Unit 2

STRING, ARRAY, FUNCTIONS, SESSION COOKIE

Strings in PHP

Strings are sequences of characters that can be treated as a unit — assigned to variables, given as input to functions, returned from functions, or sent as output to appear on your user's web page.

The simplest way to specify a string in PHP code is to enclose it in quotation marks, whether single quotation marks (') or double quotation marks ("), like this:

- $my_string = 'a literal string';
- $another_string = "another string";

If you enclose a string in single quotation marks, almost no interpolation will be performed; if you enclose it in double quotation marks, PHP will splice in the values of any variables you include, as well as make substitutions for certain special character sequences that begin with the backslash (\) character.

```
$statement = ‘everything I say’;
$question_1 = “Do you have to take $statement so literally?\n<BR>”;
$question_2 = ‘Do you have to take $statement so literally?\n<BR>’;
echo $question_1;
echo $question_2;
```

Interpolation with curly braces

- $plan1 = “I will play $sport1ball in the summertime”; //wrong
- $plan1 = “I will play {$sport1}ball in the summertime”; //right

Characters and string indexes

You can retrieve the individual characters of a string by including the number of the character, starting at 0, enclosed in curly braces immediately following a string variable.

These characters will actually be one-character strings.

```
$my_string = “Doubled”;
for ($index = 0; $index < 7; $index++) {
   $string_to_print = $my_string{$index};
   print(“$string_to_print$string_to_print”);
}
```

String operators

PHP offers two string operators: the dot (.) or concatenation operator and the .= concatenating assignment operator.

The concatenation operator, when placed between two string arguments, produces a new string that is the result of putting the two strings together in sequence. For example:

- $my_two_cents = “I want to give you a piece of my mind “;
- $third_cent = “ and another thing”;
- print($my_two_cents . “...” . $third_cent);
- How many string are there?

Concatenation and assignment

PHP has a shorthand operator (.=) that combines concatenation with assignment. The following statement:

- $my_string_var .= $new_addition;

is exactly equivalent to:

- $my_string_var = $my_string_var . $new_addition;

The heredoc syntax

PHP offers another way to specify a string, called the *heredoc syntax.*

This syntax turns out to be extremely useful for specifying large chunks of variable-interpolated text, because it spares you from the need to escape internal quotation marks.

It is especially useful in creating pages that contain HTML forms.

The operator in the heredoc syntax is <<<. What is expected immediately after this is a label (unquoted) that indicates the beginning of a multiline string.

PHP will continue including subsequent lines in this string until it sees the same label again, beginning a line.

The ending label may optionally be followed by a semicolon but by nothing else.

```
$my_string_var = <<<EOT
Everything in this rather unnecessarily wordy
ramble of prose will be incorporated into the
string that we are building up inevitably, inexorably,
character by character, line by line, until we reach that
blessed final line which is this one.
EOT;
```

Interpolation of variables happens exactly the same way as with double-quoted strings. The nice thing about heredoc, though, is that quote signs can be included without any escaping and without prematurely terminating the string. Here's another example:

```
echo <<<ENDOFFORM
<FORM METHOD=POST aCTION="{$_ENV['PHP_SELF']}">
<INPUT TYPE=TEXT NaME=FIRSTNaME VaLUE=$firstname>
<INPUT TYPE=SUBMIT NaME=SUBMIT VaLUE=SUBMIT>
</FORM>
ENDOFFORM;
```

String Functions

Inspecting strings:

strlen(): Using the strlen() function (the name is short for *string length).*

```
$short_string = “This string has 29 characters”;
print(“It does have “ . strlen($short_string) . “ characters”);
for ($index = 0; $index < strlen($short_string); $index++)
print($short_string{$index});
```

Finding characters and substrings

strpos(): the strpos() function finds the numerical position of a particular character in a string, if it exists.

- $twister = “Peter Piper picked a peck of pickled peppers”;
- print(“Location of ‘p’ is “ . strpos($twister, ‘p’) .’
’);
- print(“Location of ‘q’ is “ . strpos($twister, ‘q’) .’
’);

the strpos() function is case sensitive.

The strpos() function can also be used to search for a substring rather than a single character, simply by giving it a multicharacter string rather than a single-character string.

You can also supply an extra integer argument specifying the position to begin searching forward from.

Searching in reverse is also possible, using the strrpos() function. Unlike with strpos(), the string searched for must have only one character.

- $twister = “Peter Piper picked a peck of pickled peppers”;
- printf(“Location of ‘p’ is “ . strrpos($twister, ‘p’) .’
’);
- Location of ‘p’ is 40

Comparison and searching

The simplest method to find an answer of comparision is to use the basic comparison operator (==), which does equality testing on strings as well as numbers.

The most basic workhorse string-comparison function is strcmp(). It takes two strings as arguments and compares them byte by byte until it finds a difference.

It returns a negative number if the first string is less than the second and a positive number if the second string is less. It returns 0 if they are identical.

The strcasecmp() function works the same way, except that the equality comparison is case insensitive. The function call strcasecmp(“hey!”, “HEY!”) should return 0.

Searching

The strstr() function takes a string to search in and a string to look for (in that order).

If it succeeds, it returns the portion of the string that starts with (and includes) the first instance of the string it is looking for.

If the string is not found, a false value is returned

- $string_to_search = “showsuponceshowsuptwice”;
- $string_to_find = “up”;
- print(“Result of looking for $string_to_find” .
- strstr($string_to_search, $string_to_find) . “
”);

The strstr() function also has an alias by the name of strchr().

Other than the name, the two functions are identical. Just as with strcmp(), strstr() has a case insensitive version, by the name of stristr().

Function	Behavior
`strlen()`	Takes a single string argument and returns its length as an integer.
`strpos()`	Takes two string arguments: a string to search, and the string being searched for. Returns the (0-based) position of the beginning of the first instance of the string if found and a false value otherwise. It also takes a third optional integer argument, specifying the position at which the search should begin.
`strrpos()`	Like `strpos()`, except that it searches backward from the end of the string, rather than forward from the beginning. The search string must only be one character long, and there is no optional position argument.
`strcmp()`	Takes two strings as arguments and returns 0 if the strings are exactly equivalent. If `strcmp()` encounters a difference, it returns a negative number if the first different byte is a smaller ASCII value in the first string, and a positive number if the smaller byte is found in the second string.
`strcasecmp()`	Identical to `strcmp()`, except that lowercase and uppercase versions of the same letter compare as equal.
`strstr()`	Searches its first string argument to see if its second string argument is contained in it. Returns the substring of the first string that starts with the first instance of the second argument, if any is found — otherwise, it returns false.
`strchr()`	Identical to `strstr()`.
`stristr()`	Identical to `strstr()` except that the comparison is case independent.

Substring selection

Many of PHP's string functions have to do with slicing and dicing your strings.

By *slicing, we mean* choosing a portion of a string;

By *dicing, we mean selectively modifying a string.*

Keep in mind that (most of the time) even dicing functions do not change the string you started out with. Usually, such functions return a modified copy, leaving the original argument intact.

The most basic way to choose a portion of a string is the substr() function, which returns a new string that is a subsequence of the old one.

as arguments, it takes a string (that the substring will be selected from), an integer (the position at which the desired substring starts), and an optional third integer argument that is the length of the desired substring.

If no third argument is given, the substring is assumed to continue until the end.

For example, the statement:

echo(substr(“Take what you need, and leave the rest behind”, 23));

- prints the string leave the rest behind, whereas the statement:

echo(substr(“Take what you need, and leave the rest behind”, 5, 13));

- prints what you need — a 13-character string starting at (0-based) position 5.

Both the start-position argument and the length argument can be negative, and in each case the negativity has a different meaning.

If the start position is negative, it means that the starting character is determined by counting backward from the end of the string, rather than forward from the beginning.

a negative-length argument means that the final character is determined by counting backward from the end rather than forward from the start position.

Here are some examples, with positive and negative arguments:

$alphabet_test = “abcdefghijklmnop”;

- print(“3: “ . substr($alphabet_test, 3) . “
”);
- print(“-3: “ . substr($alphabet_test, -3) . “
”);
- print(“3, 5: “ . substr($alphabet_test, 3, 5) . “
”);
- print(“3, -5: “ . substr($alphabet_test, 3, -5) . “
”);
- print(“-3, -5: “ . substr($alphabet_test, -3, -5) . “
”);
- print(“-3, 5: “ . substr($alphabet_test, -3, 5) . “
”);

This gives us the output:

- 3: defghijklmnop
- -3: nop
- 3, 5: defgh
- 3, -5: defghijk
- -3, -5:
- -3, 5: nop

String cleanup functions

The functions chop(), ltrim(), and trim() are really used for cleaning up untidy strings. They trim whitespace off the end, the beginning, and the beginning and end, respectively, of their single string argument.

- $original = “ More than meets the eye “;
- $chopped = chop($original);
- $ltrimmed = ltrim($original);
- $trimmed = trim($original);
- print(“The original is ‘$original’
”);
- print(“Its length is “ . strlen($original) . “
”);
- print(“The chopped version is ‘$chopped’
”);
- print(“Its length is “ . strlen($chopped) . “
”);
- print(“The ltrimmed version is ‘$ltrimmed’
”);
- print(“Its length is “ . strlen($ltrimmed) . “
”);
- print(“The trimmed version is ‘$ltrimmed’
”);
- print(“Its length is “ . strlen($trimmed) . “
”);

In addition to spaces, these functions remove whitespace like that denoted by the escape sequences \n, \r, \t, and \0

String replacement

The str_replace() function enables you to replace all instances of a particular substring with an alternate string.

It takes three arguments: the string to be searched for, the string to replace it with when it is found, and the string to perform the replacement on.

```
$first_edition = “Burma is similar to Rhodesia in at least one way.”;
$second_edition = str_replace(“Rhodesia”, “Zimbabwe”, $first_edition);
$third_edition = str_replace(“Burma”, “Myanmar”, $second_edition);
print($third_edition);
```

This replacement will happen for all instances found of the search string.

str_replace() picks out portions to replace by matching to a target string; by contrast, substr_replace() chooses a portion to replace by its absolute position.

The function takes up to four arguments: the string to perform the replacement on, the string to replace it with, the starting position for the replacement, and (optionally) the length of the section to be replaced

print(substr_replace(“aBCDEFG”, “-“, 2, 3));

you are allowed to replace a substring with a string of a different length.

If the length argument is omitted, it is assumed that you want to replace the entire portion of the string after the start position.

The substr_replace() function also takes negative arguments for starting position and length, which are treated exactly the same way as in the substr() function.

It is important to remember with both str_replace and substr_replace that the original string remains unchanged by these operations

The strrev() function simply returns a new string with the characters of its input in reverse order.

The str_repeat() function takes a string argument and an integer argument and returns a string that is the appropriate number of copies of the string argument tacked together.

Function	Behavior
`substr()`	Returns a subsequence of its initial string argument, as specified by the second (position) argument and optional third (length) argument. The substring starts at the indicated position and continues for as many characters as specified by the length argument or until the end of the string, if there is no length argument. A negative position argument means that the start character is located by counting backward from the end, whereas a negative length argument means that the end of the substring is found by counting back from the end, rather than forward from the start position.
chop(), or rtrim()	Returns its string argument with trailing (right-hand side) whitespace removed. Whitespace is a blank space, `\n`, `\r`, `\t`, and `\0`.
`ltrim()`	Returns its string argument with leading (left-hand side) whitespace removed.
`Trim()`	Returns its string argument with both leading and trailing whitespace removed.
`Str_ replace()`	Used to replace target substrings with another string. Takes three string arguments: a substring to search for, a string to replace it with, and the containing string. Returns a copy of the containing string with *all* instances of the first argument replaced by the second argument.
`Substr_ replace()`	Puts a string argument in place of a position-specified substring. Takes up to four arguments: the string to operate on, the string to replace with, the start position of the substring to replace, and the length of the string segment to be replaced. Returns a copy of the first argument with the replacement string put in place of the specified substring. If the length argument is omitted, the entire tail of the first string argument is replaced. Negative position and length arguments are treated as in `substr()`.

Case functions

strtolower() The strtolower() function returns an all-lowercase string. It doesn't matter if the original is all uppercase or mixed.

```
<?php
$original = "They DON'T KnoW they're  SHOUTING";
$lower = strtolower($original);
echo $lower;
?>
```

strtoupper() The strtoupper() function returns an all-uppercase string, regardless of whether the original was all lowercase or mixed:

```
<?php
$original = "make this link stand out";
echo("<B>strtoupper($original)</B>");
?>
```

ucfirst() The ucfirst() function capitalizes only the first letter of a string:

```
<?php
$original = “polish is a word for which pronunciation depends on
capitalization”;
echo(ucfirst($original));
?>
```

ucwords() The ucwords() function capitalizes the first letter of each word in a string:

```
<?php
$original = “truth or consequences”;
$capitalized = ucwords($original);
echo “While $original is a parlor game, $capitalized is a town in New
Mexico.”;
?>
```

Printing and output

PHP also offers printf() and sprintf(), which are modeled on C functions of the same name.

```
<pre>
<?php
    $value = 3.14159;
    printf("%f,%10f,%-010f,%2.2f\n",
    $value, $value, $value, $value);
?>
</pre>
```

gives us:

3.141590, 3.141590,3.141590000000000, 3.14

Learning arrays

PHP arrays can store data of varied types and automatically organize it for you in a large variety of ways.

an array is a collection of variables indexed and bundled into a single, easily referenced super variable that offers an easy way to pass multiple values between lines of code, functions, and even pages.

What are PHP arrays?

PHP arrays are *associative arrays.* The *associative part means* that arrays store element values in association with key values rather than in a strict linear index order.

For example, storage is as simple as this:

- $state_location[‘San Mateo’] = ‘California’;

which stores the element ‘California’ in the array variable $state_location, in association with the lookup key ‘San Mateo’. after this has been stored, you can look up the stored value by using the key, like so:

- $state = $state_location[‘San Mateo’];

Similarly, if you want to associate a numerical ordering with a bunch of values, all you have to do is use integers as your key values, as in:

- $my_array[1] = “The first thing”;
- $my_array[2] = “The second thing”; // and so on ...

Creating arrays

There are three main ways to create an array in a PHP script:

- by assigning a value into one (and thereby implicitly creating it),
- by using the array() construct, and
- by calling a function that happens to return an array as its value.

Direct assignment: The simplest way to create an array is to act as though a variable is already an array and assign a value into it, like this:

```
$my_array[1] = “The first thing in my array that I just made”;
```

The array() construct: It creates a new array from the specification of its elements and associated keys.

In its simplest version, array() is called with no arguments, which creates a new empty array.

In its next simplest version, array() takes a comma separated list of elements to be stored, without any specification of keys.

The result is that the elements are stored in the array in the order specified and are assigned integer keys beginning with zero.

- $fruit_basket = array(‘apple’, ‘orange’, ‘banana’, ‘pear’);

causes the variable $fruit_basket to be assigned to an array with four string elements, with the indices 0, 1, 2, and 3, respectively.

The same effect could also have been accomplished by omitting the indices in the assignment, like so:

- $fruit_basket[] = ‘apple’;
- $fruit_basket[] = ‘orange’;

Specifying indices using array(): array() offers us a special syntax for specifying what the indices should be. Instead of element values separated by commas, you supply key/value pairs separated by commas, where the key and value are separated by the special symbol =>.

Consider the following statement:

- $fruit_basket = array(0 => ‘apple’, 1 => ‘orange’, 2 => ‘banana’, 3 => ‘pear’);

Or it can be

- $fruit_basket = array(‘red’ => ‘apple’, ‘orange’ => ‘orange’, ‘yellow’ => ‘banana’, ‘green’ => ‘pear’);

To recover the name of the yellow fruit, for example, we just evaluate the expression:

- $fruit_basket[‘yellow’] // will be equal to ‘banana’

you can create an empty array by calling the array function with no arguments. For example:

- $my_empty_array = array();

creates an array with no elements.

Functions returning arrays: This may be a user defined function, or it may be a built-in function that makes an array via methods internal to PHP.

Many database-interaction functions, for example, return their results in arrays that the functions create on the fly.

Other functions exist simply to create arrays that are handy to have as grist for later array-manipulating functions.

One such is range(), which takes two integers as arguments and returns an array filled with all the integers (inclusive) between the arguments. In other words:

- $my_array = range(1,5);
- is equivalent to:
- $my_array = array(1, 2, 3, 4, 5);

Retrieving Values

Retrieving by index: The most direct way to retrieve a value is to use its index. If we have stored a value in $my_array at index 5, $my_array[5] should evaluate to the stored value.

The list() construct: It used to assign several array elements to variables in succession. Suppose that the following two statements are executed:

- $fruit_basket = array('apple', 'orange', 'banana');
- list($red_fruit, $orange_fruit) = $fruit_basket;

This will assign the string 'apple' to the variable $red_fruit and the string 'orange' to the variable $orange_fruit.

The variables in list() will be assigned to elements of the array in the order they were originally stored in the array.

Multidimensional arrays

Facilitates storing arrays of arrays!

Extends beyond the associative array to store multiple sets of associative arrays

```
<?php
  $person = array(
        array(“name” => “John”, “age” => 19),
        array(“name” => “Mary”, “age” => 30),
        array(“name” => “aine”, “age” => 23)
        );
?>
```

each of the array elements store starting at an index value of 0

for instance:

- echo $person[0];
- echo $person[1];
- and so on and on and on!

- foreach ($person as $p) {

```
while (list($n, $a) = each ($p)) {
                    echo “Name: “.$n.”\n age: ”.$a.”<br/>”;
                    }
}
```

- Output:

  Name: John

  age: 19

  Name: Mary

  age: 30

  Name: aine

  age: 23

Inspecting arrays

Function	Behavior
`is_array()`	Takes a single argument of any type and returns a true value if the argument is an array, and false otherwise.
`count()`	Takes an array as argument and returns the number of nonempty elements in the array. (This will be 1 for strings and numbers.)
`sizeof()`	Identical to `count()`.
`in_array()`	Takes two arguments: an element (that might be a value in an array), and an array (that might contain the element). Returns `true` if the element is contained as a value in the array, `false` otherwise. (Note that this does not test for the presence of keys in the array.)
`isset($array[$key])`	Takes an *`array[key]`* form and returns `true` if the key portion is a valid key for the array. (This is a specific use of the more general function `isset()`, which tests whether a variable is bound.)

Deleting from arrays

Deleting an element from an array is simple, exactly analogous to getting rid of an assigned variable.

Just call unset(), as in the following:

- $my_array[0] = ‘wanted’;
- $my_array[1] = ‘unwanted’;
- $my_array[2] = ‘wanted again’;
- unset($my_array[1]);

Note that this is *not the same as setting the contents to an empty value. If, instead of calling* unset(), we had the following statement:

- $my_array[1] = ‘ ’;

at the end we would have three stored values (‘wanted’, ‘’, ‘wanted again’) in association with three keys (0, 1, and 2, respectively).

Iteration

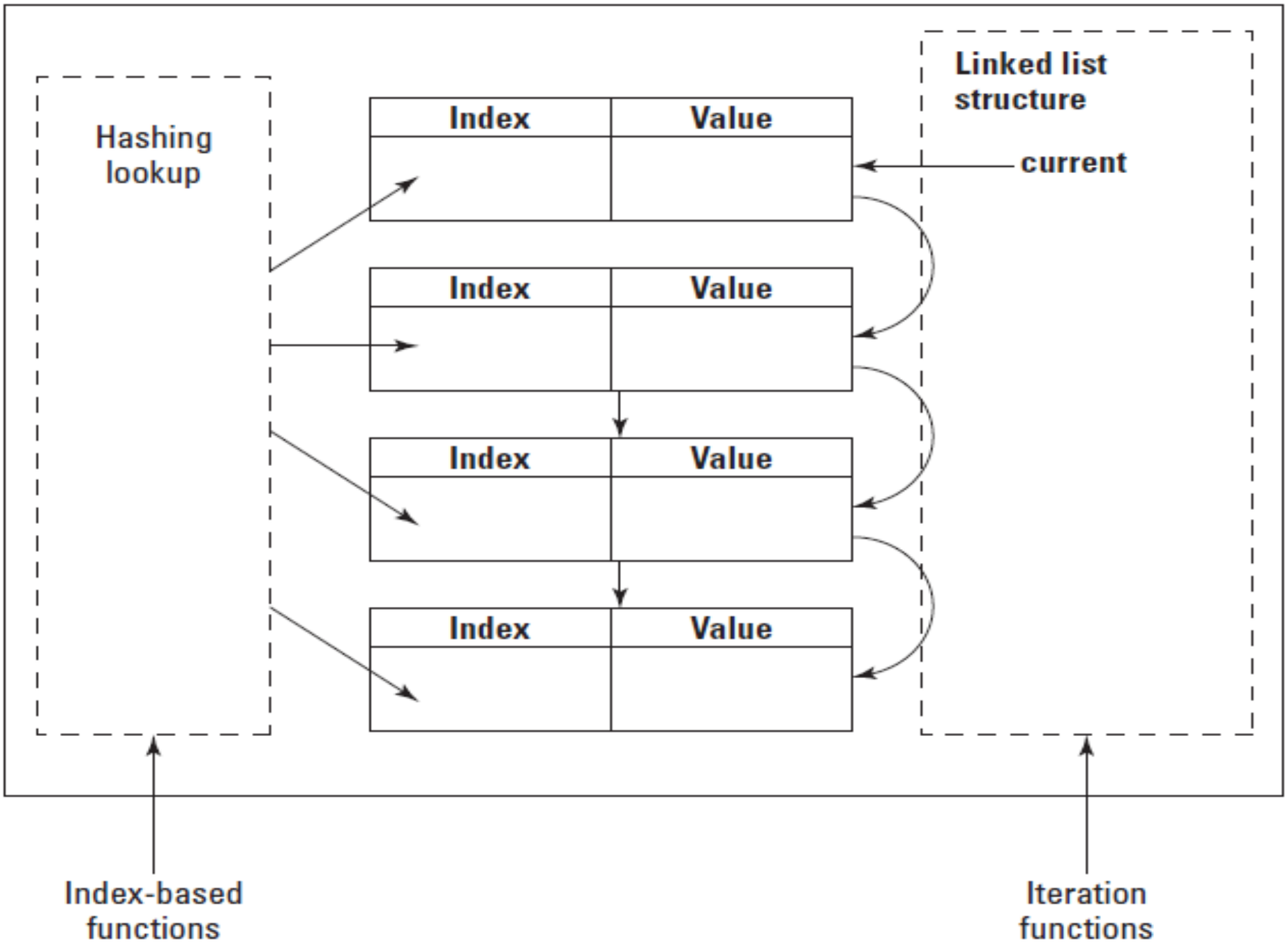
Iteration constructs help us do this by letting us step or loop through arrays, element by element or key by key.

In addition to storing values in association with their keys, PHP arrays silently build an ordered list of the key/value pairs that are stored, in the order that they are stored.

The reason for this is to support operations that iterate over the entire contents of an array.

Each stored key/value pair points to the next one, and one side effect of adding the first element to an array is that a current pointer points to the very first element, where it will stay unless disturbed by one of the iteration functions

Internal structure of an array



foreach method

```
foreach ($array_variable as $value_variable) {
   // .. do something with the value in $value_variable

}


foreach ($array_variable as $key_var => $value_var) {
   // .. do something with $key_var and/or $value_var

}
```

Iterating with current() and next()

foreach, is really only good for situations where you want to simply loop through an array's values.

For more control, let's look at current() and next().

The current() function returns the stored value that the current pointer points to.

When an array is newly created with elements, the element pointed to will always be the first element.

The next() function first advances that pointer and then returns the current value pointed to.

If the next() function is called when the current pointer is already pointing to the last stored value and, therefore, runs off the end of the array, the function returns a false value.

Starting over with reset()

- The reset() function gives us a way to “rewind” that pointer to the beginning — it sets the pointer to the first key/value pair and then returns the stored value.
- We can use it to make our printing function more robust by replacing the call to current() with a call to reset().

Reverse order with end() and prev():

- There are also the functions prev(), which moves the pointer back by one, and end(), which jumps the pointer to the last entry in the list.

Extracting keys with key():

- The keys are also retrievable from the internal linked list of an array by using the key() function.
- $current_key = key($city_array);

Walking with array_walk():

- It lets you pass an arbitrary function of your own design over an array, doing whatever your function pleases with each key/value pair.
- The array_walk() function takes two arguments: an array to be traversed and the name of a function to apply to each key/value pair.

```
function print_value_length($array_value,  array_key_ignored)
{
   $the_length = strlen($array_value);
   print("The length of $array_value is $the_length<BR>");
}
array_walk($major_city_info, 'print_value_length');
```

Some important array functions

array_diff(): Compares array values, and returns the differences

array_fill(): Fills an array with values

array_flip(): Exchanges all keys with their associated values in an array

array_intersect():Compares array values, and returns the matches

arry_merge(): Merges one or more arrays into one array

array_pop():Deletes the last element of an array

array_push(): Inserts one or more elements to the end of an array

array_shift(): Removes the first element from an array, and returns the value of the removed element

array_unshift(): Adds one or more elements to the beginning of an array

array_reverse(): Returns an array in the reverse order

array_search(): Searches an array for a given value and returns the key

array_sum(): Returns the sum of the values in an array

arry_unique(): Removes duplicate values from an array

asort(), arsort(), ksort, krsort(),sort(),rsort(),

Each(): Returns the current key and value pair from an array

Learning PHP Number Handling

Numerical Types: PHP has only two numerical types: *integer (also known as long), and double* (aka *float).*

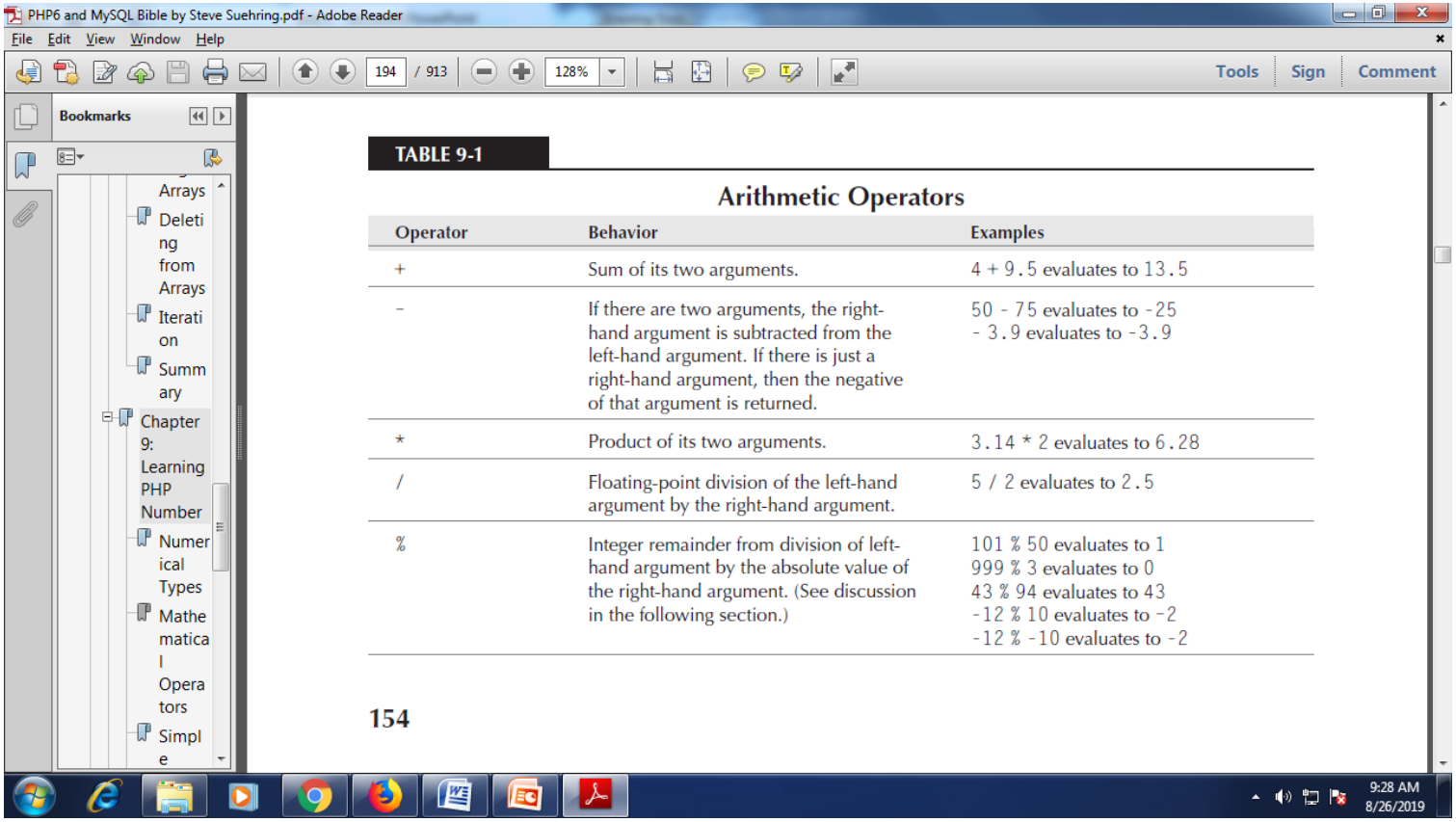
PHP does automatic conversion of numerical types.

In situations where you want a value to be interpreted as a particular numerical type, you can force a typecast by prepending the type in parentheses, such as:

- (double) $my_var
- (integer) $my_var

Or you can use the functions intval() and doubleval(), which convert their arguments to integers and doubles, respectively.

arithmetic operators



Incrementing or decrementing operators:

```
$count = 0;
$result = $count++;
print(“Post ++: count is $count, result is $result<BR>”);
$count = 0;
$result = ++$count;
print(“Pre ++: count is $count, result is $result<BR>”);
$count = 0;
$result = $count--;
print(“Post --: count is $count, result is $result<BR>”);
$count = 0;
$result = --$count;
print(“Pre --: count is $count, result is $result<BR>”);
```

assignment operators:

- **=, +=, -=,*=,/=**

Precedence and parentheses:

- arithmetic operators have higher precedence (that is, bind more tightly) than comparison operators.
- Comparison operators have higher precedence than assignment operators.
- The *, /, and % arithmetic operators have the same precedence.
- The + and – arithmetic operators have the same precedence.
- The *, /, and % operators have higher precedence than + and –.
- When arithmetic operators are of the same precedence, associativity is from left to right (that is, a number will associate with an operator to its left in preference to the operator on its right).

- The < (less than) operator is true if its left-hand argument is strictly less than its right-hand argument but false otherwise.
- The > (greater than) operator is true if its left-hand argument is strictly greater than its right-hand argument but false otherwise.
- The <= (less than or equal) operator is true if its left-hand argument is less than or equal to its right-hand argument but false otherwise.
- The >= (greater than or equal) operator is true if its left-hand argument is greater than or equal to its right-hand argument but false otherwise.
- The == (equal to) operator is true if its arguments are exactly equal but false otherwise.
- The != (not equal) operator is false if its arguments are exactly equal and true otherwise. This operator is the same as <>.
- The === operator (identical to) is true if its two arguments are exactly equal and of the same type.
- The !== operator (not identical to) is true if the two arguments are not equal or not of the same type.

Simple Mathematical Functions

Simple Math Functions

Function	Behavior
`floor()`	Takes a single argument (typically a double) and returns the largest integer that is less than or equal to that argument.
`ceil()`	Short for ceiling — takes a single argument (typically a double) and returns the smallest integer that is greater than or equal to that argument.
`round()`	Takes a single argument (typically a double) and returns the nearest integer. If the fractional part is exactly 0.5, it returns the nearest even number.
`abs()`	Short for absolute value — if the single numerical argument is negative, the corresponding positive number is returned; if the argument is positive, the argument itself is returned.
`min()`	Takes any number of numerical arguments (but at least one) and returns the smallest of the arguments.
`max()`	Takes any number of numerical arguments (but at least one) and returns the largest of the arguments.

Randomness

There are two random number generators (invoked with rand() and mt_rand(), respectively)

```
$length = strlen($string);
$position = mt_rand(0, $length - 1);
return($string[$position]);
```

Randomness

Random Number Functions

Function	Behavior
`srand()`	Takes a single positive integer argument and seeds the random number generator with it.
`rand()`	If called with no arguments, returns a "random" number between 0 and `RAND_MAX` (which can be retrieved with the function `getrandmax()`). The function can also be called with two integer arguments to restrict the range of the number returned — the first argument is the minimum and the second is the maximum (inclusive).
`getrandmax()`	Returns the largest number that may be returned by `rand()`. This number is limited to 32768 on Windows platforms.
`mt_srand()`	Like `srand()`, except that it seeds the "better" random number generator.
`mt_rand()`	Like `rand()`, except that it uses the "better" random number generator.
`mt_getrandmax()`	Returns the largest number that may be returned by `mt_rand()`.

Date Functions

Date and time are some of the most frequently used operations in PHP while executing SQL queries or designing a website etc. PHP serves us with predefined functions for these tasks.

The PHP date() function converts a timestamp to a more readable date and time format.

The computer stores dates and times in a format called UNIX Timestamp, which measures time as number of seconds since the beginning of the Unix epoch (midnight Greenwich Mean Time on January 1, 1970 i.e. January 1, 1970 00:00:00 GMT). Since this is an impractical format for humans to read, PHP converts a timestamp to a format that is readable and more understandable to humans.

date(format, timestamp)

The format parameter in the date() function specifies the format of returned date and time.

Timestamp is an optional parameter, if it is not included then current date and time will be used.

```
$today = date("d/m/Y");

echo $today;
```

Formatting options available in date() function: The format parameter of the date() function is a string that can contain multiple characters allowing to generate dates in various formats.
Date-related formatting characters that are commonly used in format string:

d – Represents day of the month; two digits with leading zeros (01 or 31).

D – Represents day of the week in text as an abbreviation (Mon to Sun).

m – Represents month in numbers with leading zeros (01 or 12).

M – Represents month in text, abbreviated (Jan to Dec).

y – Represents year in two digits (08 or 14).

Y – Represents year in four digits (2008 or 2014).

The parts of the date can be separated by inserting other characters, like hyphens (-), dots (.), slashes (/), or spaces to add additional visual formatting.

PHP time() Function: The time() function is used to get the current time as a Unix timestamp

The following characters can be used along with date() function to format the time string:

h – Represents hour in 12-hour format with leading zeros (01 to 12).

H – Represents hour in in 24-hour format with leading zeros (00 to 23).

i – Represents minutes with leading zeros (00 to 59).

s – Represents seconds with leading zeros (00 to 59).

a – Represents lowercase ante meridian and post meridian (am or pm).

A – Represents uppercase ante meridian and post meridian (AM or PM).

```
echo date("h:i:s") . "\n";
echo date("M,d,Y h:i:s A") . "\n";
echo date("h:i a");
```

```
echo date("d/m/Y") . "\n";
echo date("d-m-Y") . "\n";
echo date("d.m.Y") . "\n";
echo date("d.M.Y/D");

$timestamp = time();
echo($timestamp);
echo "\n";
echo(date("F d, Y h:i:s A", $timestamp));
```

PHP mktime() Function

The mktime() function is used to create the timestamp for a specific date and time.
If no date and time is provided, the timestamp for the current date and time is returned.

mktime(hour, minute, second, month, day, year)

```
echo mktime(23, 21, 50, 11, 25, 2017);
```

PHP getdate() function

The getdate() function return the date and time.

```
$today = getdate();

print_r ($today)."<br/>";

$my_t=getdate (date("U"));

print ("$my_t [weekday], $my_t [month] $my_t [mday], $my_t [year]");
```

PHP - Function checkdate()

This function checks the validity of the date formed by the arguments. A date is considered valid if each parameter is properly defined.

It returns TRUE if the date given is valid; otherwise returns FALSE.

checkdate ($month, $day, $year);

echo checkdate(12, 1, 1990)

echo checkdate(23, 29, 2011)

PHP File Handling

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

$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.

“x+” - New file is created for read-write.

```php
<?php
$file = "data.txt";
// Check the existence of file
if(file_exists($file)){
    // Attempt to open the file
    $handle = fopen($file, "r");
} else{
    echo "ERROR: File does not exist.";
}
?>
```

Closing a File with PHP fclose() Function:

Once you've finished working with a file, it needs to be closed. The fclose() function is used to close the file, as shown in the following example:

```
$handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
```

fclose($handle);

Although PHP automatically closes all open files when script terminates, but it's a good practice to close a file after performing all the operations.

Reading from Files with PHP fread() Function:

PHP has several functions for reading data from a file. You can read from just one character to the entire file with a single operation.

Reading Fixed Number of Characters

The fread() function can be used to read a specified number of characters from a file. The basic syntax of this function can be given with.

fread(file handle, length in bytes)

This function takes two parameter — A file handle and the number of bytes to read. The following example reads 20 bytes from the "data.txt" file including spaces.

```
$file = "data.txt";
// Check the existence of file
if(file_exists($file)){
    // Open the file for reading
    $handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
    // Read fixed number of bytes from the file
    $content = fread($handle, "20");
    // Closing the file handle
    fclose($handle);
    // Display the file content
    echo $content;
} else{
    echo "ERROR: File does not exist.";
}
```

Reading the Entire Contents of a File

The fread() function can be used in conjugation with the filesize() function to read the entire file at once. The filesize() function returns the size of the file in bytes.

$content = fread($handle, filesize($file));

The easiest way to read the entire contents of a file in PHP is with the **readfile() function**. This function allows you to read the contents of a file without needing to open it.

```
if(file_exists($file)){
    // Reads and outputs the entire file
    readfile($file) or die("ERROR: Cannot open the file.");
} else{
    echo "ERROR: File does not exist.";
}
```

Another way to read the whole contents of a file without needing to open it is with the **file_get_contents() function. This function accepts the name and path to a file**, and reads the entire file into a string variable.

```
if(file_exists($file)){

    // Reading the entire file into a string

    $content = file_get_contents($file) or die("ERROR: Cannot open the file.");

}
```

One more method of reading the whole data from a file is the PHP's file() function. It does a similar job to file_get_contents() function, but it returns the file contents as an **array of lines, rather than a single string. Each element of the returned array corresponds to a line in the file.**

To process the file data, you need to iterate over the array using a foreach loop. Here's an example, which reads a file into an array and then displays it using the loop:

```
if(file_exists($file)){
    // Reading the entire file into an array
    $arr = file($file) or die("ERROR: Cannot open the file.");
    foreach($arr as $line){
        echo $line;
    }
}
```

Writing the Files Using PHP fwrite() Function

You can write data to a file or append to an existing file using the PHP fwrite() function.

fwrite(file handle, string)

The fwrite() function takes two parameter — A file handle and the string of data that is to be written, as demonstrated in the following example:

```
$data = “Name of subject is open source programming with PHP.";

 // Open the file for writing

$handle = fopen($file, "w") or die("ERROR: Cannot open the file.");

// Write data to the file

fwrite($handle, $data) or die ("ERROR: Cannot write the file.");
```

In the above example, if the "note.txt" file doesn't exist PHP will automatically create it and write the data. But, if the "note.txt" file already exist, **PHP will erase the contents of this file, if it has any, before writing the new data, however if you just want to append the file and preserve existing contents just use the mode a instead of w** in the above example.

An alternative way is using the **file_put_contents() function. It is counterpart of file_get_contents()** function and provides an easy method of writing the data to a file without needing to open it. This function accepts the name and path to a file together with the data to be written to the file.

file_put_contents($file, $data) or die("ERROR: Cannot write the file.");

If the file specified in the file_put_contents() function already exists, **PHP will overwrite it by default. If you would like to preserve the file's contents you can pass the special FILE_APPEND flag as a third parameter to the file_put_contents() function**. It will simply append the new data to the file instead of overwitting it.

file_put_contents($file, $data, FILE_APPEND)

Renaming Files with PHP rename() Function

```
if(rename($file, "newfile.txt")){
        echo "File renamed successfully.";
}
```

Removing Files with PHP unlink()

You can delete files or directories using the PHP's unlink() function,

```
unlink($file)
```

PHP Filesystem Functions

unction	Description
fgetc()	Reads a single character at a time.
fgets()	Reads a single line at a time.
fgetcsv()	Reads a line of comma-separated values.
filetype()	Returns the type of the file.
feof()	Checks whether the end of the file has been reached.
is_file()	Checks whether the file is a regular file.
is_dir()	Checks whether the file is a directory.
is_executable()	Checks whether the file is executable.
realpath()	Returns canonicalized absolute pathname.
rmdir()	Removes an empty directory.

http://zetcode.com/php/filesysfuns/

ftell( ) Function

The ftell() function in PHP is an inbuilt function which is used to return the current position in an open file.

```
echo ftell($myfile);
```

fseek( ) Function

The *fseek()* function in PHP is an inbuilt function which is used to seek in an open file. It moves the file pointer from its current position to a new position, forward or backward specified by the number of bytes.

```
$myfile = fopen("gfg1.txt", "w");

// reading first line

fgets($myfile);

// moving back to the beginning of the file

echo fseek($myfile, 0);
```

rewind( ) Function

The rewind() function in PHP is an inbuilt function which is used to set the position of the file pointer to the beginning of the file.

```
$myfile = fopen("gfg.txt", "r");

// Changing the position of the file pointer

fseek($myfile, "10");

// Setting the file pointer to 0th

// position using rewind() function

rewind($myfile);
```

copy( ) Function

The copy() function in PHP is an inbuilt function which is used to make a copy of a specified file. It makes a copy of the source file to the destination file and if the destination file already exists, it gets overwritten. The copy() function returns true on success and false on failure.

```
<?php

// Copying gfg1.txt to gfg2.txt

echo copy("gfg1.txt", “gfg2.txt");


?>
```

Session Management using PHP

Why Session Management?

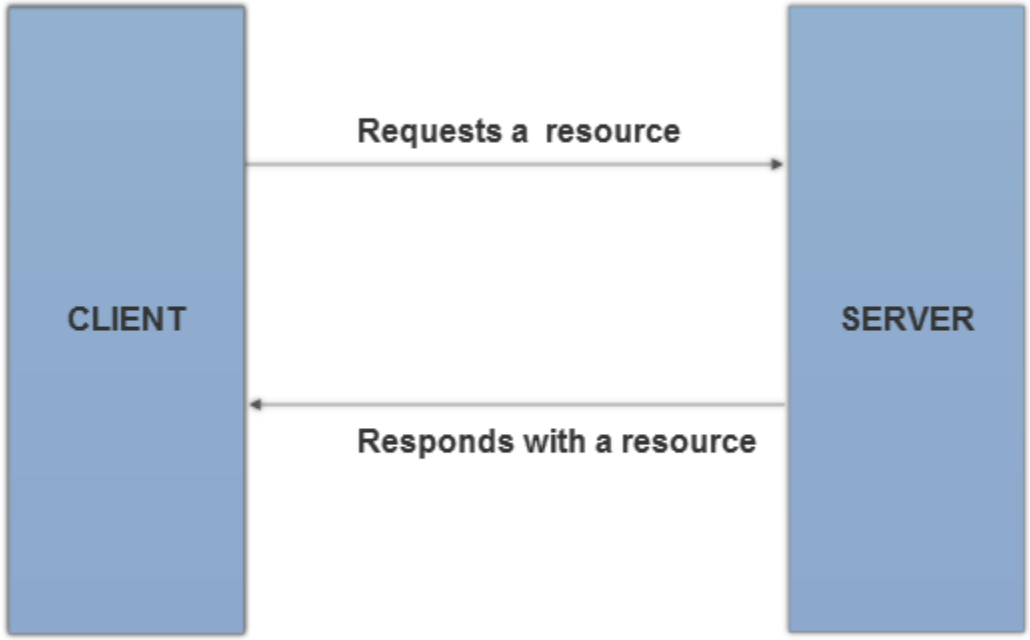
HTTP is state less protocol, and when you refresh the page, it lost everything, which your project should not !

What is stateless?

Stateleass means there's no way a server can remember a specific user between multiple requests.

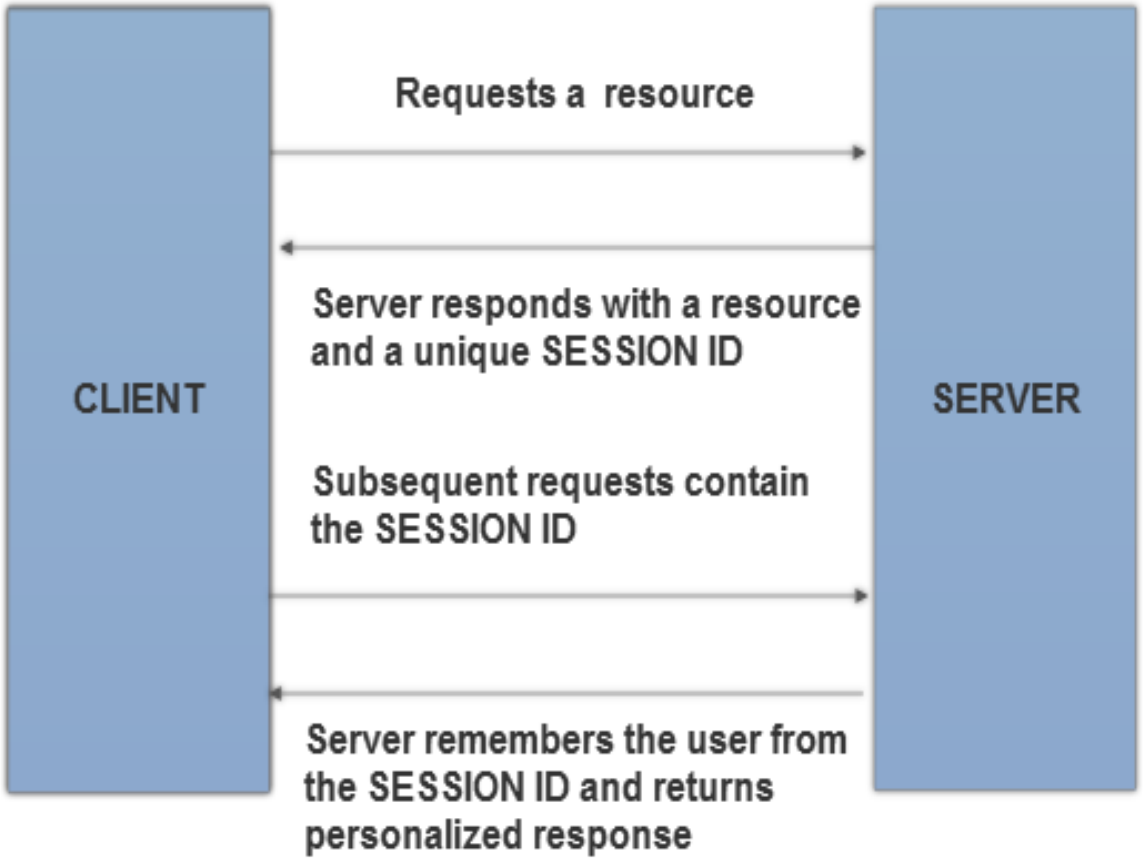
For example, when you access a web page, the server is just responsible for providing the contents of the requested page. So when you access other pages of the same website, the web server interprets each and every request separately, as if they were unrelated to one another.

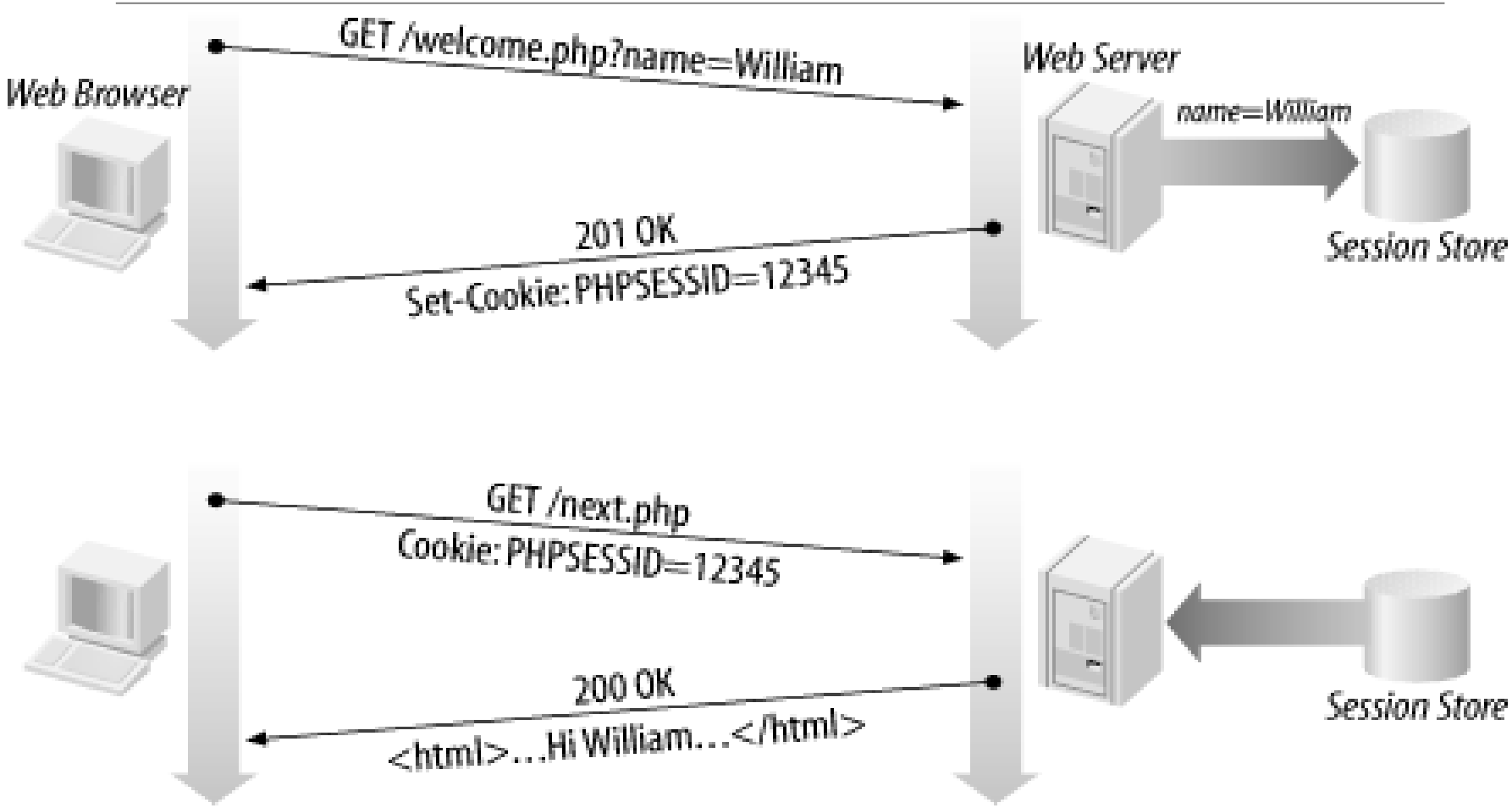
There's no way for the server to know that each request originated from the same user.


CLIENT
Requests a resource
SERVER
Responds with a resource

In this model, if you wanted to display user-specific information, you'd have to authenticate a user in each request. Imagine if you had to type your username and password on every page that displayed your profile information! Yes, it would be cumbersome and not practical at all, and that's where sessions come into the picture.

A session allows you to share information across the different pages of a single site or app—thus it helps maintain state. This lets the server know that all requests originate from the same user, thus allowing the site to display user-specific information and preferences.


CLIENT
SERVER
Requests a resource
Server responds with a resource and a unique SESSION ID
Subsequent requests contain the SESSION ID
Server remembers the user from the SESSION ID and returns personalized response


GET /welcome.php?name=William
Web Browser
Web Server
name=William
Session Store
201 OK
Set-Cookie: PHPSESSID=12345
GET /next.php
Cookie: PHPSESSID=12345
Session Store
200 OK
<html>…Hi William…</html>

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

Login Flow With Sessions

1. A user opens the login page of a website.

2. After submitting the login form, a server on the other end authenticates the request by validating the credentials that were entered.

3. If the credentials entered by the user are valid, the server creates a new session. The server generates a unique random number, which is called a session id. It also creates a new file on the server which is used to store the session-specific information.

4. Next, a session id is passed back to the user, along with whatever resource was requested. Behind the scenes, this session id is sent in the PHPSESSID cookie in the response header.

5. When the browser receives the response from the server, it comes across the PHPSESSID cookie header. If cookies are allowed by the browser, it will save this PHPSESSID cookie, which stores the session id passed by the server.

6. For subsequent requests, the PHPSESSID cookie is passed back to the server. When the server comes across the PHPSESSID cookie, it will try to initialize a session with that session id. It does so by loading the session file which was created earlier during session initialization. It will then initialize the super-global array variable $_SESSION with the data stored in the session file.

Now that you've seen a brief introduction to how sessions work, we'll create a few practical examples to demonstrate how to create and manipulate session variables.

How to Start a Session?

Whenever you want to deal with session variables, you need to make sure that a session is already started. There are a couple of ways you can start a session in PHP.

1. Use the session_start Function

This is the method that you'll see most often, where a session is started by the session_start(); function.

The important thing is that the session_start function must be called at the beginning of the script, before any output is sent to the browser. Otherwise, you'll encounter the infamous ***"Headers are already sent***" error.

2. Automatically Start a Session

If there's a need to use sessions throughout your application, you can also opt in to starting a session automatically without using the session_start function.

There's a configuration option in the php.ini file which allows you to start a session automatically for every request—session.auto_start. By default, it's set to 0, and you can set it to 1 to enable the auto startup functionality.

session.auto_start = 1

On the other hand, if you don't have access to the php.ini file, and you're using the Apache web server, you could also set this variable using the .htaccess file.

php_value session.auto_start 1

How to Get a Session Id: the server creates a unique number for every new session. If you want to get a session id, you can use the session_id function, as shown in the following snippet.

```
session_start();
echo session_id();
```

How to Create Session Variables

Once a session is started, the $_SESSION super-global array is initialized with the corresponding session information. By default, it’s initialized with a blank array, and you ca

```
session_start();
// initialize session variables
$_SESSION['logged_in_user_id'] = '1';
$_SESSION['logged_in_user_name'] = 'Tutsplus';
 // access session variables
echo $_SESSION['logged_in_user_id'];
echo $_SESSION['logged_in_user_name'];n store more information by using a key-value pair.
```

Cookie

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies. Each time the browser requests a page to the server, all the data in the cookie is automatically sent to the server within the request.
Setting a Cookie in PHP
The setcookie() function is used to set a cookie in PHP. Make sure you call the setcookie() function before any output generated by your script otherwise cookie will not set.

setcookie(name, value, expire, path, domain, secure);

If the expiration time of the cookie is set to 0, or omitted, the cookie will expire at the end of the session i.e. when the browser closes.
All the arguments except the name are optional.

Parameter	Description
name	The name of the cookie.
value	The value of the cookie. Do not store sensitive information since this value is stored on the user's computer.
expires	The expiry date in UNIX timestamp format. After this time cookie will become inaccessible. The default value is 0.
path	Specify the path on the server for which the cookie will be available. If set to /, the cookie will be available within the entire domain.
domain	Specify the domain for which the cookie is available to e.g www.example.com.
secure	This field, if present, indicates that the cookie should be sent only if a secure HTTPS connection exists.

```
<?php // Setting a cookie
        setcookie("username", "John Carter", time()+30*24*60*60);
?>
```

Accessing Cookies Values

The PHP $_COOKIE superglobal variable is used to retrieve a cookie value. It typically an associative array that contains a list of all the cookies values sent by the browser in the current request, keyed by cookie name.

The individual cookie value can be accessed using standard array notation.

```
<?php // Accessing an individual cookie value echo
                $_COOKIE["username"];
?>
```

It's a good practice to check whether a cookie is set or not before accessing its value. To do this you can use the PHP isset() function, like this:

```
<?php // Verifying whether a cookie is set or not
if(isset($_COOKIE["username"])){
        echo "Hi " . $_COOKIE["username"];
} else{
        echo "Welcome Guest!";
}
?>
```

Removing Cookies

```
<?php // Deleting a cookie
        setcookie("username", "", time()-3600);
?>
```

Miscellaneous Functions

define function: We can create our own constants using the define() function. The code:

```
define(MY_ANSWER, 42);
```

Constants do not have a $ before their names, and by convention the names of constants usually are in uppercase letters.

There is no way to change this assignment after it has been made, and like variables, user-defined constants that are not part of PHP itself do not persist across pages unless they are explicitly passed to a new page.

When created constants are used, they are generally most usefully defined in an external include file

and might be used for such information as a sales-tax rate or perhaps an exchange rate.

Include and require

It's very common to want to use the same set of functions across a set of web site pages, and the usual way to handle this is with either include or require, both of which import the contents of some other file into the file being executed.

Using either one of these forms is vastly preferable to cloning your function definitions (that is, repeating them at the beginning of each page that uses them); when you want to modify your functions, you will have to do it only once.

For example:

include "basic-functions.inc";

include "advanced-function.inc";

(Note that parentheses are optional with both include() and require().)

Both include and require have the effect of splicing in the contents of their file into the PHP code at the point that they are called.

The only difference between them is how they fail if the file cannot be found.

- The include construct will cause a warning to be printed, but processing of the script will continue;
- require, on the other hand, will cause a fatal error if the file cannot be found.

header() function

The header() function is an inbuilt function in PHP which is used to send a raw HTTP header.

The HTTP functions are those functions which manipulate information sent to the client or browser by the Web server, before any other output has been sent.

The PHP header() function send a HTTP header to a client or browser in raw form. Before HTML, XML, JSON or other output has been sent to a browser or client, a raw data is sent with request (especially HTTP Request) made by the server as header information.

HTTP header provide required information about the object sent in the message body more precisely about the request and response.

The header() functions is used multiple time in the example as one header is allowed to send at a time (since PHP 4.4) to prevent header injection attacks.

Uses:

- Redirect page.
- Set Content-Type in header response
- Set HTTP Status in the header response
- Sent the response to a browser with no cache

```
// Redirect the browser

header("Location: http://www.google.com");
```

```
<?php
header("Expires: Sun, 25 Jul 1997 06:02:34 GMT");
header("Cache-Control: no-cache");
header("Pragma: no-cache");
?>
<html>
        <body>
                <p>Hello World!</p>
                <!-- PHP program to display
                header list -->
                <?php
                        print_r(headers_list());
                ?>
        </body>
</html>
```

```php
<?php
    // We'll be outputting a PDF
    header('Content-Type: application/pdf');
    // It will be called downloaded.pdf
    header('Content-Disposition: attachment; filename="downloaded.pdf"');
    // The PDF source is in original.pdf
    readfile('original.pdf');

?>
```

die() and exit() function

The exit() construct takes either a string or a number as argument, prints out the argument, and then terminates execution of the script.

Everything that PHP produces up to the point of invoking exit() is sent to the client browser as usual, and nothing in your script after that point will even be parsed — execution of the script stops immediately.

If the argument given to exit is a number rather than a string, the number will be the return value for the script's execution. Because exit is a construct, not a function, it's also legal to give no argument and omit the parentheses.

The die() construct is an alias for exit() and so behaves exactly the same way. (We'll usually use the die() version because we find the name more evocative.) So what's the point of exit() and die()?

One possible use is to cut off production of a web page when your script has determined that there is no more interesting information to send, without bothering to wrap up the different branches in a conditional construct.

A better use for die() is to make your crashes informative. It's good to get into the habit of testing for unexpected conditions that would crash your script if they were true, and throw in a die() statement with an informative message.

If you're correct in your expectations, the die() will never be invoked; if you're wrong, you will have an error message of your own rather than a possibly obscure PHP error.

```
$connection = make_database_connection();
if (!$connection)
        die("No database connection!");
use_database_connection($connection);
OR
$connection = make_database_connection()
or die("No database connection!");
use_database_connection($connection);
```

User-defined functions

What is a function?

A function is a way of wrapping up a chunk of code and giving that chunk a name, so that you can use that chunk later in just one line of code.

Functions are most useful when you will be using the code in more than one place, but they can be helpful even in one-use situations, because they can make your code much more readable.

Function definition syntax

```
function function-name ($argument-1, $argument-2, ..)
{
statement-1;
statement-2;
...
}
```

That is, function definitions have four parts:

- The special word function
- The name that you want to give your function
- The function's parameter list — dollar-sign variables separated by commas
- The function body — a brace-enclosed set of statements

Just as with variable names, the name of the function must be made up of letters, numbers, and underscores, and it must not start with a number.

Unlike variable names, function names are converted to lowercase before they are stored internally by PHP, so a function is the same regardless of capitalization

Function naming rules

There are a few rules about which characters are allowed in function names.

Uppercase/lowercase letters

Numbers

Underscores

No spaces

The short version of what happens when a user-defined function is called is:

1. PHP looks up the function by its name (you will get an error if the function has not yet been defined).

2. PHP substitutes the values of the calling arguments into the variables in the definition's parameter list. (Parameters- argument-formal-actual)

3. The statements in the body of the function are executed. If any of the executed statements are return statements, the function stops and returns the given value. Otherwise, the function completes after the last statement is executed, without returning a value.

Argument number mismatches

What happens if you call a function with fewer arguments than appear in the definition, or with more?

PHP handles this without anything crashing, but it may print a warning depending on your settings for error reporting.

Too many arguments

If you hand too many arguments to a function, the excess arguments will simply be ignored

(Global, Local and static variable)

File Upload

PHP allows you to upload single and multiple files through few lines of code only.

PHP file upload features allows you to upload binary and text files both. Moreover, you can have the full control over the file to be uploaded through PHP authentication and file operation functions.

PHP $_FILES

The PHP global $_FILES contains all the information of file. By the help of $_FILES global, we can get file name, file type, file size, temp file name and errors associated with file.

Here, we are assuming that file name is *filename*.

$_FILES['filename']['name'] returns file name.

$_FILES['filename']['type'] returns MIME type of the file.

$_FILES['filename']['size'] returns size of the file (in bytes).

$_FILES['filename']['tmp_name'] returns temporary file name of the file which was stored on the server.

$_FILES['filename']['error'] returns error code associated with this file.

move_uploaded_file() function

The move_uploaded_file() function moves the uploaded file to a new location. The move_uploaded_file() function checks internally if the file is uploaded thorough the POST request. It moves the file if it is uploaded through the POST request.

bool move_uploaded_file (string $filename , string $destination)