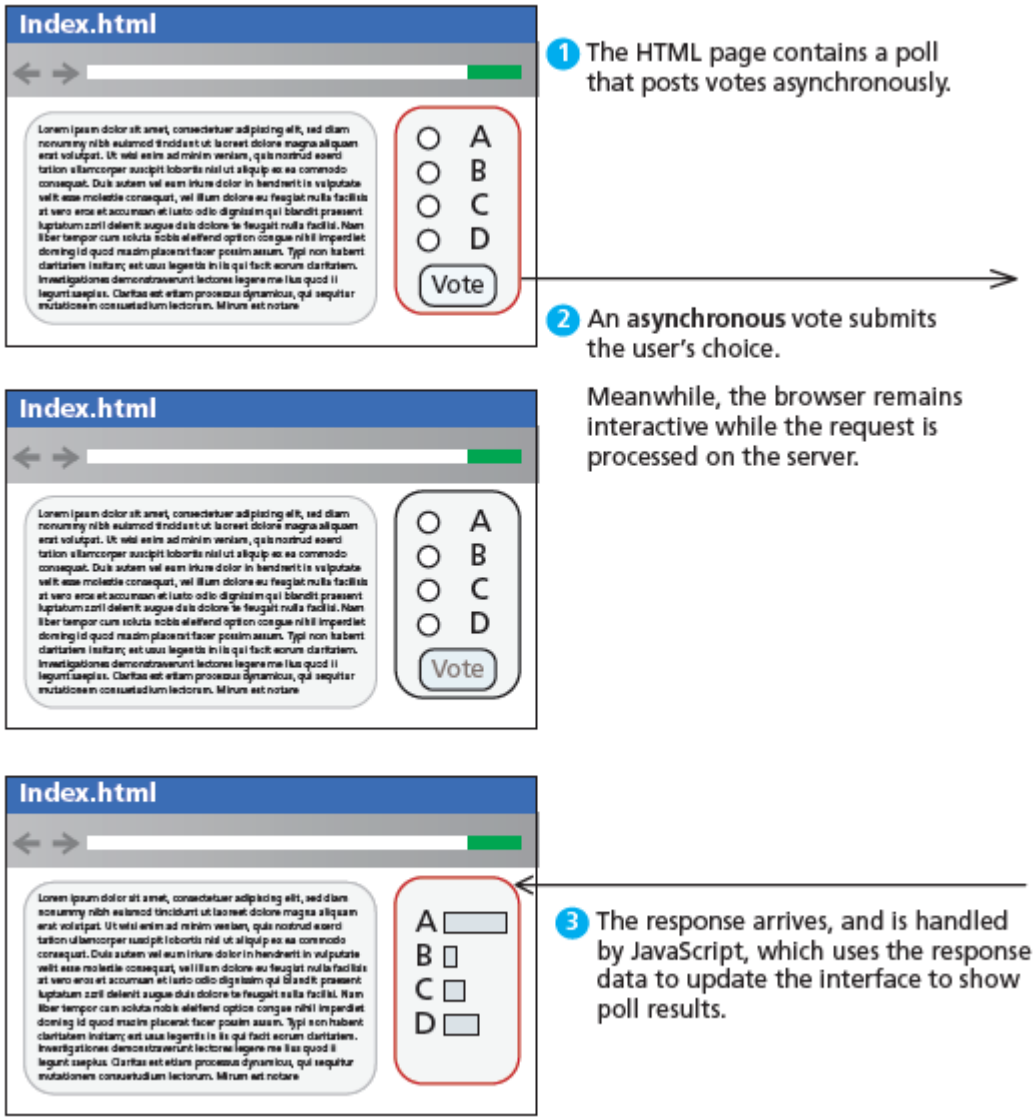


FIGURE 15.11 Illustration of a simple asynchronous web poll

Making a request to vote for option C in a poll could easily be encoded as a URL request GET /vote.php?option=C. However, rather than submit the whole page just to vote in the poll, jQuery's $.get() method sends that GET request asynchronously as follows:

```
$.get("/vote.php?option=C");
```

Note that the $ symbol is followed by a dot. Recall that since $ is actually shorthand for jQuery(), the above method call is equivalent to **jQuery().("/vote.php?option=C");**

7(b)	Explain serialization with examples. **Serialization** **Serialization** is the process of taking a complicated object and reducing it down to zeros and ones for either storage or transmission. Later that sequence of zeros and ones can be reconstituted into the original object.	[05]	CO 5	L2

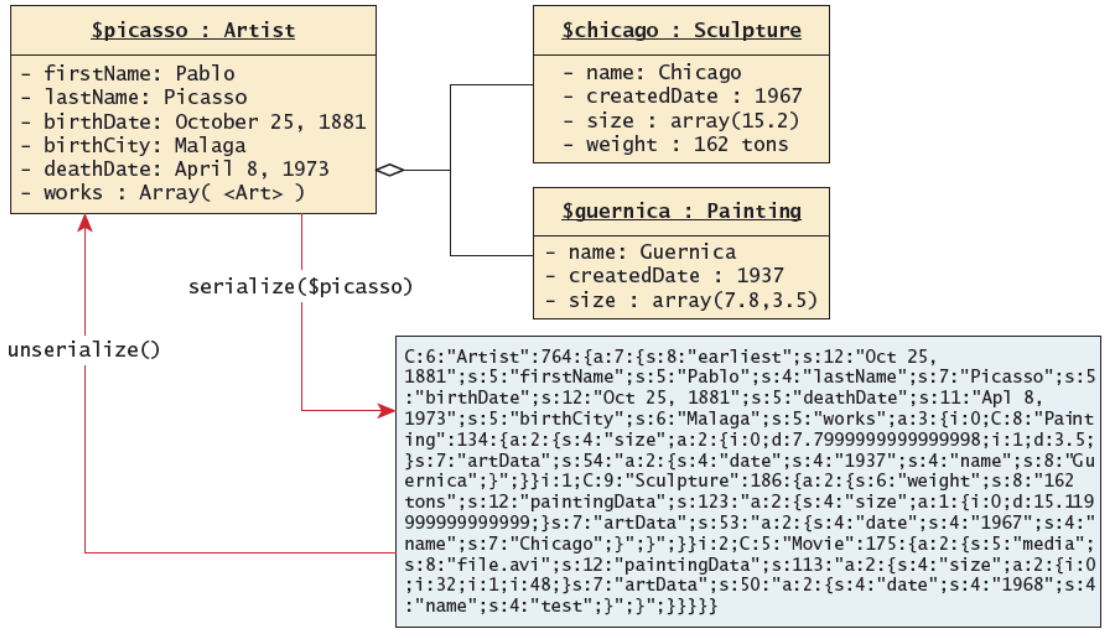


FIGURE 13.7 Serialization and deserialization

In PHP objects can easily be reduced down to a binary string using the **serialize()** function. The resulting string is a binary representation of the object and therefore may contain unprintable characters. The string can be reconstituted back into an object using the unserialize() method. While arrays, strings, and other primitive types will be serializable by default, classes of our own creation must implement the Serializable interface shown below. Which requires adding implementations for serialize() and unserialize() to any class that implements this interface.

```
interface Serializable {
        /* Methods */
        public function serialize();
        public function unserialize($serialized);
        }
```

The example shows how the Artist class must be modified to implement the Serializable interface by adding the implements keyword to the class definition and adding implementations for the two methods.

listing 13.4 Artist class modified to implement the Serializable interface

```
class Artist implements Serializable {
//...
        // Implement the Serializable interface methods
        public function serialize() {
        // use the built-in PHP serialize function
        return serialize(
        array("earliest" =>self::$earliestDate,
        "first" => $this->firstName,
        "last" => $this->lastName,
        "bdate" => $this->birthDate,
        "ddate" => $this->deathDate,
        "bcity" => $this->birthCity,
        "works" => $this->artworks
        );
        );
        }
        public function unserialize($data) {
        // use the built-in PHP unserialize function
        $data = unserialize($data);
        self::$earliestDate = $data['earliest'];
        $this->firstName = $data['first'];
```

```
$this->lastName = $data['last'];
$this->birthDate = $data['bdate'];
$this->deathDate = $data['ddate'];
$this->birthCity = $data['bcity'];
$this->artworks = $data['works'];
}
//...
}
```

If the data above is assigned to $data, then the following line will instantiate a new object identical to the original:

```
$picassoClone = unserialize($data);
```

Application of Serialization

Since each request from the user requires objects to be reconstituted, using serialization to store and retrieve objects can be a rapid way to maintain state between requests. At the end of a request you store the state in a serialized form, and then the next request would begin by deserializing it to reestablish the previous state.

8(a)	How do you pass information through URL and query string?	[05]	CO 4	L2

Passing Information via Query Strings

A web page can pass query string information from the browser to the server using one of the two methods: a query string within the URL (GET) and a query string within the HTTP header (POST). Figure 13.4 reviews these two different approaches.

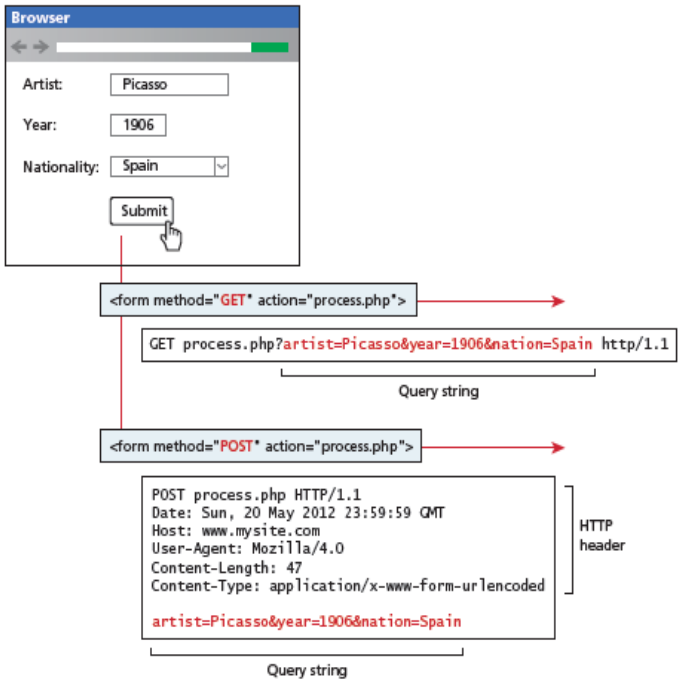


FIGURE 13.4 Recap of GET versus POST

4.3 Passing Information via the URL Path

While query strings are a vital way to pass information from one page to another, they do have a drawback. The URLs that result can be long and complicated. While for many users this is not that important, many feel that for one particular type of user, query strings are not ideal.

	While there is some dispute about whether dynamic URLs (i.e., ones with query string parameters) or static URLs are better from a search engine result optimization (or SEO for search engine optimization) perspective, the consensus is that static URLs do provide some benefits with search engine result rankings. Many factors affect a page's ranking in a search engine, but the appearance of search terms within the URL does seem to improve its relative position. Another benefit to static URLs is that users tend to prefer them. As we have seen, dynamic URLs (i.e., query string parameters) are a pretty essential part of web application development. How can we do without them? The answer is to rewrite the dynamic URL into a static one (and vice versa). This process is commonly called **URL rewriting**. We can try doing our own rewriting. Let us begin with the following URL with its query string information: **www.somedomain.com/DisplayArtist.php?artist=16** One typical alternate approach would be to rewrite the URL to: **www.somedomain.com/artists/16.php** Notice that the query string name and value have been turned into path names. One could improve this to make it more SEO friendly using the following: **www.somedomain.com/artists/Mary-Cassatt** The **mod_rewrite** module uses a rule-based rewriting engine that utilizes Perl compatible regular expressions to change the URLs so that the requested URL can be mapped or redirected to another URL internally.			

8(b)	Write a PHP program to create a class STUDENT with the following specification.
	Data members : Name, Roll number, Average marks
	Member function : Read(getters) and write (setters)
	Use the above specification to read and print the information of 2 students.
	
	`<?php`
	`class MyClass`
	`{`
	`/* Private attribute, cannot be accessed directly */`
	`private $name;`
	
	`/* Constructor */`
	`public function __construct()`
	`{`
	`$this->name = '';`
	`}`
	
	`/* Getter function to read the attribute */`
	`public function get_name()`
	`{`
	`return $this->name;`
	`}`
	
	`/* Setter function to change the attribute */`
	`public function set_name($new_name)`
		[05]	CO 4	L3

```
    {
        if ($this->is_valid_name($new_name))
        {
            $this->name = $new_name;
        }
    }

    /* Checks if the name is valid */
    private function is_valid_name($name)
    {
        $valid = TRUE;

        /* Just checks if the string length is between 3 and 16 */
        if (mb_strlen($name) < 3)
        {
            $valid = FALSE;
        }
        else if (mb_strlen($name) > 16)
        {
            $valid = FALSE;
        }

        return $valid;
    }
}

$mc = new MyClass();
$mc->set_name('Alex');
echo $mc->get_name();
?>
```