PHP

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor"(PHP originally meant Personal Home Page).
- PHP is a server-side scripting language that is embedded in HTML.
- PHP is an interpreted language, in that the PHP interpreter program reads the PHP source code, translates the code and executes it at the same time.
- PHP is dynamically typed (type checking at run-time).
- Loosely Typed Language (PHP supports variable usage without declaring its data type.)
- USES:
  - Perform system functions (PHP can create, open, read, write, and close files).
  - Form handling (Collect data from HTML forms, validate form data, send form data to database etc.).
  - Database access (You add, delete, modify elements within your database through PHP).
  - Access cookies and session variables.
  - Using PHP, you can restrict users to access some pages of your website.
  - It can encrypt data.
  - Creating dynamic web pages.
- When a client requests a document that has embedded PHP, the Web server calls a PHP processor which copies XML code and client-side scripts verbatim to an output document as it finds them, Interprets the PHP scripts when they occur and writes the PHP script output to the output document
  The Web server then sends the output XHTML document to the client.
- The output of a PHP script is an XHTML page.

General Syntactic Characteristics

- PHP code can be specified in an HTML document internally or externally.
- Internally: [opening tag: <?php and closing tag: ?>]

```
<?php
// php code goes here

?>
```

- Externally: [using include keyword]

```
include ("myScript.php");      //the file can have both PHP and HTML code
```

- **All variable names begin with dollar sign ($).**
  Example: $a=10; $s="hello";
- Types of commenting styles
  - // and # for single line comments
  - /* */ for multiline comments
- PHP statements are terminated with semicolons.

Primitives, Operations, and Expressions

- Because PHP is dynamically typed, it has no type declarations.
- Type of variable is set every time it is assigned a value.
- **Primitive data types**:
    - Four scalar types - **Integer, double, string, Boolean**
    - Two compound types- **array and object**
    - Two special types - **resource and null**
- Variables are not declared They are dynamically typed.
- Variable names are case sensitive
- Unassigned variables are also called unbounded variables and have type NULL.
- A resource is a special variable, holding a reference to an external resource. Resources are created and used by special functions.
- Integer type corresponds to the long type of C and its successors.
- Double type corresponds to the double type of C and its successors. Double literals can include a decimal point, an exponent, or both.
- Characters in PHP are single bytes; UNICODE is not supported.
- A single character data value is represented as a string of length 1.
- **String literals** are defined with either **single-quote (') or double-quote (")** delimiters.
    - In single-quoted string literals, escape sequences, such as \n, are not recognized as anything special and the values of embedded variables are not substituted for their names.
    - In double-quoted string literals, escape sequences are recognized and embedded variables are replaced by their current values.

  Example:
  If $sum=2.5;

  Then 'the sum is $sum' produces the output: the sum is $sum
  But "the sum is $sum" produces the output: the sum is 2.5

  Inside single quotes
  • There is no interpolation
  • Escaped characters such as \n and variables are not replaced by their values
  Inside double quotes
  • Variable values are substituted for their names
  • Escaped characters are replaced by their values

- The only two possible values for the Boolean type are TRUE and FALSE, both of which are case insensitive.
    - . If an integer expression is used in a Boolean context, it evaluates to FALSE if it is zero; otherwise, it is TRUE.
    - If a string expression is used in a Boolean context, it evalutes to FALSE if it is either the empty string or the string "0"; otherwise, it is TRUE.
    - This implies that the string "0.0" evaluates to TRUE.

- PHP Arithmetic operations
  - arithmetic operators : +, -, *, /, %, ++, and –
  - Assignment operators
    =, +=, -=, *=, /=, %=
  - Comparison operators
    = =, !=, <>, >, < , >=, <=, = = =
  - Logic operators
    &&, | |, !

  - Predefined functions:

Function	Parameter Type	Returns
`floor`	Double	Largest integer less than or equal to the parameter
`ceil`	Double	Smallest integer greater than or equal to the parameter
`round`	Double	Nearest integer
`srand`	Integer	Initializes a random-number generator with the parameter
`rand`	Two numbers	A pseudorandom number greater than the first parameter and smaller than the second
`abs`	Number	Absolute value of the parameter
`min`	One or more numbers	Smallest
`max`	One or more numbers	Largest

  - String functions:

Function	Parameter Type	Returns
`strlen`	A string	The number of characters in the string
`strcmp`	Two strings	Zero if the two strings are identical, a negative number if the first string belongs before the second (in the ASCII sequence), or a positive number if the second string belongs before the first
`strpos`	Two strings	The character position in the first string of the first character of the second string if the second string is in the first string; `false` if it is not there
`substr`	A string and an integer	The substring of the string parameter, starting from the position indicated by the second parameter; if a third parameter (an integer) is given, it specifies the length of the returned substring
`chop`	A string	The parameter with all white-space characters removed from its end
`trim`	A string	The parameter with all white-space characters removed from both ends
`ltrim`	A string	The parameter with all white-space characters removed from its beginning
`strtolower`	A string	The parameter with all uppercase letters converted to lowercase
`strtoupper`	A string	The parameter with all lowercase letters converted to uppercase

- **Concatenation**: Concatenation is done with **dot(.)** .

  Example:

  ```
  $str1 = "Hello!";
  $str2 = "What's up?";
  $str3 = $str1 . " " . $str2;
  ```

- **Scalar Type Conversions:**

  Explicit conversion:

  if $total=4.777

  - (int)$total => 4
  - intval($total) => 4
  - settype($total, "integer") => 4

  The type of the value of a variable can be determined in two different ways

  - using gettype() function
    example:

  - using type testing functions.
    - is_int, is_integer, is_long : tests for integer type
    - is_double, is_float, is_real :test for double type
    - is_bool : test for boolean type
    - is_string :test for string type

    Examples:

OUTPUT

1. print function
   - Used to create simple unformatted output.
   - It can be called with or without parentheses around its parameter
   - can only have 1 string argument.

   Examples:

   ```
   print “Apples are red <br /> oranges aren’t <br />“;
   print “The result is: $result <br />”;
   print (47);
   ```

2. printf function
   - PHP borrows the printf function from C.
   - It is used when control over the format of displayed data is required.
   - printf requires parentheses around its parameters.
   - The general syntax:

     ```
     printf(literal_string, param1, param2, ...);
     ```
   - The literal string can include labeling information about the parameters whose values are to be displayed.
   - It also contains format codes for those values. The form of the format codes is a percent sign (%) followed by a field width and a type specifier.
   - The most common type specifiers are s for strings, d for integers, and f for floats and doubles.
   - The field width is either an integer literal (for integers) or two integer literals separated by a decimal point (for floats and doubles). The integer literal to the right of the decimal point specifies the number of digits to be displayed to the right of the decimal point.
   - The following examples illustrate how to specify formatting information:
     - %10s—a character string field of 10 characters
     - %6d—an integer field of six digits
     - %5.2f—a float or double field of eight spaces, with two digits to the right of the decimal point, the decimal point, and five digits to the left
   - The position of the format code in the first parameter of printf indicates the place in the output where the associated value should appear, as in the following code:

     ```
     $day = “Tuesday”;
     $high = 79;
     printf(“The high on %7s was %3d”, $day, $high);
     ```

3. echo
   - Is a native PHP construct
   - can have multiple string arguments
   - parentheses () are optional
   - only a single argument if using ()

   Examples:

   ```
   echo "<p>Cool</p>", "<h1>Hi</h1>";
   echo("PIC40A");
   echo "The sum is: $sum";
   ```

   ```
   $s = "hello";
   echo $s;
   echo "$s";
   echo ($s);
   echo ("$s");
   ```

Control statements

- Relational operators
  >,<,>=,<=,!= ,==,===,!==
- Boolean operator
  and, or, xor, !, && and | |
- Selection statements
  If-else

```
if ($day == "Saturday" || $day == "Sunday")
  $today = "weekend";
else {
  $today = "weekday";
  $work = true;
}
```

  switch

```
switch ($bordersize) {
  case "0": print "<table>";
            break;
  case "1": print "<table border = '1'>";
            break;
  case "4": print "<table border = '4'>";
            break;
  case "8": print "<table border = '8'>";
            break;
  default: print "Error-invalid value: $bordersize <br />";
}
```

- Loop statements
  while

  The following example computes the factorial of $n:

```
$fact = 1;
$count = 1;
while ($count < $n) {
  $count++;
  $fact *= $count;
}
```

  do-while

  This example computes the sum of the positive integers up to 100

```
$count = 1;
$sum = 0;
do {
  $sum += $count;
  $count++;
} while ($count <= 100);
```

  for

  The following example computes the factorial of $n:

```
for ($count = 1, $fact = 1; $count < $n;) {
  $count++;
  $fact *= $count;
}
```

Arrays

- Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable
- There are two ways to create an array in PHP.
  - Assigning a value to a subscripted variable that previously was not an array creates the array.

    Example:

    $fruits[0]=”apple”;
    $fruits[1]=”mango”;
    $fruits[2]=”orange”;
  - Array is created using an **array()** function.

    Example:

    $fruits=array(“apple”,”mango”,”orange”);

There are basically three types of arrays in PHP:

1. **Indexed or Numeric Arrays**: An array with a numeric index where values are stored linearly.
2. **Associative Arrays**: An array with a string index where instead of linear storage, each value can be assigned a specific key.
3. **Multidimensional Arrays**: An array which contains single or multiple array within it and can be accessed via multiple indices.

Examples:

Numeric array

```php
<?php

    //array creation
    $fruits=array("apple","mango","orange","grapes");

    //accessing array elements
    echo $fruits[0],"\n";
    echo $fruits[1],"\n";
    echo $fruits[2],"\n";
    echo $fruits[3],"\n";
    echo "<br/>";
    //sequential access using while-next loop
    print("<br/><br/>sequential access using while-next loop<br/>");
    $fruit=current($fruits);
    print("the first fruit is $fruit<br/>");

    while($fruit= next($fruits)){
        print("$fruit <br/>");
    }

    //sequential accesss using while-each loop
    print("<br/>sequential access using for loop<br/>");

    // count() or sizeof() function is used to count  the number of elements in an array
    $length=count($fruits);

    for($i=0;$i<$length;$i++){
        print("$fruits[$i]<br/>");
    }

    //sequential accesss using for-each loop
    print("<br/>sequential access using for-each loop<br/>");

    foreach ($fruits as $f) {

        print("$f<br/>");
    }
?>
```

output

```
apple mango orange grapes


sequential access using while-next loop
the first fruit is apple
mango
orange
grapes

sequential access using for loop
apple
mango
orange
grapes

sequential access using for-each loop
apple
mango
orange
grapes
```

Associative array

```php
<?php
    // associative array
    $weather=array("sun"=>24,"mon"=>27,"tue"=>30,"wed"=>20);

    //accessing array elements
    $m=$weather["mon"];
    print("Temp on maonday is $m<br>");

    //sequential access
    print("<br/>sequential access using for-each loop<br/>");

    foreach ($weather as $day => $temp) {
        print("the temperature on $day is $temp <br/>");
    }

    //sequential access using while-each
    print("<br/>sequential access using while-each loop<br/>");

    while ($w=each($weather)) {

        $day=$w["key"];
        $temp=$w["value"];
        print("the temperature on $day is $temp <br/>");
    }

    //use of array_keys and array_values
    $days=array_keys($weather);
    $temps=array_values($weather);

//printing arrays as index=>value using print_r() function
    print("<br><br>Using print_r() to print entire array<br>");
    print_r($days);
    print("<br/>");
    print_r($temps);
?>
```

output

Temp on maonday is 27

sequential access using for-each loop
the temperature on sun is 24
the temperature on mon is 27
the temperature on tue is 30
the temperature on wed is 20

sequential access using while-each loop
the temperature on sun is 24
the temperature on mon is 27
the temperature on tue is 30
the temperature on wed is 20

Using print_r() to print entire array
Array ([0] => sun [1] => mon [2] => tue [3] => wed)
Array ([0] => 24 [1] => 27 [2] => 30 [3] => 20)

Multidimensional array

```php
<?php
// Defining a multidimensional array
$favorites = array(
    "Dave Punk" => array(
        "mob" => "5689741523",
        "email" => "davepunk@gmail.com",
    ),
    "Monty Smith" => array(
        "mob" => "2584369721",
        "email" => "montysmith@gmail.com",
    ),
    "John Flinch" => array(
        "mob" => "9875147536",
        "email" => "johnflinch@gmail.com",
    )
);

//accessing elements
echo "Monty's mob number is ".$favorites["Monty Smith"]["mob"]."<br/>";

// Using for and foreach in nested form
$keys = array_keys($favorites);
for($i = 0; $i < count($favorites); $i++) {
    echo $keys[$i] . "\n";
    foreach($favorites[$keys[$i]] as $key => $value) {
        echo $key . " : " . $value . "\n";

    }
    print("<br>");
}

print_r($favorites);
?>
```

output

```
Monty's mob number is 2584369721
Dave Punk mob : 5689741523 email : davepunk@gmail.com
Monty Smith mob : 2584369721 email : montysmith@gmail.com
John Flinch mob : 9875147536 email : johnflinch@gmail.com
Array ( [Dave Punk] => Array ( [mob] => 5689741523 [email] => davepunk@gmail.com ) [Monty Smith] => Array ( [mob] => 2584369721
[email] => montysmith@gmail.com ) [John Flinch] => Array ( [mob] => 9875147536 [email] => johnflinch@gmail.com ) )
```

Implode() and explode() functions

- **The implode() function returns a string from elements of an array.** It takes an array of strings and joins them together into one string using a delimiter (string to be used between the pieces) of your choice.
- Syntax: implode (separator, array);
- The explode function in PHP breaks a string into smaller text with each break occurring at some symbol. This symbol is known as the delimiter.
- **Using the explode command we create an array from a string.**
- Syntax: explode(delimiter,string);

Example:

```php
<?php

//syntax
#implode(glue, pieces)
#explode(delimiter, string)

print("implode example<br>");
$arr=array("I","love","php");

$str=implode(" ", $arr);

echo "the array is \n";
print_r($arr);

echo "<br>the string is $str";
print("<br><br>Explode example");
$s="nepal college of information technology";
$a=explode(" ", $s);
echo "<br>the string is $s";
echo "<br>the array is \n";
print_r($a);



?>
```

output

```
implode example
the array is Array ( [0] => I [1] => love [2] => php )
the string is I love php

Explode example
the string is nepal college of information technology
the array is Array ( [0] => nepal [1] => college [2] => of [3] => information [4] => technology )
```

- ✓ The existence of an element of a specific key can be determined with the **array_key_exists** function, which returns a Boolean value.
- ✓ Example:

```
$highs = array("Mon" => 74, "Tue" => 70, "Wed" => 67,
               "Thu" => 62, "Fri" => 65);
if (array_key_exists("Tue", $highs)) {
  $tues_high = $highs["Tue"];
  print "The high on Tuesday was $tues_high <br />";
} else
  print
   "There is data for Tuesday in the \$highs array <br />";
```

- ✓ The **is_array** method takes a variable as its parameter and returns TRUE if the variable is an array, FALSE otherwise.
- ✓ The **in_array** function takes two parameters—an expression and an array—and returns TRUE if the value of the expression is a value in the array; otherwise, it returns FALSE.
- ✓ The number of elements in an array can be determined with the **sizeof** function.

Array sorting methods

1. sort() – sorts arrays in ascending order
2. rsort() – sorts arrays in descending order
3. asort() – sorts associative arrays in ascending order, according to the value
4. ksort() – sorts associative arrays in ascending order, according to the key
5. arsort() – sorts associative arrays in descending order, according to the value
6. krsort() – sorts associative arrays in descending order, according to the key

Examples:

```php
<?php

    $num=array(5,8,3,2,7);
    $original=$num;

    print("original array <br>");
    print_r($num);
    //sorting in ascending order
    sort($num);
    print("<br>after sorting<br>");
    print_r($num);
    //sorting in descending order
    rsort($num);
    print("<br>after sorting<br>");
    print_r($num);

    $country=array("name"=>"nepal","capital"=>"ktm","population"=>5674839,"area"=>147181);
    $c=$country;

    sort($c);
    print("<br>original array<br>");
    print_r($country);
    print("<br>after sorting using sort()<br>");
    print_r($c);

    $c=$country;
    asort($c);
    print("<br>after sorting using asort()<br>");
    print_r($c);

    $c=$country;
    ksort($c);
    print("<br>after sorting using ksort()<br>");
    print_r($c);


?>
```

Output:

original array
Array ([0] => 5 [1] => 8 [2] => 3 [3] => 2 [4] => 7)
after sorting
Array ([0] => 2 [1] => 3 [2] => 5 [3] => 7 [4] => 8)
after sorting
Array ([0] => 8 [1] => 7 [2] => 5 [3] => 3 [4] => 2)
original array
Array ([name] => nepal [capital] => ktm [population] => 5674839 [area] => 147181)
after sorting using sort()
Array ([0] => ktm [1] => nepal [2] => 147181 [3] => 5674839)
after sorting using asort()
Array ([capital] => ktm [name] => nepal [area] => 147181 [population] => 5674839)
after sorting using ksort()
Array ([area] => 147181 [capital] => ktm [name] => nepal [population] => 5674839)

PHP PATTERN MATCHING

- 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, like those of JavaScript.
- Regular expression used here is similar to those in JavaScript.

commonly used regular expression functions in PHP

- **preg_match** - this function is used to perform a pattern match on a string. It returns true if a match is found and false if a match is not found.
- **preg_split** - this function is used to perform a pattern match on a string and then split the results into a numeric array.
- **preg_replace** - this function is used to perform a pattern match on a string and then replace the match with the specified text.

Syntax: <?php function_name(" / pattern/ ", subject) ; ?>

Examples:

```php
$str="Kathmandu is the capital of Nepal";
echo "Given String: $str</br>";

echo "</br>preg-match!</br>";
if(preg_match("/Nepal/",$str)){
    echo "the string $str contains the word Nepal";
}
else
    echo "word Nepal doesn't found!";


echo "</br></br>preg-split</br>";
$array=preg_split("/\s/",$str);
print_r($array);

echo "</br></br>preg-replace</br>";
$new=preg_replace("/Nepal/","<span style='color:blue;font-size:20px'>the Republic of
    Nepal</span>",$str);
echo "after replacing Nepal with the Republic of Nepal the string is : $new";
```

Output:

Given String: Kathmandu is the capital of Nepal

preg-match!
the string Kathmandu is the capital of Nepal contains the word Nepal

preg-split
Array ([0] => Kathmandu [1] => is [2] => the [3] => capital [4] => of [5] => Nepal)

preg-replace
after replacing Nepal with the Republic of Nepal the string is : Kathmandu is the capital of the Republic of Nepal

Text-book examples:

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

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

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

Example: counting no.of words in a given string

```
<?php

function splitter($str){

    //creating empty array for storing word frequencies
    $freq=array();
    //splitting given string into words
    $words=preg_split("/[ \.,;:!\?]\s*/", $str);
    //loop to count the words

    foreach ($words as $word) {
        $keys=array_keys($freq);
        if(in_array($word, $keys))
            $freq[$word]++;
        else
            $freq[$word]=1;
    }
    return $freq;
}
$str="apples are good for you, or don't you like apples?
 or may be you like oranges better than apples";
//calling splitter function
$tbl=splitter($str);
//displaying words and their frequencies
$sorted_keys=array_keys($tbl);
sort($sorted_keys);
foreach ($sorted_keys as $word) {
    print "$word $tbl[$word] <br/>";
}
?>
```

output

apples 3
are 1
be 1
better 1
don't 1
for 1
good 1
like 2
may 1
or 2
oranges 1
than 1
you 3

PHP FORM HANDLING

Commonly used methods that are related to form handling in PHP.

- ✓ $_GET & $_POST
  - $_POST is an associative array of variables passed to the current script via the HTTP POST method
  - It is used when you are sending large data to server or if you have sensitive information like passwords, credit card details etc
  - $_GET is an associative array of variables passed to the current script via the URL parameters.
  - Difference between GET and POST methods in php

GET vs POST Method in PHP	
GET is a method that sends information by appending them to the page request.	POST is a method that transfers information via HTTP header.
URL	
The form information is visible in the URL	The form information is not visible in the URL
Information Amount	
Limited amount of information is sent. It is less than 1500 characters.	Unlimited amount of information is sent.
Usage	
Helps to send non-sensitive data	Helps to send sensitive data (passwords), binary data (word documents, images)and uploading files
Security	
Not very secure.	More secure.
Bookmarking the Page	
Possible to bookmark the page	Not possible to bookmark the page

GET requests can be cached
GET requests remain in the browser history

POST requests are never cached
POST requests do not remain in the browser history

- htmlspecialchars() : used here to convert specific HTML characters to their HTML entity names,
  e.g. & (ampersand) becomes &
  " (double quote) becomes " ' (single quote) becomes '
  < (less than) becomes < > (greater than) becomes >

- $_SERVER["PHP_SELF"] : it is used as a value for action attribute of html form & thus sending the submitted form data to the page itself, instead of jumping to a different page.
- $_SERVER["REQUEST_METHOD"] : used to check whether the form is submitted or not.
  If the REQUEST_METHOD is POST or GET, then you know the script has been submitted.
  (or you can check form submission using **isset()** method; e.g: isset($_GET["submit"]))
- empty() : the empty() function is used to check whether a variable is empty or not.
- Filter_var() : used for email validation . e.g: filter_var($email, FILTER_VALIDATE_EMAIL)
- trim() : The trim() function removes whitespace and other predefined characters from both sides of a string.
- stripslashes : it removes backslashes.

COOKIES AND SESSIONS

- A cookie is a small object of information that includes a name and a textual value.
- A cookie is created by some software system on the server and stored at client side.
- cookie is information that is exchanged exclusively between one specific browser and one specific server.
- Cookies can store only "string" datatype.
- Cookie is non-secure since stored in text-format at client side.
- Cookies can store only small(limited) amount of information(maximum size of cookie is 4KB)

Why Cookies?

- Http is a stateless protocol; cookies allow us to track the state of the application using small files stored on the user's computer
- Personalizing the user experience – this is achieved by allowing users to select their preferences.
- Tracking the pages visited by a user

Creating cookies

- "setcookie" is the PHP function used to create the cookie.
  Basic syntax: **setcookie(cookie_name, cookie_value, [expiry_time]);**
- "cookie_name" is the name of the cookie that the server will use when retrieving its value from the $_COOKIE array variable. It's mandatory.
- "cookie_value" is the value of the cookie and its mandatory
- "[expiry_time]" is optional; it can be used to set the expiry time for the cookie such as 1 hour. The time is set using the PHP time() functions plus or minus a number of seconds greater than 0 i.e. time() + 3600 for 1 hour.

Accessing Cookie Values

- For accessing a cookie value, the PHP $_COOKIE super global variable is used.
- It is an associative array that contains a record of all the cookies values sent by the browser in the current request.
  Basic syntax: $_COOKIE["cookie_name"];

Deleting Cookies

- Cookies are automatically deleted after the expiry_time.
- Manually they can be deleted by setcookie() function but the expiration time is required to be set in the past.

Example:

```
<?php

setcookie("user_name", "xyz", time()+3600);  //setting cookie for 1 hour

echo "The user name is" . $_COOKIE["user_name"];  //accessing cookie value

setcookie("user_name", "xyz", time()-3600);  //deleting cookie

?>
```

- A session is the time span during which a browser interacts with a particular server.
- A session is a global variable stored on the server.
- Each session is assigned a unique id which is used to retrieve stored values.
- Session values are stored at server side.
- Session can store any type of data. Sessions have the capacity to store relatively large data compared to cookies.
- Sessions are secured because it is stored in binary format/encrypted form and gets decrypted at server.
- The session values are automatically deleted when the browser is closed
- session variables are stored in the $_SESSION array variable.

Why sessions?

- to make data accessible across the various pages of an entire website.
- to store global variables in an efficient and more secure way compared to passing them in the URL
- You are developing an application such as a shopping cart that has to temporary store information with a capacity larger than 4KB
- to store important information such as the user id more securely on the server where malicious users cannot tamper with them.
- (Each user's shopping cart is identified by a session identifier, which could be implemented as a cookie. So, cookies can be used to identify each of the customers visiting the site at a given time.)

Creating, Accessing and deleting sessions

- To create a session, you must first call the PHP **session_start()** function and then store the values in the $_SESSION array variable.
- Syntax for creation session variable: $_SESSION["variable_name"]="variable_value";
- The **session_destroy()** function is used to destroy the whole Php session variables.
- to destroy only a single session item, use the **unset()** function.
- Syntax:

```
unset($_SESSION['variable_name']);
```

Example: a program to count no.of times a page loaded

```
<?php
session_start(); //start the PHP_session function
if(isset($_SESSION['page_count']))  // checking session variable is set or not
{
$_SESSION['page_count'] += 1;
}
else {
 $_SESSION['page_count'] = 1;       // creating a session variable with value 1
}
echo 'You are visitor number ' . $_SESSION['page_count'];
?>
```

Destroying and unsetting session

```
session_destroy(); //destroy entire session

unset($_SESSION['page_count']); //unsetting/deleting page_count session item
```

PHP FILE HANDLING

- Because PHP is a server-side technology, it is possible to create, read, and write files on the server system using it.
- PHP has many functions to work with normal files. Some of the basic functions are:

1. fopen()
    - PHP fopen() function is used **to open a file**.
    - It takes two parameters: the **file name**, including the path to it if it is in a different directory, and a **use indicator**, which specifies the operation or operations that will be performed on the file.
    - The fopen() function returns the reference to the file for the file variable.
    - Every open file has an **internal pointer(file pointer)** that is used to indicate where the next operation should take place within the file.
    - Syntax

```
<?php
        $file = fopen("file_name",'mode');
?>
```

    - Example:

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

    - Example:

```
$file_var = fopen("testdata.dat", "r") or
            die ("Error – testdata.dat cannot be opened");
```

[The fopen function returns FALSE if it fails. The die( ) function is used to display a message and exit the script.]

Table 9.4 File use indicators

Use Indicator	Description
"r"	Read only. The file pointer is initialized to the beginning of the file.
"r+"	Read from and write to an existing file. The file pointer is initialized to the beginning of the file; if a read operation precedes a write operation, the new data is written just after the location at which the read operation left the file pointer.
"w"	Write only. Initializes the file pointer to the beginning of the file; creates the file if it does not exist.
"w+"	Write and read. Initializes the file pointer to the beginning of the file; creates the file if it does not exist. Always initializes the file pointer to the beginning of the file before the first write, destroying any existing data.
"a"	Write only. If the file exists, initializes the file pointer to the end of the file; if the file does not exist, creates it and initializes the file pointer to its beginning.
"a+"	Read and write, creating the file if necessary; new data is written to the end of the existing data.

2. fread()
   - The fread function reads part or all of a file and returns a string of what was read.
   - This function takes two parameters: a **file variable** and the number of bytes to be read. The reading operation stops when either the end-of-file marker is read or the specified number of bytes has been read.
   - Example:

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

OR

```
$file_var = fopen("testdata.dat", "r") or
            die ("Error – testdata.dat cannot be opened");

$file_string = fread($file_var,
                     filesize("testdata.dat"));
```

   - One alternative to fread is **file**
   - It which takes a **file name** as its parameter and **returns an array of all of the lines** of the file.
   - One advantage of file is that the file open and close operations are not necessary.
   - For example, the following statement places the lines of testdata.dat into an array named file_lines:

```
$file_lines = file("testdata.dat");
```

   - The function **file_get_contents** takes the file's name as its parameter.
   - This function reads the entire contents of the file, as exemplified in the following call:

```
$file_string = file_get_contents("testdata.dat");
```

   - A single line of a file can be read with **fgets,** which takes two parameters:
   - the **file variable** and a **limit on the length of the line to be read**. $line =

```
fgets($file_var, 100);
```

     It reads characters from the file whose file variable is $file_var until it finds a newline character, encounters the end-of-file marker, or has read 99 characters.

   - A single character can be read from a file with **fgetc**

3. fwrite()
    - The fwrite() function takes two parameters: a **file variable** and the **string to be written** to the file. The fwrite function returns the number of bytes written.
    - Example

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

    - The **file_put_contents**() function writes the value of its second parameter, a string, to the file specified in its first parameter.

```
file_put_contents("savedata.dat", $str);
```

4. fclose()
    - file is closed using fclose() function. Its argument is file which needs to be closed
    - Example:

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

5. unlink()
    - The unlink function is used to delete the file
    - Example:

```
unlink("demo.txt");
```

[file_exists(): used to determine whether a file exists or not.

Example:

```
<?php
        file_exists($filename);
?>
```

"file_exists()" is the PHP function that returns true if the file exists and false if it does not exist.]