# Unit Two

# Server Side Scripting Basics

## Overview of Web Scripting

Web page is a document, typically written in [HTML](file:///C:\topic\html) that is almost always accessible via HTTP. Or pages on which, information is displayed on the web. It can be static or dynamic.

**Static** web page means that what is displayed does not change until the underlying HTML or XML is changed.

When the content that is displayed changes in response to actions taken by the user, then the web page is said to be **dynamic**. This kind of web page is developed using scripting languages.

A **scripting** **language** is a programming language in a simple text format. Generally, code written in a scripting language does not have to be compiled – unlike, for example, code written in C++ or Visual Basic. Instead, it is interpreted at the time of execution. Web browsers can interpret certain scripting languages.

Scripts can be written to run either **server-side** or **client-side.** A script must be interpreted at the time it is requested from the web server. Each scripting language has its own script interpreter – called a **script engine**.

A **client-side script** is executed on the client, by the browser. Client-scripting is often used to validate data entered on a form by the user, before the form is submitted to the server e.g. check that an email address has a **@** sign in it. Some of client side scripts are java script, VB script etc. In client side script, users may be able to see the [source code](https://en.wikipedia.org/wiki/Source_code) by viewing a file that contains the script.

A **server-side script** is executed on the server, and generally produces HTML which is then output HTML to the client.

## Introduction to server-side scripting

**Server-side scripting** is a web server technology in which a user's request is fulfilled by running a script directly on the web server to generate dynamic web pages. It is usually used to provide interactive web sites that interface to databases or other data stores. The primary advantage of server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores. From security point of view, server-side scripts are never visible to the browser as these scripts are executed on the server and emit HTML corresponding to user's input to the page.

In contrast, [server-side scripts](http://en.wikipedia.org/wiki/Server-side_scripting), written in languages such as *PHP, ASP.NET,  Java,  ColdFusion,* [*Perl*](http://en.wikipedia.org/wiki/Perl)*, Ruby,* [*Go*](http://en.wikipedia.org/wiki/Go_(programming_language))*,*[*Python*](http://en.wikipedia.org/wiki/Python_(programming_language))*, and server-side JavaScript*,are executed by the web server when the user requests a document. They produce output in a format understandable by web browsers (usually HTML), which is then sent to the user's computer. The user cannot see the script's source code (unless the author publishes the code separately), and may not even be aware that a script was executed. Documents produced by server-side scripts may, in turn, contain client-side scripts.

Server-side Web scripting is mostly about connecting Web sites to back end servers, such as databases. This enables two-way communication:

1. Client to server: Customer-enter information as request.
2. Server to client: Web pages can be assembled from back end-server to give output.

A server side script can:-

* Dynamically edit, change or add any content to a Web page to make it more useful for individual users
* Respond to user queries or data submitted from HTML forms
* Access any data or databases and return the result to a browser
* Provide security since server side codes cannot be viewed from a browser

In server side script, since the scripts are executed on the server, the browser that displays the file does not need to support scripting at all. The following are **server-side scripting languages**:

* PHP (\*.php)
* Active Server Pages (ASP)
* ANSI C [scripts](http://www.wikipedia.org/wiki/Scripts)
* Java via JavaServer Pages (\*.jsp)
* [JavaScript](http://www.wikipedia.org/wiki/JavaScript) using [Server-side JavaScript](http://www.wikipedia.org/wiki/Server-side_JavaScript) (\*.ssjs)
* Lasso (\*.lasso) etc

The main focus here is PHP.

## What is PHP?

PHP stands for Hypertext Preprocessor and is a server-side language. This means that the script is run on your web server, not on the user's browser, so you do not need to worry about compatibility issues. PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, Microsoft SQL Server etc.). PHP files can contain text, HTML tags and scripts. Like other files written in server side language, PHP files are returned to the browser as plain HTML and it has a file extension of ".php", ".php3", or ".phtml".

## Why PHP?

* Allows easy *storage and retrieval* of information from supported database.
* *Accessibility*: You can reach the internet from any browser, any device, anytime, anywhere.
* *Manageability*: It does not require distribution of application code and it is easy to change code.
* *Security*: The source code is not exposed. Once user is authenticated, can only allow certain actions. It also allows encryption.
* *Scalability*: Web-based 3-tier architecture can scale out

## What you need to get started

Hopefully you have established a basic idea of what server-side scripting is, and when you should use it. Next you need some basic requisites to be able to write and execute scripts. Basically you need:

1. First of all you need a computer with a web server installed. PHP uses web servers such as Apache, IIS etc. But for this course we will install WAMP server. **WAMP server** refers to a software stack for the Microsoft Windows operating system, created by Romain Bourdon and consisting of the *Apache web server*, *OpenSSL* for SSL (Secure Sockets Layer) support, *MySQL* database and *PHP* programming language. WAMP sever has a root directory called WWW. So all the files of a website must be stored under this directory (C:\wamp\www\) to be executed by the server.
2. You need some sort of text editor such as Notepad, Notepad++, etc. to write the scripts.
3. You also need a web browser to display the web content. The web browser can be Internet Explorer, Mozilla Firefox, Opera, and Google Chrome etc.

## PHP Basic Syntax

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as ‘escaping to PHP.’ There are four ways to do this:

1. **Canonical PHP tags**

The most universally effective PHP tag style is:

*<?php*

*Your PHP code here*

*?>*

If you use this style, you can be positive that your tags will always be correctly interpreted.

1. **Short-open (SGML-style) tags:** Short or short-open tags look like this:

*<?*

*Your PHP code here*

*?>*

Short tags are, as one might expect, the shortest option. We must do one of two things to enable PHP to recognize the tags:

* Choose the --enable-short-tags configuration option when building PHP.
* Set the short\_open\_tag setting in php.ini file to on. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

1. **ASP-style tags**: ASP-style tags mimic the tags used by Active Server Pages to delineate code blocks. ASP-style tags look like this:

*<%*

*Your PHP code here*

*%>*

To use ASP-style tags, we should set the configuration option in your php.ini file.

1. **HTML script tags:** HTML script tags look like this: PHP output Statement

*<script language="PHP">*

*Your PHP code here*

*</script>*

## PHP Output Statement

As shown above PHP has different syntaxes but for maximum compatibility, it is recommended to use *<?php…?> .*

Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

There are two basic statements to output text with PHP: **echo** and **print.**

* **echo** has no return value whereas **prin**t has a return value. The returned value represents whether the print statement is succeeded or not. If the print statement succeeds, the statement returns 1 otherwise 0. It is rare that a syntactically correct print statement will fail, but in theory this return value provides a means to test, for example, if the user’s browser has closed the connection and sometimes the returned value can be used in expressions.
* **echo** can take multiple parameters (although such usage is rare) but **print** can only take one argument.
* **echo** is marginally faster than print.
* The **echo** or **print** statement can be used with or without parentheses: echo or echo().

The general format of the echo statement is as follows:

*echo outputitem1,outputitem2,outputitem3, . . .;*

*echo (output statement);*

The parameterized version of echo does not accept multiple arguments.

The general format of the **print** statement is as follows:

*print outputstatement;*

*print(outputstatement);*

Example: different ways of echo and print

*echo 123*; //output: 123

*echo “Hello World!”;* //output: ***Hello world!***

*echo (“Hello World!”);* //output: ***Hello world!***

*echo “Hello”,”World!”;* //output: ***Hello World!***

*echo* *Hello World!;* //output: ***error***, string should be enclosed in quotes

*print (“Hello world!”);* //output: ***Hello world!***

The command print is very similar to echo, with two important differences:

* Unlike echo, print can accept only one argument.
* Unlike echo, print returns a value, which represents whether the print statement succeeded.

It is possible to embed HTML tags in **echo** or **print** statements. The browser will parse and interpret them like any tag included in HTML page and display the page accordingly.

Table1.1 Using echo/print statements

|  |  |  |
| --- | --- | --- |
| **echo/print statement** | **PHP output** | **web page display** |
| *echo “Hello World”;* | HelloWorld! | HelloWorld! |
| *echo “Hello”;*  *echo “World!”;* | HelloWorld! | HelloWorld! |
| *echo “Hello\nWorld!”;* | Hello  World! | Hello World! |
| *echo “Hello<br>World!”;* | Hello<br>World | Hello  World! |
| *echo “<u> <i> Hello world!</i></u>”;* | <u>Hello world!</u> | *Hello world!* |

The first echo statement includes a space so the space is output. The second row has two echo statements, but neither includes a space, so no space appears in the Web page. Each echo statement output does not go to new line unless we insert \n. The third row shows a space on the Web page, even though no space is included in the echo statement. The space is added by the browser when it reads the PHP output as HTML. In HTML, a new line is not displayed as a new line; it is just interpreted as a single space.

**Multi-lines printing:** use the key word **END** next to <<< symbol and before the statement terminating symbol (;) or enclose the statements to be displayed using double quotes.Here are the examples to print multiple lines in a single print statement:

*<?php*

# First Example

***print*** *<<<****END***

*This uses the "here document" syntax to output multiple lines with $variable interpolation. Note that the here document terminator must appear on a line with just a semicolon no extra whitespace!*

***END****;*

# Second Example

***print*** *"This spans multiple lines. The newlines will be*

*output as well";*

*?>*

**Note:** The file must have a .php extension. If the file has a .html extension, the PHP code will not be executed.

## PHP Comments

A *comment* is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

* **Single-line comments:** They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

*<?php*

*# This is a comment*

*// This is a comment too*

print “An example with single line comment”;

*?>*

* **Multi-lines comments:** They are generally used to provide pseudo code algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here is the example of multi lines comments.

*<?php*

*/\* This is a comment with multiline*

*Author : Abebe Kebede*

*Purpose: Multiline Comments Demo*

*Subject: PHP*

*\*/*

*echo "An example with multi line comments";*

*?>*

## Working with Variables

A **variable** is a special container that can be defined to hold a value such as number, string, object, array, or a Boolean. The main way to store information in the middle of a PHP program is by using a variable. Here are the most important things to know about variables in PHP.

* All variables in PHP are denoted with a leading dollar sign ($).
* The value of a variable is the value of its most recent assignment.
* Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
* Variables can, but do not need, to be declared before assignment.
* Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
* Variables used before they are assigned have default values.
* PHP does a good job of automatically converting types from one to another when necessary.

**Variable Naming Rules**

When creating PHP variables, you must follow these four rules:

* Variable names must start with a letter of the alphabet or the *\_* (underscore) character.
* Variable names can contain only the characters: a-z, A-Z, 0-9, and \_ (underscore).
* Variable names may not contain spaces. If a variable must comprise more than one word it should be separated with the *\_* (underscore) character. (e.g., $user\_name).
* Variable names are case-sensitive. The variable $High\_Score is not the same as the variable $high\_score.

The correct way of declaring a variable in PHP is:

*$Variable\_Name=value;*

PHP has a total of eight data types which we use to construct our variables: ***integers, doubles, Booleans, null, strings, arrays, objects and resources*.** The first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

**Integers:**

Integers are whole numbers, without a decimal point, like 3214. They are the simplest type .they correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so:

*$int\_var = 12345;*

*$another\_int = -12345 + 12345;*

**Doubles:**

They are floating point numbers. By default, doubles print with the minimum number of decimal places needed. For example, the code:

*$pi= 3.14;*

*$version=1.12;*

**Boolean:**

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so:

*if (TRUE)*

*print("This will always print<br>");*

*else*

*print("This will never print<br>");*

**NULL:**

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this:

*$my\_var = NULL;*

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed:

*$my\_var = null;*

A variable that has been assigned NULL has the following properties:

* It evaluates to FALSE in a Boolean context.
* It returns FALSE when tested with IsSet() function.

**Strings:**

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

*$string\_1 = "This is a string in double quotes";*

*$string\_2 = "This is a somewhat longer, singly quoted string";*

*$string\_39 = "This string has thirty-nine characters";*

*$string\_0 = ""; // a string with zero characters*

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

*<? php*

*$variable = "name";*

*$literally = 'My $variable will not print!\\n';*

*print($literally);*

*$literally = "My $variable will print!\\n";*

*print($literally);*

*?>*

This will produce the following result:

*My $variable will not print!\n*

*My name will print*

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

* Certain character sequences beginning with backslash (\) are replaced with special characters
* Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are:

* \n is replaced by the newline character
* \r is replaced by the carriage-return character
* \t is replaced by the tab character
* \$ is replaced by the dollar sign itself ($)
* \" is replaced by a single double-quote (")
* \\ is replaced by a single backslash (\)

PHP provides a large number of predefined variables to all scripts. The variables represent everything from external variables to built-in environment variables, last error messages to last retrieved headers.

* **Superglobals** — Superglobals are built-in variables that are always available in all scopes
* **$GLOBALS** — References all variables available in global scope
* **$\_SERVER** — Server and execution environment information
* **$\_GET** — HTTP GET variables
* [**$\_POST**](http://www.php.net/manual/en/reserved.variables.post.php)— HTTP POST variables
* **$\_FILES** — HTTP File Upload variables
* **$\_REQUEST** — HTTP Request variables, and can replace $\_POST, $\_GET and $\_COOKIE variables
* **$\_SESSION** — Session variables
* **$\_COOKIE** — HTTP Cookies
* **$php\_errormsg** — The previous error message
* **$HTTP\_RAW\_POST\_DATA** — Raw POST data
* **$http\_response\_header** — HTTP response headers
* **$argc** — The number of arguments passed to script
* **$argv** — Array of arguments passed to script

Many of these variables, however, cannot be fully documented as they are dependent upon which server are running, the version and setup of the server, and other factors.

**Removing Variables**

* We can uncreated the variable by using this statement: **unset(VariableName);**
* After this statement, the variable $age no longer exists. If we try to echo it, you get an “undefined variable” notice. It is possible to unset more than one variable at once, as follows: **unset($age, $name, $address);**

## Variable Scope:

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

* Local variables
* Function parameters
* Global variables
* Static variables

## PHP Local Variables

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function:

*<?*

*$x = 4;*

*function assignx () {*

*$x = 0;*

*print "\$x inside function is $x. ";*

*}*

*assignx();*

*print "\$x outside of function is $x. ";*

*?>*

This will produce the following result.

*$x inside function is 0.*

*$x outside of function is 4.*

## PHP Function Parameters

PHP Functions are covered in detail in PHP Function Chapter. In short, a function is a small unit of program which can take some input in the form of parameters and does some processing and may return some value.

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be:

*<?*

*// multiply a value by 10 and return it to the caller*

*function multiply ($value) {*

*$value = $value \* 10;*

*return $value;*

*}*

*$retval = multiply (10);*

*Print "Return value is $retval\n";*

*?>*

This will produce the following result.

*Return value is 100*

## PHP Global Variables

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword GLOBAL in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name. Consider an example:

*<?*

*$somevar = 15;*

*function addit() {*

*GLOBAL $somevar;*

*$somevar++;*

*print "Somevar is $somevar";*

*}*

*addit();*

*?>*

This will produce the following result.

*Somevar is 16*

## PHP Static Variables

In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

*<?*

*function keep\_track() {*

*STATIC $count = 0;*

*$count++;*

*print $count;*

*print " ";*

*}*

*keep\_track();*

*keep\_track();*

*keep\_track();*

*?>*

This will produce the following result.

*1*

*2*

*3*

## Work with constants

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use define() function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a $. You can also use the function constant() to read a constant's value if you wish to obtain the constant's name dynamically.

**constant() function**

As indicated by the name, this function will return the value of the constant.

This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function.

**Example**

*<?php*

*define("MINSIZE", 50);*

*echo MINSIZE;*

*echo constant("MINSIZE"); // same thing as the previous line*

*?>*

Only scalar data (boolean, integer, float and string) can be contained in constants.

**Differences between constants and variables are**

* There is no need to write a dollar sign ($) before a constant, where as in Variable one has to write a dollar sign.
* Constants cannot be defined by simple assignment, they may only be defined using the define() function.
* Constants may be defined and accessed anywhere without regard to variable scoping rules.
* Once the Constants have been set, may not be redefined or undefined.

## Manipulate Numbers

In everyday life, numbers are easy to identify. They're 3:00 P.M., as in the current time, or 1.29 Birr, as in the cost of an item. Maybe they're like, the ratio of the circumference to the diameter of a circle. In PHP, numbers can be all these things.

However, PHP doesn't treat all these numbers as "numbers." Instead, it breaks them down into two groups: integers and floating-point numbers. Integers are whole numbers, such as -4, 0, 5, and 1,975. Floating-point numbers are decimal numbers, such as -1.23, 0.0, 3.14159, and 9.9999999999.

Conveniently, most of the time PHP doesn't make you worry about the differences between the two because it automatically converts integers to floating-point numbers and floating-point numbers to integers. This conveniently allows you to ignore the underlying details. It also means 3/2 is 1.5, not 1, as it would be in some programming languages. PHP also automatically converts from strings to numbers and back. For instance, 1+"1" is 2.

***Checking Whether a String Contains a Valid Number:***

You want to ensure that a string contains a number. For example, you want to validate an age that the user has typed into a form input field. **Use is\_numeric( )****:**

*if (is\_numeric('five')) { /\* false \*/ }*

*if (is\_numeric(5)) { /\* true \*/ }*

*if (is\_numeric('5')) { /\* true \*/ }*

*if (is\_numeric(-5)) { /\* true \*/ }*

*if (is\_numeric('-5')) { /\* true \*/ }*

***Rounding Floating-Point Numbers:***

To round a number to the closest integer, use **round( )** :

*$number = round(2.4); // $number = 2*

To round up, use **ceil( ):**

*$number = ceil(2.4); // $number = 3*

To round down, use **floor( ):**

*$number = floor(2.4); // $number = 2*

***Operating on a Series of Integers***

Use the **range( )** function, which returns an array populated with integers:

*foreach(range($start,$end) as $i) {*

*echo “$i<br>”; }*

Instead of using range(), it can be more efficient to use a for loop. Also, you can increment using values other than 1. For example:

*for ($i = $start; $i <= $end; $i += $increment) {*

*echo “$i<br>”; }*

***Generating Random Numbers within a Range***

To generate a random number within a range of numbers: Use **mt\_rand( ):**

*// random number between $upper and $lower, inclusive*

*$random\_number = mt\_rand($lower, $upper);*

***Calculating Exponents***

To raise e to a power, use **exp( ):**

*$exp = exp(2); // 7.3890560989307*

To raise it to any power, use **pow( )****:**

*$exp = pow( 2, M\_E); // 6.5808859910179*

*$pow = pow( 2, 10); // 1024*

***Formatting Numbers***

You have a number and you want to print it with thousands and decimals separators. For instance, you want to display prices for items in a shopping cart.

Use the number\_format( ) function to format as an integer:

*$number = 1234.56;*

*print number\_format($number); // 1,235* *because number is rounded up*

Specify a number of decimal places to format as a decimal:

*print number\_format($number, 2); // 1,234.56*

## Manipulate Strings

Strings in PHP are a sequence of characters, such as "We hold these truths to be self evident," or "Once upon a time," or even "111211211". When you read data from a file or output it to a web browser, your data is represented as strings. The following are string manipulation operations:-

* ***String concatenation operation:*** - To concatenate two string variables together, use the dot (.) operator like echo $string1 . " " . $string2;
* ***Substr()-***uses to copy strings.

Syntax :- *variable = substr(*string*,* start *[,* length *]);*

The *start* argument is the position in *string* at which to begin copying, with 0 meaning the start of the string. The *length* argument is the number of characters to copy (the default is to copy until the end of the string).

For example:

*$name = "Fred Flintstone";*

*$fluff = substr($name, 6, 4);// $fluff is "lint" i.e copy strings from 6th about 4 chars consecutively*

*$sound = substr($name, 11);// $sound is "tone" i.e copy from 11th character on wards and assign to the variable $sound*

* ***substr\_count():-*** uses to count how many times a smaller string occurs in a larger one.

written as ***$number = substr\_count(big\_string,* small\_string*);***

The output is always integer.

For example:

*$sketch = <<< End\_of\_Sketch*

*Well, there's egg and bacon; egg sausage and bacon; egg and spam;*

*egg bacon and spam; egg bacon sausage and spam; spam bacon sausage*

*and spam; spam egg spam spam bacon and spam; spam sausage spam spam*

*bacon spam tomato and spam;*

*End\_of\_Sketch;*

*$count = substr\_count($sketch, "spam");*

*print("The word spam occurs $count times.");*

The word spam occurs 14 times.

* ***substr\_replace():*** permits many kinds of string modifications:

Has syntax of $string = substr\_replace(*original string*, new string, start [, length ]); where start shows starting from where we need replace by new string/ start is the index of the first character that we need to replace. The length parameter uses to indicate how many characters we need to replace from original characters.

The function replaces the part of *original* indicated by the *start* (0 means the start of the string) and *length* values with the string *new*. If no fourth argument is given, substr\_replace( ) removes the text from *start* to the end of the string.

For instance:

*$greeting = "good morning citizen";*

*$farewell = substr\_replace($greeting, "bye", 5, 7);*

*// $farewell is "good bye citizen"*

Use a *length* value of 0 to insert without deleting the original strings:

*$farewell = substr\_replace($farewell, "kind ", 9, 0);*

*// $farewell is "good bye kind citizen"*

Use a replacement of "" to delete without inserting:

*$farewell = substr\_replace($farewell, "", 8);*

*// $farewell is "good bye"*

Here's how we can insert at the beginning of the string without deleting from original character:

Let $*farewell= "good bye"*

*$well = substr\_replace($farewell, "now it's time to say ", 0, 0);*

*//$well is "now it's time to say good bye"*'

A negative value for *start* indicates the number of characters from the end of the string from which to start the replacement:

*$farewell = substr\_replace($farewell, "riddance", -3);*

*// $farewell is "now it's time to say good riddance"*

A negative *length* indicates the number of characters from the end of the string at which to stop deleting:

*$farewell = substr\_replace($farewell, "", -8, -5);*

*// $farewell is "now it's time to say good dance"*

* ***strrev()*** ***function:-*** takes a string and returns a reversed copy of it. Has syntax:-

*$string = strrev(*string*);*

For example:

*echo strrev("There is no cabal");*

labac on si erehT

* ***str\_repeat()*** ***function:-*** takes a string and a count and returns a new string consisting of the argument **string** repeated **count** times.

Written as $*repeated = str\_repeat(*string*,* count*);*

For example, to build a crude horizontal rule:

echo *str\_repeat('-', 40); this will displays 40 dashes*.

* ***str\_pad( )*** ***function****:-* pads one string with another i.e left blank space. Optionally, we can say what string to pad with, and whether to pad on the left, right, or both:

*$padded = str\_pad(to\_pad, length [, with [, pad\_type ]]);*

The default is to pad on the right with spaces:

*$string = str\_pad('Fred Flintstone', 30);*

*echo "$string:35:Wilma";*

**Fred Flintstone :35:Wilma**

The optional third argument is the string to pad with:

*$string = str\_pad('Fred Flintstone', 30, '. ');*

*echo "{$string}35";*

**Fred Flintstone. . . . . . . .35**

The optional fourth argument can be either STR\_PAD\_RIGHT (the default), STR\_PAD\_LEFT, or STR\_PAD\_BOTH (to center). For example:

*echo '[' . str\_pad('Fred Flintstone', 30, ' ', STR\_PAD\_LEFT) . "]\n";*

*echo '[' . str\_pad('Fred Flintstone', 30, ' ', STR\_PAD\_BOTH) . "]\n";*

**[ Fred Flintstone]**

**[ Fred Flintstone ]**

* ***strpos() function:-*** used to search for a string or character within a string. If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Written as *strops(orginal string, new string)*

Example: the following code used to show from where the word “world” started.

*<?php*

*echo strpos("Hello world!","world");*

*?>*

The output will be 6. As seen the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

Example:-

*$long = "Today is the day we go on holiday to Florida";*

*$to\_find = "day";*

*$pos = strpos(strrev ($long), strrev($to\_find));*

*if ($pos === false) {*

*echo("Not found");*

*} else {*

*// $pos is offset into reversed strings*

*// Convert to offset into regular strings*

*$pos = strlen($long) - $pos - strlen($to\_find);;*

*echo("Last occurrence starts at position $pos");*

*}*

Last occurrence starts at position 30

* ***String-Searching Functions***

Several functions find a string or character within a larger string. They come in three families: **strpos() and strrpos(**), which return a position; **strstr() and strchr(),** which return the string they find; and **strspn() and strcspn(),** which return how much of the start of the string matches a mask.

**The strstr()** function finds the first occurrence of a small string in a larger string and returns from that small string on. For instance:

*$record = "Fred,Flintstone,35,Wilma";*

*$rest = strstr($record, ","); // $rest is ",Flintstone,35,Wilma"*

As with strrpos(), strrchr() searches backward in the string, but only for a character, not for an entire string.

* ***Using the strlen() function***

The strlen() function is used to find the length of a string. For example:- To find the length of "Hello world!", we can write as follows

*<?php*

*echo strlen("Hello world!");*

*?>*

**Reading Assignment**

* ***PHP Date and Time***
* ***PHP Operators***