USN | | | | | | | | | | |

Internal Assessment Test 5 – Feb. 2022

Sub:	Web Technologies & its applications			Sub Code:	17CS71	Branch:	CSE
Date:	05-02-2022	Duration:	90 min's	Max Marks:	50	Sem / Sec:	7 – D	OBE

Answer any FIVE FULL Questions — MARKS | CO | RBT

1(a) How do you read or write a file on the server from PHP? Give examples. [10] CO3 L2

Reading/Writing Files

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.

Stream Access

functions like fopen(), fclose(), and fgets() from the C programming language,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 9.19 illustrates a script using fopen(), fgets(), and fclose() to read a file and echo it out (replacing new lines with <br> tags).

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

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.

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.

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

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);
```

let us imagine we have a comma-delimited text file that contains information about paintings, where each line in the file corresponds to a different painting:

```
01070,Picasso,The Actor,1904
01080,Picasso,Family of Saltimbanques,1905
02070,Matisse,The Red Madras Headdress,1907
05010,David,The Oath of the Horatii,1784
```

```
// read the file into memory; if there is an error then stop processing
$paintings = file($filename) or die('ERROR: Cannot find file');
```

```
// our data is comma-delimited
$delimiter = ',';
// loop through each line of the file
foreach ($paintings as $painting) {
// returns an array of strings where each element in the array
// corresponds to each substring between the delimiters
$paintingFields = explode($delimiter, $painting);
$id= $paintingFields[0];
$artist = $paintingFields[1];
$title = $paintingFields[2];
$year = $paintingFields[3];
// do something with this data
. . .
}
```

2(a) Explain selectors in JQuery with examples. [10] CO5 L2

Basic Selectors

The four basic selectors were defined back in Chapter 3, 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 <div> 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.
The implementation of selectors in jQuery purposefully mirrors the CSS specification,
which is especially helpful since CSS is something you have learned and
used throughout this book.
In addition to these basic selectors, you can use the other CSS selectors that
were covered in Chapter 3: attribute selectors, pseudo-element selectors, and contextual
selectors as illustrated in Figure 15.4. The remainder of this section reviews
some of these selectors and how they are used with jQuery.

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. Chapter 3 mentioned that not all

```
<body>
<nav>
<ul>
<li><a href="#">Canada</a></li>
<li><a href="#">Germany</a></li>
<li><a href="#">United States</a></li>
</ul>
</nav>
<div id="main">
Comments as of <time>November 15, 2012</time>
<div>
<p>By Ricardo on <time>September 15, 2012</time></p>
<p>Easy on the HDR buddy.</p>
</div>
<hr/>
<div>
<p>By Susan on <time>October 1, 2012</time></p>
<p>I love Central Park.</p>
</div>
<hr/>
</div>
<footer>
<ul>
<li><a href="#">Home</a> | </li>
<li><a href="#">Browse</a> | </li>
</ul>
</footer>
</body>
$("ul a:link")
$("#main>time")
```

$("#main time")
$("#main div p:first_child")
Figure 15.4 Illustration of some jQuery selectors and the HTML being selected

browsers implemented it. jQuery overcomes those browser limitations, providing
the ability to select elements by attribute. 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 in
the 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. As
you may recall from Chapter 3, 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");
Since this chapter reviews and builds on CSS selectors, you are hopefully remembering
some of the selectors you have used earlier and are making associations between
those selectors and the ones in jQuery. As you already know from Chapter 6, once you
have the ability to select an element, you can do many things to manipulate that element
from changing its content or style all the way to removing it.

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 <div> elements you would write var para = $("div p");			
3(a)	Write short notes on the following PHP super global arrays. a) $_GET b) $_POST c) $_SERVER 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. They are called superglobal because these arrays are always in scope and always exist, ready for the programmer to access or modify them without having to use the global keyword. 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 4.2.1 $_GET and $_POST Super global Arrays 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.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. 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. In PHP, there are several techniques to accomplish this task. First, you can	[10]	CO4	L2

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.

4(a) How cookies and session work? Give examples. [10] CO2 L2

While cookie information is stored and retrieved by the browser, the information ina 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 acookie (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 acookie does not have an expiry date specified, the browser will delete it when thebrowser closes (or the next time it accesses the site). For this reason, some commentatorswill say that there are two types of cookiessession cookies and persistentcookies. A **session cookie** has no expiry stated and thus will be deleted at the end ofthe user browsing session. **Persistent cookies** have an expiry date specified; they willpersist in the browser's cookie file until the expiry date occurs, after which they aredeleted.

The most important limitation of cookies is that the browser may be configuredto refuse them. As a consequence, sites that use cookies should not depend on theiravailability for critical features. Similarly, the user can also delete cookies or eventamper with the cookies, which may lead to some serious problems if not handled.

Several years ago, there was an instructive case of a website selling stereos and televisionsthat used a cookie-based shopping cart. The site placed not only the productidentifier but also the product price in the cart. Unfortunately, the site then used theprice in the cookie in the checkout. Several curious shoppers edited the price in thecookie stored on their computers, and then purchased some big-screen televisionsfor only a few cents!

Using Cookies

Like any other web development technology, PHP provides mechanisms for writingand reading cookies. Cookies in PHP are *created* using the setcookie() functionand 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 furthercustomize the new cookie. You can examine the online official PHP documentationfor more information. The below example illustrates the reading of cookie values. Notice that when we reada cookie, we must also check to ensure that the cookie exists. In PHP, if the cookiehas 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'];
}
?>
```

Persistent Cookie Best Practices

Many sites provide a "Remember Me" checkbox on login forms, which relies onthe use of a persistent cookie. This login cookie would contain the user's usernamebut not the password. Instead, the login cookie would contain a random token; thisrandom token would be stored along with the username in the site's back-end database.Every time the user logs in, a new token would be generated and stored in thedatabase and cookie.

Another common, nonessential use of cookies would be to use them to storeuser preferences. For instance, some sites allow the user to choose their preferredsite color scheme or their country of origin or site language. In these cases, savingthe user's preferences in a cookie will make for a more contented user, but if theuser's browser does not accept cookies, then the site will still work just fine; at worstthe user will simply have to reselect his or her preferences again.

Another common use of cookies is to track a user's browsing behavior on a site.Some sites will store a pointer to the last requested page in a cookie; this informationcan be used by the site administrator as an analytic tool to help understand howusers navigate through the site.

5 (a)	Explain the following PHP OOPS concept with examples. a) Data Encapsulation b) Inheritance c) Polymorphism	[10]	CO4	L2

- Another way of understanding encapsulation is: it is the **hiding of an object's implementation details**.
- Properly encapsulated class will define its properties, hidden (that is, private).
- Accessing and modifying such a properties will be done through by writing methods 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 methods 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:

```
  private $firstName;
```

Can represent getter method to return the value as:

```
public function getFirstName()
 {
                              return $this->firstName;
}
```

- Setter methods **modify properties**, and allow extra logic to be added to prevent properties from being set to strange values.
- For instance:

```
  private $firstName;
```

Can represent getter method to return the value as:

```
public function setFirstName($firstName)
 {
                              $this->firstName=$firstName;
}
```

	• Two forms of the updated **UML class** diagram for our data encapsulated class. • The longer one includes all the **getter and setter** methods. • It is quite common, however, to exclude the getter and setter methods from a class.			

6 (a)	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.
	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.
	Class STUDENT{
	Public $Name;
	Public $Roll-number
	Public $Average-marks
	}
	$student1=new STUDENT()
	$student2=new STUDENT()
		[05]	CO4	L3

6 (b)	Write short notes on Cookies with examples While cookie information is stored and retrieved by the browser, the information ina 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 acookie (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 acookie does not have an expiry date specified, the browser will delete it when thebrowser closes (or the next time it accesses the site). For this reason, some commentatorswill say that there are two types of cookiessession cookies and persistentcookies. A **session cookie** has no expiry stated and thus will be deleted at the end ofthe user browsing session. **Persistent cookies** have an expiry date specified; they	[05]	CO4	L3

willpersist in the browser’s cookie file until the expiry date occurs, after which they aredeleted.

The most important limitation of cookies is that the browser may be configuredto refuse them. As a consequence, sites that use cookies should not depend on theiravailability for critical features. Similarly, the user can also delete cookies or eventamper with the cookies, which may lead to some serious problems if not handled.

Several years ago, there was an instructive case of a website selling stereos and televisionsthat used a cookie-based shopping cart. The site placed not only the productidentifier but also the product price in the cart. Unfortunately, the site then used theprice in the cookie in the checkout. Several curious shoppers edited the price in thecookie stored on their computers, and then purchased some big-screen televisionsfor only a few cents!

Using Cookies

Like any other web development technology, PHP provides mechanisms for writingand reading cookies. Cookies in PHP are *created* using the setcookie() functionand 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 furthercustomize the new cookie. You can examine the online official PHP documentationfor more information. The below example illustrates the reading of cookie values. Notice that when we reada cookie, we must also check to ensure that the cookie exists. In PHP, if the cookiehas 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'];

}

?>
```

Persistent Cookie Best Practices

Many sites provide a “Remember Me” checkbox on login forms, which relies onthe use of a persistent cookie. This login cookie would contain the user’s usernamebut not the password. Instead, the login cookie would contain a random token; thisrandom token would be stored along with the username in the site’s back-end database.Every time the user logs in, a new token would be generated and stored in thedatabase and cookie.

Another common, nonessential use of cookies would be to use them to storeuser preferences. For instance, some sites allow the user to choose their preferredsite color scheme or their country of origin or site language. In these cases, savingthe user’s preferences in a cookie will make for a more contented user, but if theuser’s browser does not accept cookies, then the site will still work just fine; at worstthe user will simply have to reselect his or her preferences again.

Another common use of cookies is to track a user’s browsing behavior on a site.Some sites will store a pointer to the last requested page in a cookie; this informationcan be used by the site administrator as an analytic tool to help understand howusers navigate through the site.