CMR
INSTITUTE OF
TECHNOLOGY

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

Internal Assessment Test2 Answer Key– – April. 2019

Sub:	Advanced Web Programming					Sub Code:	17MCA42	Branch:	MCA
Date:	15.04.2019	Duration:	90 min's	Max Marks:	50	Sem	IV		OBE

1 How to create an array in PHP? Explain different functions for dealing with arrays. [10]

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- Numeric array − An array with a numeric index. Values are stored and accessed in linear fashion.
- Associative array − An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- Multidimensional array − An array containing one or more arrays and values are accessed using multiple indices

Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

Example

Following is the example showing how to create and access numeric arrays. Here we have used array() function to create array. This function is explained in function reference.

```
<html>
    <body>
    <?php
    /* First method to create array. */
    $numbers = array( 1, 2, 3, 4, 5);
    foreach( $numbers as $value ) {
    echo "Value is $value <br />";
    }
    /* Second method to create array. */
    $numbers[0] = "one";
    $numbers[1] = "two";
    $numbers[2] = "three";
    $numbers[3] = "four";
    $numbers[4] = "five";
    foreach( $numbers as $value ) {
    echo "Value is $value <br />";
    }
    ?>
    </body>
</html>
```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

```
<html>
	<body>
	<?php
	/* First method to associate create array. */
	$salaries = array("mohammad" => 2000, "qadir" => 1000, "zara" => 500);
	echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
	echo "Salary of qadir is ". $salaries['qadir']. "<br />";
	echo "Salary of zara is ". $salaries['zara']. "<br />";
	/* Second method to create array. */
	$salaries['mohammad'] = "high";
	$salaries['qadir'] = "medium";
	$salaries['zara'] = "low";
	echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
	echo "Salary of qadir is ". $salaries['qadir']. "<br />";
	echo "Salary of zara is ". $salaries['zara']. "<br />";
	?>
	</body>
</html>
```

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key Sort Array in Ascending Order - sort()

2 What is the difference between cookies and sessions? How to create a cookie and sessions? Give an example [10]

This is a suitable method for tracking users. Web servers are stateless entities.

Cookies :-

Cookies provide a way for a server to store information about a user on the user's machine. This enables to maintain the user's visit to the site , so that they can track their movement through the site , or to store information such as their user name .

Cookies allow data to be stored in the form of a name/value pair. Both the name and the value are set at choice.

It contains the following syntax

Name=value;expires=expiration date gmt; path=url(optional)

To create a cookie

Setcookie(name,value,expires,path);

Cookie name/value pair: The first section of the cookie defines the name of the cookie and the value assigned to the cookie. Both the name and value settings can be any value as per the users choice.

Cookie Expiration time: The optional expires= section specifies the date on which the associated cookie should expire. The PHP time() can be used to obtain and manipulate dates for this purpose

Example:

```
<?php
Setcookie('username','abcd',time()+4800);
Echo "cookie has been set";
?>
```

The cookie will expire after 4800 seconds.

Cookies are sent in the HTTP headers in pages sent by the browser . Once the cookies has been set they can be accessed on the next page load with the $_COOKIE array. This is an

associative array where the name of the cookie provides the index into the array to extract the corresponding value of the name/value pair.

Example:

```
<?php
Echo "cookie value is ".$_COOKIE['username'];
?>
```

To delete a cookie

Cookies are deleted by calling the setcookie() function with the cookie name , a null for the value and an expiration date in the past.

```
<?php
Setcookie("username","",time()-4800);
?>
```

Sessions

A session is a way to store information (in variables) to be used across multiple pages. A session creates a file in a temporary directory on the server where registered session variables and their values are stored. The data will be available to all pages on the site during the visit.

A session like a cookie provides a way to track data for a user over a series of pages.

The main difference between cookie and session is that cookie stores the data on the client side in the web browser whereas the session data is stored on the server.

Sessions are generally more secure, because the data is not transmitted back and forth between the client and server repeatedly.

Sessions let you store more information than you can in cookie.

When session starts, PHP generates a random session id , a reference to that particular session and its stored data.

To start a session

```
Session_start();
```

Session variables are set with the PHP global variables $_SESSION.

```
<?php
Session_start();
$_SESSION["username"] = "abc";
?>
```

To delete the session variables , we unset $_SESSION array

```
Unset($_SESSION["username"]);
```

To remove session data from the server where it is stored in the temporary file, we use

```
Session_destroy();
```

3 Explain pattern matching in PHP. [10]

PHP includes two different kinds of string pattern matching using regular expressions: one that is based on POSIX regular expressions and one that is based on Perl regular expressions. The POSIX regular expressions are compiled into PHP, but the Perl-Compatible Regular Expression (PCRE) library must be compiled before Perl regular expressions can be used. A detailed discussion of PHP pattern matching is beyond the scope of this chapter. Furthermore, Perl-

The preg_match[6] function takes two parameters, the first of which is the Perl-style regular expression as a string. The second parameter is the string to be searched. For example:

```
if (preg_match("/^PHP/", $str))
  print "\$str begins with PHP <br />";
else
  print "\$str does not begin with PHP <br />";
```

The preg_split function operates on strings but returns an array and uses patterns, so it is discussed here rather than with the other string functions in Section 11.4.7. preg_split takes two parameters, the first of which is a Perl-style pattern as a string. The second parameter is the string to be split.[7] For example, consider the following sample code:

```
$fruit_string = "apple : orange : banana";
$fruits = preg_split("/ : /", $fruit_string);
```

The array $fruits now has ("apple", "orange", "banana").

The following example illustrates the use of preg_split on text to parse out the words and produce a frequency-of-occurrence table:

```
//           the keys and their frequencies are the values.
function splitter($str) {

// Create the empty word frequency array
  $freq = array();

// Split the parameter string into words
  $words = preg_split("/[\.,;:!\?]\s*/", $str);

// Loop to count the words (either increment or initialize to 1)
  foreach ($words as $word) {
    $keys = array_keys($freq);
    if(in_array($word, $keys))
      $freq[$word]++;
    else
      $freq[$word] = 1;
  }
  return $freq;
} #** End of splitter

// Main test driver
  $str = "apples are good for you, or don't you like apples?
          or maybe you like oranges better than apples";

// Call splitter
  $tbl = splitter($str);

// Display the words and their frequencies
  print "<br /> Word Frequency <br /><br />";
  $sorted_keys = array_keys($tbl);
  sort($sorted_keys);
  foreach ($sorted_keys as $word)
    print "$word $tbl[$word] <br />";
```

4 Explain all the file operations in PHP. [10]

File handling is needed for any application. For some tasks to be done file needs to be processed. File handling in PHP is similar as file handling is done by using any programming language like C. PHP has many functions to work with normal files. Those functions are:

1) fopen() – PHP fopen() function is used to open a file. First parameter of fopen() contains name of the file which is to be opened and second parameter tells about mode in which file needs to be opened, e.g.,

```
<?php
     $file = fopen("demo.txt",'w');
?>
```

Files can be opened in any of the following modes :

- **"w" –** Opens a file for write only. If file not exist then new file is created and if file already exists then contents of file is erased.
- **"r" –** File is opened for read only.
- **"a" –** File is opened for write only. File pointer points to end of file. Existing data in file is preserved.
- **"w+" –** Opens file for read and write. If file not exist then new file is created and if file already exists then contents of file is erased.
- **"r+" –** File is opened for read/write.
- **"a+" –** File is opened for write/read. File pointer points to end of file. Existing data in file is preserved. If file is not there then new file is created.
- **"x" –** New file is created for write only.

2) fread() –– After file is opened using fopen() the contents of data are read using fread(). It takes two arguments. One is file pointer and another is file size in bytes, e.g.,

```
<?php
$filename = "demo.txt";
$file = fopen( $filename, 'r' );
$size = filesize( $filename );
$filedata = fread( $file, $size );
?>
```

3) fwrite() – New file can be created or text can be appended to an existing file using fwrite() function. Arguments for fwrite() function are file pointer and text that is to written to file. It can contain optional third argument where length of text to written is specified, e.g.,

```
<?php
$file = fopen("demo.txt", 'w');
$text = "Hello world\n";
fwrite($file, $text);
?>
```

4) fclose() – file is closed using fclose() function. Its argument is file which needs to be closed, e.g.,

```
<?php
$file = fopen("demo.txt", 'r');
//some code to be executed
fclose($file);
?>
```

5 Explain all string methods in Ruby. [10]

The Ruby `String` class has more than 75 methods, a few of which are described in this section. Many of these methods can be used as if they were operators. In fact, we sometimes call them operators, even though underneath they are all methods.

The `String` method for catenation is specified by plus (+), which can be used as a binary operator. This method creates a new string from its operands. For example:

```
>> "Happy" + " " + "Holidays!"
=> "Happy Holidays!"
```

To append a string to the right end of another string, which of course only makes sense if the left operand is a variable, use the << method. Like +, the << method can be used as a binary operator. For example:

```
>> mystr = "G'day "
=> "G'day "
>> mystr << "mate"
=> "G'day mate"
```

The first assignment above creates the specified string literal and sets the variable `mystr` to reference that memory location. If `mystr` is assigned to another variable, that variable will reference the same memory location as `mystr`. For example:

```
>> mystr = "Wow!"
=> "Wow!"
>> yourstr = mystr
=> "Wow!"
>> yourstr
=> "Wow!"
```

Now, both `mystr` and `yourstr` reference the same memory location, the place that has the string `"Wow!"`. If a different string literal is assigned to `mystr`, Ruby will build a memory location with the value of the new string literal and `mystr` will reference that location. But `yourstr` will still reference the location with `"Wow!"`. For example:

```
>> mystr = "Wow!"
=> "Wow!"
>> yourstr = mystr
=> "Wow!"
>> mystr = "What?"
=> "What?"
>> yourstr
=> "Wow!"
```

If you want to change the value of the location that `mystr` references, but let `mystr` reference the same memory location, the `replace` method is used. For example:

```
>> mystr = "Wow!"
=> "Wow!"
>> yourstr = mystr
=> "Wow!"
>> mystr.replace("Golly!")
=> "Golly!"
>> mystr
=> "Golly!"
>> yourstr
=> "Golly!"
```

Now, `mystr` and `yourstr` still reference the same memory location.

The append operation can also be done with the += assignment operator. So, instead of `mystr << "mate"`, `mystr += "mate"` could be used.

In the following paragraphs, other string functions will be introduced that also change a string value but leave the affected variable referencing the same memory location.

The other most commonly used methods are similar to those of other programming languages. Among these are the following, all of which create new strings:

Method	Action
`capitalize`	Convert the first letter to uppercase and the rest of the letters to lowercase
`chop`	Removes the last character
`chomp`	Removes a newline from the right end, if there is one
`upcase`	Converts all of the lowercase letters in the object to uppercase
`downcase`	Converts all of the uppercase letters in the object to lowercase
`strip`	Removes the spaces on both ends
`lstrip`	Removes the spaces on the left end
`rstrip`	Removes the spaces on the right end
`reverse`	Reverses the characters of the string
`swapcase`	Convert all uppercase letters to lowercase and all lowercase letters to uppercase

As stated previously, all of these produce new strings, rather than modify the given string in place. However, all of these methods have versions that do modify their objects in place. These are called *bang* or *mutator* methods, which are specified by following their names with an exclamation point (!). To illustrate the difference between a string method and its bang counterpart, consider the following interactions:

```
>> str = "Frank"
=> "Frank"
>> str.upcase
=> "FRANK"
```

```
>> str
=> "Frank"
>> str.upcase!
=> "FRANK"
>> str
=> "FRANK"
```

Note that after using `upcase`, the value of `str` is unchanged (it is still `"Frank"`), but after using `upcase!`, it is changed (it is `"FRANK"`).

Ruby strings can be indexed, somewhat as if they were arrays. As one would expect, the indices begin at zero. The brackets of this method specify a getter method. The catch with this getter method is that it returns the ASCII code (as a `Fixnum` object), rather than the character. To get the character, the `chr` method must be called. Consider the following example:

```
>> str = "Shelley"
=> "Shelley"
>> str[3]
=> 108
>> str[3].chr
=> "l"
```

If a negative subscript is used as an index, the position is counted from the right.

A multicharacter substring of a string can be accessed by including two numbers in the brackets, in which case the first is the position of the first character of the substring and the second is the number of characters in the substring. Unlike the single-character reference, however, in this case the value is a string, not a number. For example:

```
>> str = "Shelley"
=> "Shelley"
>> str[2,4]
=> "elle"
```

The substring getter can be used on individual characters to get one character without calling the `chr` method.

Specific characters of a string can be set with the set method, `[]=`. For example:

```
>> str = "Donald"
=> "Donald"
>> str[3,3] = "nie"
=> "nie"
>> str
=> "Donnie"
```

6 With an example, explain the built in methods for arrays and lists in Ruby [10]

Frequently it is necessary to place new elements on one end or the other of an array. Ruby has four methods for this purpose: `unshift` and `shift`, which deal with the left end of arrays; and `pop` and `push`, which deal with the right end of arrays.

The `shift` method removes and returns the first element (lowest subscript) of the array object to which it is sent. For example, the following statement removes the first element of `list` and places it in `first`:

```
>> list = [3, 7, 13, 17]
=> [3, 7, 13, 17]
>> first = list.shift
=> 3
>> list
=> [7, 13, 17]
```

The subscripts of all of the other elements in the array are reduced by 1 as a result of the `shift` operation.

The `pop` method removes and returns the last element from the array object to which it is sent. In this case, there is no change in the subscripts of the array's other elements.

The `unshift` method takes a scalar or an array literal as a parameter. The scalar or array literal is appended to the beginning of the array. This results in an increase in the subscripts of all other array elements. The `push` method also takes a scalar or an array literal. The scalar or array is added to the high end of the array:

```
>> list = [2, 4, 6]
=> [2, 4, 6]
>> list.push(8, 10)
=> {2, 4, 6, 8, 10]
```

Either `pop` and `unshift` or `push` and `shift` can be used to implement a queue in an array, depending on the direction the queue should grow.

While `push` is a convenient way to add literal elements to an array, if an array is to be catenated to the end of another array, another method, `concat`, is used. For example:

```
>> list1 = [1, 3, 5, 7]
=> [1, 3, 5, 7]
>> list2 = [2, 4, 6, 8]
=> [2, 4, 6, 8]
>> list1.concat(list2)
=> [1, 3, 5, 7, 2, 4, 6, 8]
```

If two arrays need to be catenated together and the result saved as a new array, the plus (+) method can be used as a binary operator, as in the following:

```
>> list1 = [0.1, 2.4, 5.6, 7.9]
=> [0.1, 2.4, 5.6, 7.9]
>> list2 = [3.4, 2.1, 7.5]
=> [3.4, 2.1, 7.5]
>> list3 = list1 + list2
=> [0.1, 2.4, 5.6, 7.9, 3.4, 2.1, 7.5]
```

Note that neither `list1` nor `list2` are affected by the plus method.

The `reverse` method does what its name implies. For example:

```
>> list = [2, 4, 8, 16]
=> [2, 4, 8, 16]
>> list.reverse
=> [16, 8, 4, 2]
```

The `include?` predicate method searches an array for a specific object. For example:

```
>> list = [2, 4, 8, 16]
```

The `sort` method sorts the elements of an array, as long as Ruby knows how to compare those elements. The most commonly sorted elements are either numbers or strings and Ruby knows how to compare numbers with numbers and strings with strings. So, `sort` works well on arrays of elements of either of these two types. For example:

```
>> list = [16, 8, 4, 2]
=> [16, 8, 4, 2]
>> list.sort
=> [2, 4, 8, 16]
>> list2 = ["jo", "fred", "mike", "larry"]
=> ["jo", "fred", "mike", "larry"]
>> list2.sort
=> ["fred", "jo", "larry", "mike"]
```

If the `sort` method is sent to an array that has mixed types, Ruby produces an error message indicating the comparison failed. For example:

```
>> list = [2, "jo", 8, "fred"]
=> [2, "jo", 8, "fred"]
>> list.sort
ArgumentError: comparison of Fixnum with String failed
        from (irb):13:in 'sort'
        from (irb):13
        from :0
```

`sort` returns a new array and does not change the array to which it is sent. The mutator method, `sort!`, sorts the array to which it is sent, in place.

In some situations, arrays represent sets. There are three methods that perform set operations on two arrays. All are used as binary infix operators. They are `&`, for set intersection, `-`, for set difference, and `|`, for set union. Consider the following examples:

```
>> set1 = [2, 4, 6, 8]
=> [2, 4, 6, 8]
>> set2 = [4, 6, 8, 10]
=> [4, 6, 8, 10]
>> set1 & set2
=> [4, 6, 8]
>> set1 - set2
=> [2]
>> set1 | set 2
=> [2, 4, 6, 8, 10]
```

7 Write a Ruby function to swap two numbers [10]

```
def swap(a,b)
        temp = a
        a = b
        b = temp
        return a,b
end

puts  “Enter first number”
gets a
puts  “Enter second  number”
gets b

var= swap(a,b)
```

puts “Sorted values are #{var}”

8 Write a PHP program to insert name and age information entered by the user into a table created using MySQL and to display the current contents of this table. [10]

```
<html>
<body>
<form action="http://localhost//WEBDEMO/3.php" method="POST">
<table align="center" width="100" height="100">
<caption> Student Information</caption>
<tr><td>Name:</td><td><input type="text" name="name"></td></tr>
<tr><td>Age :</td><td><input type="text" name="age"></td></tr>
<tr><td></td><td><input type="submit" value="Submit"></td></tr></table></form>
</body>
</html>
```

```
<?php
$name=$_POST['name'];
$age=$_POST['age'];
mysql_connect("localhost","root","");
mysql_select_db("web1");
mysql_query("insert into p3(name,age) values('$name','$age')");

$result=mysql_query("select * from p3 where name='$name' and age='$age'");
$row=mysql_fetch_row($result);

?>
<table>
<tr>
<th>Name</th>
<th>age</th>
</tr>
<tr>
<td><?php echo $row[0];?></td>
<td><?php echo $row[1];?></td>
</tr>
</table>
```