Unit 1 – Web Programming

Web Page:

A web page (HTML file) is a plain text file that uses HTML code tags to instruct web browsers to display text and objects in specific ways. They can be prepared using any plain text editor (such as Notepad) or by using a visual editor such as Webpage.

To create a web site using Webpage, you need to create multiple web pages and link them together.

Website:

A Web site is a related collection of World Wide Web (WWW) files that includes a beginning file called a home page. From the home page, you can get to all the other pages on their site.

World Wide Web (WWW):

The Web, or World Wide Web, is basically a system of Internet servers that support specially formatted documents. The documents are formatted in a markup language called HTML (Hypertext Markup Language) that supports links to other documents, as well as graphics, audio, and video files.

Static Web Page:

Web pages can be either static or dynamic. "Static" means unchanged or constant. static Web pages contain the same prebuilt content each time the page is loaded.

Standard HTML pages are static Web pages. They contain HTML code, which defines the structure and content of the Web page. Each time an HTML page is loaded, it looks the same. The only way the content of an HTML page will change is if the Web developer updates and publishes the file.

Dynamic Web Page:

"Dynamic" means changing or lively. The content of dynamic Web pages can be generated on-the-fly. Web pages, such as PHP, ASP, and JSP pages are dynamic Web pages. These pages contain "server-side" code, which allows the server to generate unique content each time the page is loaded. For example, the server may display the current time and date on the Web page.

Client-side Environment:

The client-side environment used to run scripts is usually a browser. The processing takes place on the end users computer. The source code is transferred from the web server to the users computer over the internet and run directly in the browser.

The scripting language needs to be enabled on the client computer. Sometimes if a user is conscious of security risks, they may switch the scripting facility off. When this is the case a message usually pops up to alert the user when script is attempting to run.

Server-side Environment:

The server-side environment that runs a scripting language is a web server. A user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages. This HTML is then sent to the client browser. It is usually used to provide interactive web sites that interface to databases or other data stores on the server.

This is different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript. The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.

Web server scripting languages

Perl

Perl was developed by Larry Wall in 1987. Perl is a general-purpose scripting language that was found to be very useful for web page scripting. Perl was developed for Unix, but is now available on most platforms. File written in Perl have either a '.pl' or a '.cgi' extension.
Perl is based on several other programming languages notable c, and shell scripts, these show it roots in Unix.

PHP

PHP Hypertext Pre-Processor (PHP), formally known as Personal Home Page (PHP) was developed by Rasmus Lerdorf in 1995 and was based on Perl scripting. PHP is now developed by the PHP Group. PHP is a application that interpreters a web file before being sent to the client. Is application is available for most server platforms and has become a recognised standard in web page production. These files have the extension PHP.

ASP

Active Server Pages (ASP) was developed by Microsoft as there preferred scripting language for their Window NT server and Internet Information Service (IIS) application in 1998.
With the development of the .NET framework, ASP.NET was introduced resolving some of the reliability and speed issues with ASP.
These scripts have the extension of .asp or .aspx for ASP.NET scripts.
ASP is based on Microsoft's VB script, but in ASP.net you can develop in C# or VB and have option to develop in two models Web Forms or MVC.

JSP

Java Server Pages (JSP) was developed by Sun Microsystems in 1999 as a alternative to Asp and PHP.
Based on the java language JSP uses specialised java classes, called servlets to interface with the server. Which is ran using the java runtime. This enables any server platform to run JSP assuming the runtime is installed.

Cold Fusion

Cold fusion is a development environment for web page production and has its associated cold fusion mark-up language (CFML). CFML was developed by Adobe in 1995. CFML is similar in function to the previous scripting language, but it is based on tag like XML and HTML. CFML file have the extension '.cfm'.

Web Server (IIS & Apache):

IIS

IIS (Internet Information Services) is Microsoft's web server. As is expected of a core Microsoft product, it only runs and is bundled on Windows operating systems, but is otherwise free for use. It consists of a series of services including File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP) and others that enable a Windows machine to manage Web sites. The latest version also includes various modules for security, filtering, caching, ASP.net integration, compression, redirects, logging and diagnostics.

Apache HTTP Server

The Apache HTTP server is an open source Web server application developed by the Apache Software Foundation. The server software is freely distributed, and the open source license means users can edit the underlying code to tweak performance and contribute to the future development of the program.
Although Apache will run on pretty much any operating system, it is frequently used in combination with Linux, which is also open source. These two – combined with MySQL and PHP database and scripting language

- **HTTP and HTTPS Protocol:**

HTTP Protocol

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (ie. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extension of its request methods, error codes and headers.

Basically, HTTP is an TCP/IP based communication protocol, which is used to deliver data such as HTML files, image files, query results etc over the Web. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients request data will sent to the server, and how servers respond to these requests.

Understanding the protocol is very important to get capable hands on Security testing. You will be able to appreciate the importance of the protocol when we intercept the packet data between the webserver and the client.

Basic Features

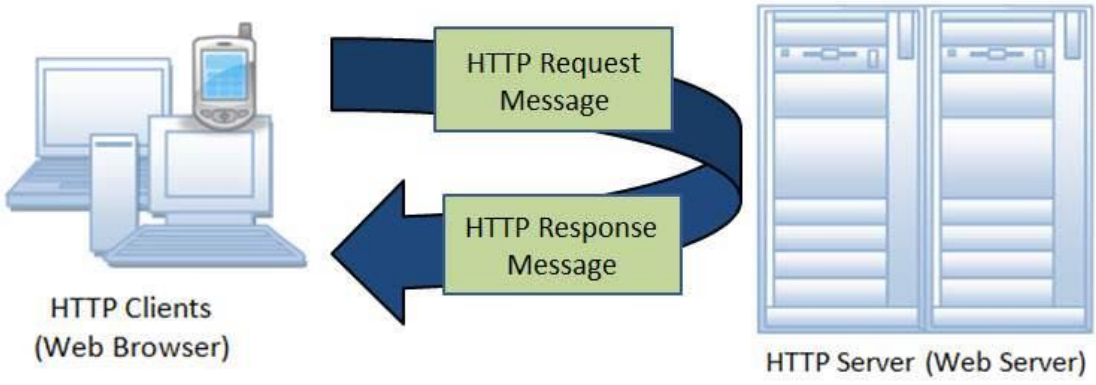
There are following three basic features which makes HTTP a simple but powerful protocol:

- HTTP is connectionless: The HTTP client ie. browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server process the request and re-establish the connection with the client to send response back.
- HTTP is media independent: This means, any type of data can be sent by HTTP as long as both the client and server know how to handle the data content. This is required for client as well as server to specify the content type using appropriate MIME-type.
- HTTP is stateless: As mentioned above, HTTP is a connectionless and this is a direct result that HTTP is a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different request across the web pages.

HTTP/1.0 uses a new connection for each request/response exchange where as HTTP/1.1 connection may be used for one or more request/response exchanges.

Architecture

Following diagram shows a very basic architecture of a web application and depicts where HTTP sits:



The HTTP protocol is a request/response protocol based on client/server based architecture where web browser, robots and search engines, etc. act like HTTP clients and Web server acts as server.

- Client - The HTTP client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a TCP/IP connection.
- Server - The HTTP server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.

Disadvantages

- HTTP is NOT a secured protocol.
- HTTP uses port 80 as default for communication.
- HTTP operates at Application Layer.
- No Encryption/digital certificates required for using HTTP

HTTPS Protocol

HTTPS (Hypertext Transfer Protocol over Secure Socket Layer) or HTTP over SSL is a web protocol developed by Netscape. It is not a protocol but it is just the result of layering the HTTP on top of SSL/TLS (Secure Socket Layer/Transport Layer Security).

Inshort, HTTPS = HTTP + SSL

When Https Required

When we browse, we normally send and receive information using HTTP protocol. So this leads anyone to eavesdrop on the conversation between our computer and the web server. Many a times we need to exchange sensitive information which needs to be secured and to prevent unauthorized access.

Https protocol used in the following scenarios

- Banking Websites
- Payment Gateway
- Shopping Websites

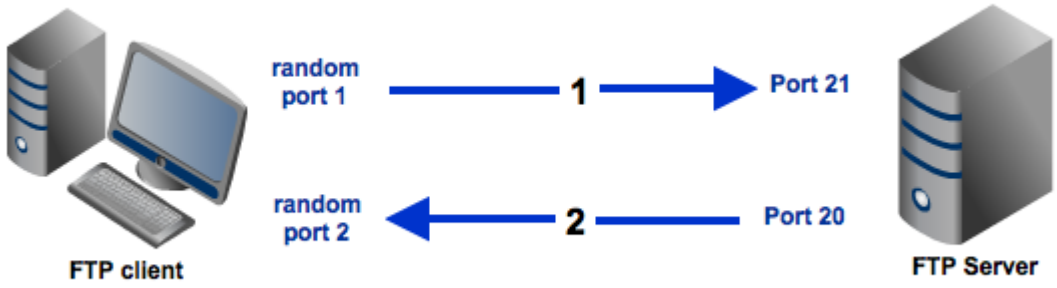
- All Login Pages
- Email Apps

Basic Working of HTTPS

- Public key and signed certificates are required for the server in HTTPS Protocol.
- Client requests for the https:// page
- When using an https connection, the server responds to the initial connection by offering a list of encryption methods the webserver supports.
- In response, the client selects a connection method, and the client and server exchange certificates to authenticate their identities.
- After this is done, both webserver and client exchange the encrypted information after ensuring that both are using the same key, and the connection is closed.
- For hosting https connections, a server must have a public key certificate, which embeds key information with a verification of the key owner's identity.
    - **File Transfer Protocol:**

FTP, File Transfer Protocol, is a protocol through which internet users can upload files from their computers to a website or download files from a website to their PCs. Originated by Abhay Bhushan in 1971 for use in the military and scientific research network known as ARPANET, FTP has evolved into a protocol for far wider applications on the World Wide Web with numerous revisions throughout the years.

FTP is the easiest way to transfer files between computers via the internet, and utilizes TCP, transmission control protocol, and IP, internet protocol, systems to perform uploading and downloading tasks.



How It Works

TCP and IP are the two major protocols that keep the internet running smoothly. TCP manages data transfer while IP directs traffic to internet addresses. FTP is an underling of TCP and shuttles files back and forth between FTP server and FTP client. Because FTP requires that two ports be open--the server's and the client's--it facilitates the exchange of large files of information.

First, you as client make a TCP control connection to the FTP server's port 21 which will remain open during the transfer process. In response, the FTP server opens a second connection that is the data connection from the server's port 20 to your computer.

Using the standard active mode of FTP, your computer communicates the port number where it will stand by to receive information from the controller and the IP address--internet location--from which or to which you want files to be transferred.

If you are using a public--or anonymous--FTP server, you will not need proprietary sign-in information to make a file transfer, but you may be asked to enter your email address. If you are using a private FTP server, however, you must sign in with a user name and password to initiate the exchange of data.

- **ISP and It's Services:**

Short for Internet Service Provider, it refers to a company that provides Internet services, including personal and business access to the Internet. For a monthly fee, the service provider usually provides a software package, username, password and access phone number. Equipped with a modem, you can then log on to the Internet and browse the World Wide Web and USENET, and send and receive e-mail. For broadband access you typically receive the broadband modem hardware or pay a monthly fee for this equipment that is added to your ISP account billing.

In addition to serving individuals, ISPs also serve large companies, providing a direct connection from the company's networks to the Internet. ISPs themselves are connected to one another through Network Access Points (NAPs). ISPs may also be called IAPs (Internet Access Providers).

Internet Service Providers deliver a variety of services to their customers. Some of these services include Internet access, domain name registration, domain name hosting, dial-up access, leased line access and co-location. As most know, the Internet is a global system of interconnected computer networks used by billions of people across the world. This system of networks is linked by a vast assortment of electronic, optical networking and wireless technologies.

Web Hosting

Web hosting is a service that allows organizations and individuals to post a website or web page on to the Internet. A web host, or web hosting service provider, is a business that provides the technologies and services needed for the website or webpage to be viewed in the Internet. Websites are hosted, or stored, on special computers called servers.
When Internet users want to view your website, all they need to do is type your website address into their browser. Their computer will then connect to your server and your webpages will be delivered to them through the browser.
Most hosting companies require that you own your domain name to host with them. If you do not have a domain name, the hosting companies will help you purchase one.

Virtual Hosting

Virtual hosting is a method for hosting multiple domain names (with separate handling of each name) on a single server (or pool of servers). This allows one server to share its resources, such as memory and processor cycles, without requiring all services provided to use the same host name. The term virtual hosting is usually used in reference to web servers but the principles do carry over to other internet services.

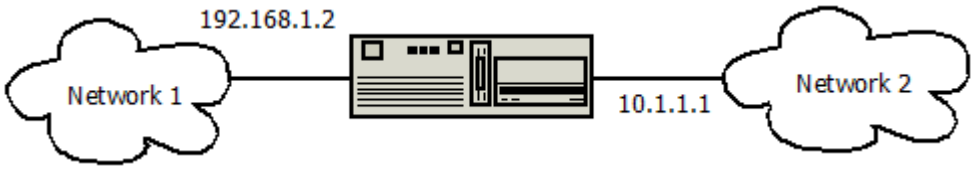
There are two main types of virtual hosting, name-based and IP-based. Name-based virtual hosting uses the host name presented by the client. This saves IP addresses and the associated administrative overhead but the protocol being served must supply the host name at an appropriate point. In particular, there are significant difficulties using name-based virtual hosting with SSL/TLS. IP-based virtual hosting uses a separate IP address for each host name, and it can be performed with any protocol but requires a dedicated IP address per domain name served. Port-based virtual hosting is also possible in principle but is rarely used in practice because it is unfriendly to users.

Name-based and IP-based virtual hosting can be combined: a server may have multiple IP addresses and serve multiple names on some or all of those IP addresses. This technique can be useful when using SSL/TLS

with wildcard certificates. For example, if a server operator had two certificates, one for *.example.com and one for *.example.net, he could serve foo.example.com and bar.example.com off the same IP address but would need a separate IP address for baz.example.net.

Multihoming

Simply put, multihoming is giving your computing device or network a presence on more than one network. In fact, many people already have a multihomed device: the router connecting their home network to the Internet. It is also used in network bridges, repeaters, range extenders, firewalls, proxy servers, gateways, and when using a virtual machine configured to use network address translation (NAT).



Advantages of Multihoming

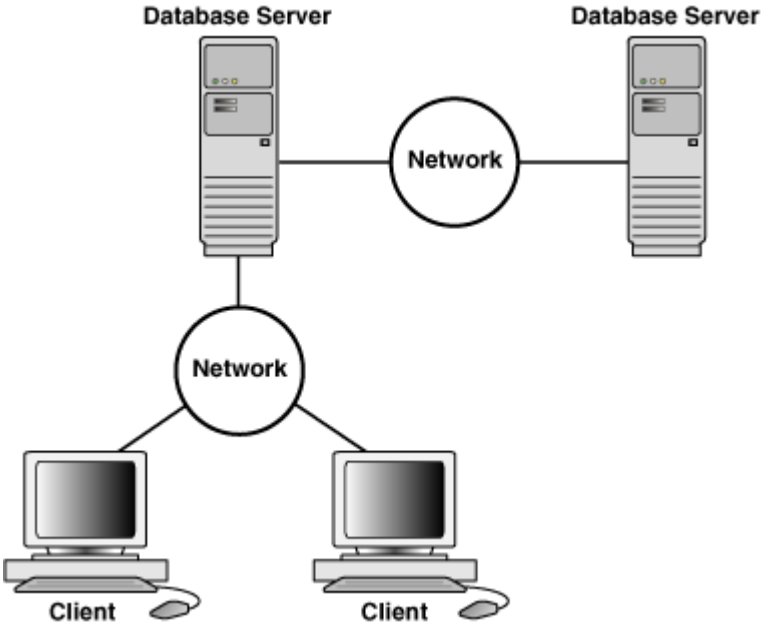
Let's look at an example of multihoming your network as a whole. This would involve a connection from two ISPs. Now all of the machines on your network have two paths to the Internet. All of the traffic directed to the Internet can be sent via either link thereby providing load-balancing and relieving congestion along any single link. As long as both links are up, you can fully saturate both links increasing overall throughput. And if one of the links fails, the other still carries traffic so you have redundancy.

Another instance where I use multihomed hosts is for network backups. Since I do not want these backups to cause congestion on my production network, my computers all have two network cards. One is connected to a switch or router on the production network, and the second is connected to a switch on my dedicated backup network. All network backups happen over this second network keeping the production network clear for the "real work" even if I am working during the scheduled backups.

- **Distributed Web Server:**

Companies with sensitive or high availability needs will want to deploy multiple, fully-redundant servers to maintain continuous operation even in the face of serious disruptions. Multiple Servers also help to distribute the load and create a more efficient deployment. Here is a bare-bones diagram of how multiple servers might be set up to provide redundancy.

Grid computing is a computing model involving a distributed architecture of large numbers of computers connected to solve a complex problem. In the grid computing model, servers or personal computers run independent tasks and are loosely linked by the Internet or low-speed networks.



- **Document Root :**

The document root is a directory (a folder) that is stored on your host's servers and that is designated for holding web pages. When someone else looks at your web site, this is the location they will be accessing.

In order for a website to be accessible to visitors, it must be published to the correct directory, the "document root."

You might think that there would only be one directory in your space on your host's servers, but often hosts provide services beyond just publishing a website. In this case, they are likely to set up every account with several directories, since each service would require its own.

If you log into your host you might see a number of directories as shown below:

Remember, every host is different, so yours will probably not look exactly like the picture above. Yours is quite likely to have a completely different set of folders. It might even have none.

In the example shown above, the document root is the "www" directory, but of course your host may well use a different folder. Some of the more common alternatives are:

- htdocs
- httpdocs
- html
- public_html
- web
- empty — no value (such as GoDaddy-hosted sites)

Your host may even have several folders nested inside each other, one of which is the document root. In the example above the document root was:

/www/

but your host may use something like one of these:

/web/public/
/example.com/www/public/
/example.com/public_html/

Web Services

JSON

- **Introduction**

JSON — short for JavaScript Object Notation. It is a lightweight data-interchange format. It is based on a subset of the JavaScript programming language. JSON is derived from the JavaScript programming language, but it's available for use by many languages including Python, Ruby, PHP, and Java.
JSON uses the .json extension when it stands alone. When it's defined in another file format (as in .html), it can appear inside of quotes as a JSON string, or it can be an object assigned to a variable. This format is easy to transmit between web server and client or browser.

- **Syntax and Structure**

JSON is built on two structures :

- **A collection of name/value pair**

In various languages, this is realized as an object, record, struct, dictionary, keyed list or associative array. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).
Example : An object with three properties named "a", "b", and "c"

{ "a":1, "b":2, "c":3 }

- **An ordered list of values**

In most languages, this is realized as an array, vector, list, or sequence. An array begin with [(left bracket) and ends with] (right bracket). Values are separated by , (comma).
Example : An array of three integers and one string value.

[1,2,3, "value with"]

- **Installation & Configuration**

As of PHP 5.2.0, the JSON extension is bundled and compiled into PHP by default. So the need of installation is not required.

- **Resource Types**

A resource accessed through this convention is represented as a JSON value with an "application/json" Internet media type. It is also referred to as MIME (Multipurpose Internet Mail Extensions).
A collection of resources must have a unique location. Each resource must have a unique identifier within a collection. The reject requests that have identifiers that do not confirm to a 403 Forbidden status code.

- **JSON Serializable**

JsonSerialzable::jsonSerialize – Specify data which should be serialized to JSON. It is an interface of JSON Serializable.

Syntax :

abstract public mixed JsonSerializable::jsonSerialize (void)

The function has no parameter and returns data which can be serialized by json_encode(), which is a value of any type other than a resource.

Example :

Use of jsonSerialize() which returns array()

```
<?php
Class ArrayValue implements JsonSerializable {
        public function __construct(array $array) {
                $this->array = $array;
        }
        public function jsonSerialize() {
                return $this->array;
        }
}
$array = [1,2,3];
Echo json_encode(new ArrayValue($array), JSON_PRETTY_PRINT);
?>
```

Output :

[1,2,3]

- **Json_encode()**

json_encode -- Encodes a JSON string.

Syntax :

string json_encode (mixed $value)

Returns a string that contains JSON representation of $value.

$value : The value being encoded. Can be any type except a resource.

This function only works with UTF-8 encoded data.

Example :

```
<?php
$arr=array('a'=>1, 'b'=>2, 'c'=>3, 'd'=>4, 'e'=>5);
echo json_encode($arr);
$arr=array(1,2,3,4,5);
echo json_encode($arr);
?>
```

Output :

["a":1,"b":2,"c":3,"d":4,"e":5]

[1,2,3,4,5]

- **Json_decode()**

json_decode -- Decodes a JSON string.

Syntax :

string json_decode (string $json, bool $assoc)

It accepts a JSON encoded string and converts it into a PHP value.

$json : The JSON string being decoded

$assoc : false (default), return the value as an associative array.

Example :

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5)';
var_dump(json_decode($json));
var_dump(json_decode($json, true));
?>
```

Output :

```
object(stdClass)#1 (5) {
["a"]=> int(1)
["b"]=> int(2)
["c"]=> int(3)
["d"]=> int(4)
["e"]=> int(5)
}

Array(5) {
["a"]=> int(1)
["b"]=> int(2)
["c"]=> int(3)
["d"]=> int(4)
["e"]=> int(5)
}
```

Unit 2 – PHP Basics

➔ WHAT IS PHP?

The full form of php is “Hypertext PreProcessor”. Its original name is “personal home page”.

- PHP is a powerful server-side scripting language for creating dynamic and interactive websites.
- PHP is the widely used, free, and efficient alternative to competitors such as Microsoft's ASP. PHP is perfectly suited for Web development and can be embedded directly into the HTML code.
- The PHP syntax is very similar to Perl and C. PHP is often used together with Apache (web server) on various operating systems.
- It also can be used with Microsoft's IIS on Windows.

➔ INTRODUCTION:

A PHP file may contain text, HTML tags and scripts. Scripts in a PHP file are executed on the server.

- What is PHP?
  - PHP stands for PHP: Hypertext Preprocessor
  - PHP is a server-side scripting language, like ASP
  - PHP scripts are executed on the server
  - PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
  - PHP is an open source software (OSS)
  - PHP is free to download and use

- What is a PHP File?
  - PHP files may contain text, HTML tags and scripts
  - PHP files are returned to the browser as plain HTML
  - PHP files have a file extension of ".php", ".php3", or ".phtml"

- What is MySQL?
  - MySQL is a database server
  - MySQL is ideal for both small and large applications
  - MySQL supports standard SQL
  - MySQL compiles on a number of platforms
  - MySQL is free to download and use

- PHP + MySQL
  - PHP combined with MySQL are cross-platform (means that you can develop in Windows and serve on a Unix platform)
- Why PHP?
  - PHP runs on different platforms (Windows, Linux, Unix, etc.)
  - PHP is compatible with almost all servers used today (Apache, IIS, etc.)
  - PHP is FREE to download from the official PHP resource: www.php.net
  - PHP is easy to learn and runs efficiently on the server side

ADVANTAGE OF PHP

- Cost
  - PHP costs you nothing. It is open source software and doesn't need to purchase it for development.

- Ease of Use
  - PHP is easy to learn, compared to the other ways to achieve similar functionality. Unlike Java Server Pages or C-based CGI, PHP doesn't require you to gain a deep understanding of a major programming language.
  - Many of the most useful specific functions are predefined for you. A lot of it is possible to use PHP just by modifying freely available scripts rather than writing

- HTML-embedded ness
  - PHP is embedded within HTML. In other words, PHP pages are ordinary HTML pages that escape into PHP mode only when necessary.
  - When a client requests this page, the Web server preprocesses it. This means it goes through the page from top to bottom, looking for sections of PHP, which it will try to resolve. For one thing, the parser will suck up all assigned variables (marked by dollar signs) and try to plug them into later PHP commands (in this case, the echo function). If everything goes smoothly, the preprocessor will eventually return a normal HTML page to the client's browser.

- Cross-platform compatibility
  - PHP and MySQL run native on every popular flavor of Unix (including Mac OS X) and Windows.
  - PHP is compatible with the three leading Web servers: Apache HTTP Server for UNIX and Windows, Microsoft Internet Information Server, and Netscape Enterprise Server (a.k.a.i Planet Server).
  - It also works with several lesser-known servers, including Alex Belits' fhttpd, Microsoft's Personal Web Server, AOLServer, and Omnicentrix's Omni server application server.

- Not tag-based
  - PHP is a real programming language. It is a bunch of predefined tags, like HTML? In PHP, you can define functions just by typing a name and a definition.

- Stability
  - The word stable means two different things in this context:
    - The server doesn't need to be rebooted often.
    - The software doesn't change radically and incompatibly from release to release.

- Speed
  - PHP is much faster in execution than CGI script.

➔ **Understanding of PHP.INI file**

Anyone who has a server using PHP has undoubtedly heard of php.ini – it's the configuration file used to control and customize PHP's run-time behavior. It provides a simple way to configure settings for such things as:

- Upload Directories
- Log Errors
- Maximum Script execution time
- File Upload limit

… and so much more.

php.ini is the first file PHP looks for when starting up because of the importance of the configuration directives it sets. However if you make changes to your php.ini, it will requires a server reboot before the changes take effect.

A pre-made php.ini file with recommended settings ships with PHP. Many hosting providers who support PHP allow you some way of customizing php.ini directives so you can tweak PHP's behavior just the way you like it. In these instances, PHP only uses the values for the directives which you have specified in your custom php.ini file; any settings you don't declare in your file are assigned their respective values from the original php.ini and PHP's built-in defaults.

In this article I'll give an overview of some important settings I believe you should be concerned with when tweaking your own php.ini file. This is just a personal selection of course, and everyone will develop their own preferences over time, so I recommend you do some solid research into the myriad of configuration directives available in php.ini.

The location of php.ini depends on the server and how PHP was installed. For the purposes of this article I'll assume it's at /usr/local/lib/php/php.ini. Your path may be different. To find where your php.ini file is located, you can use the phpinfo() function.

➔ **Understanding of PHP .htaccess file**

.htaccess is a configuration file for use on web servers running the Apache Web Server software. When a .htaccess file is placed in a directory which is in turn 'loaded via the Apache Web Server', then the .htaccess file is detected and executed by the Apache Web Server software. These .htaccess files can be used to alter the configuration of the Apache Web Server software to enable/disable additional functionality and features that the Apache Web Server software has to offer. These facilities include basic redirect functionality, for instance if a 404 file not found error occurs, or for more advanced functions such as content password protection or image hot link prevention.

➔ **PHP SYNTAX**

- Basic PHP Syntax
  - A PHP scripting block always starts with <?php and ends with ?>.
  - A PHP scripting block can be placed anywhere in the document.
  - On servers with shorthand support enabled you can start a scripting block with <? And end with?>.
  - However, for maximum compatibility, we recommend that you use the standard form (<?php ?>) rather than the shorthand form.

- A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.
- Below, we have an example of a simple PHP script which sends the text "Hello World" to the browser:

- 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.
- In the example above we have used the echo statement to output the text "Hello World".

Comments in PHP

- In PHP, we use // can be used make a single-line comment or /* and */ to make a large comment block.

➔ **VARIABLES**

- Variables in PHP is things that store data which begin with $ followed by a letter or an underscore, then any combination of letters, numbers, and the underscore character.
- This means you may not start a variable with a number.
- Variables are case-sensitive, which means that $Foo is not the same variable as $foo, $FOO, or $fOO. Assigning variables is as simple as using the assignment operator (=) on a variable, followed by the value you want to assign.

Note: Variable must begin with a dollar sign ($) and are followed by a meaningful name. the variable name cannot begin with a numeric character, but it can contains numbers and underscore character(_). Variable are case sensitive.

To concatenate two or more variables together, use the dot (.) operator:

Variable Naming Rules

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (a-Z, 0-9, and _)
- A variable name should not contain spaces. If a variable name should be more than one word, it should be separated with underscore ($my_string), or with capitalization ($myString)

Variable scope

- The scope of a variable is the context within which it is defined.
- For the most part all PHP variables only have a single scope. This single scope spans included and required files as well

- However, within user-defined functions a local function scope is introduced. Any variable used inside a function is by default limited to the local function scope.
- This script will not produce any output because the echo statement refers to a local version of the $a variable, and it has not been assigned a value within this scope.
- You may notice that this is a little bit different from the C language in that global variables in C are automatically available to functions unless specifically overridden by a local definition.
- This can cause some problems in that people may inadvertently change a global variable.

The global keyword

- In PHP global variables must be declared global inside a function if they are going to be used in that function.
- By declaring $a and $b global within the function, all references to either variable will refer to the global version First, an example use of global:
- There is no limit to the number of global variables that can be manipulated by a function.
- A second way to access variables from the global scope is to use the special PHP-defined $GLOBALS array. The previous example can be rewritten as:

Using static variables

- Another important feature of variable scoping is the static variable.
- A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope. Consider the following example:

- This function is quite useless since every time it is called it sets $a to 0 and prints "0".
- The $a++ which increments the variable serves no purpose since as soon as the function exits the $a variable disappears.
- To make a useful counting function which will not lose track of the current count, the $a variable is declared static:

    - Now, $a is initialized only in first call of function and every time the test() function is called it will print the value of $a and increment it.

➔ **GET AND POST METHOD:**

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand (&).

Spaces are removed and replaced with the + character and any other non-alphanumeric characters are replaced with a hexadecimal values. After the information is encoded it is sent to the server.

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character.

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send up to 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides $_GET associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides $_POST associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

➔ **PHP OPERATORS:**

Operators are used to operate on values.

Arithmetic Operators

Arithmetic Operators		
Example	Name	Result
-$a	Negation	Opposite of $a.
$a + $b	Addition	Sum of $a and $b.
$a - $b	Subtraction	Difference of $a and $b.
$a * $b	Multiplication	Product of $a and $b.
$a / $b	Division	Quotient of $a and $b.
$a % $b	Modulus	Remainder of $a divided by $b.

- The division operator ("/") returns a float value unless the two operands are integers (or strings that get converted to integers) and the numbers are evenly divisible, in which case an integer value will be returned.
- Operands of modulus are converted to integers (by stripping the decimal part) before processing.
- Remainder $a % $b is negative for negative $a.

Increment/decrement Operators

Increment/decrement Operators		
Example	Name	Effect
++$a	Pre-increment	Increments $a by one, then returns $a.
$a++	Post-increment	Returns $a, then increments $a by one.
--$a	Pre-decrement	Decrements $a by one, then returns $a.
$a--	Post-decrement	Returns $a, then decrements $a by one.

Here's a simple example script:

Incrementing or decrementing Booleans has no effect.

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Comparison Operators

Comparison operators, as their name implies, allow you to compare two values. You may also be interested in viewing the type comparison tables, as they show examples of various type related comparisons.

Comparison Operators		
Example	Name	Result
$a== $b	Equal	TRUE if $a is equal to $b.
$a=== $b	Identical	TRUE if $a is equal to $b, and they are of the same type. (introduced in PHP 4)
$a != $b	Not equal	TRUE if $a is not equal to $b.
$a<> $b	Not equal	TRUE if $a is not equal to $b.

$a!==$b	Not identical	TRUE if $a is not equal to $b, or they are not of the same type. (introduced in PHP 4)
$a < $b	Less than	TRUE if $a is strictly less than $b.
$a > $b	Greater than	TRUE if $a is strictly greater than $b.
$a<= $b	Less than or equal to	TRUE if $a is less than or equal to $b.
$a>= $b	Greater than or equal to	TRUE if $a is greater than or equal to $b.

If you compare an integer with a string, the string is converted to a number. If you compare two numerical strings, they are compared as integers. These rules also apply to the switch statement.

Logical Operators

Logical Operators		
Example	Name	Result
$a and $b	And	TRUE if both $a and $b are TRUE.
$a or $b	Or	TRUE if either $a or $b is TRUE.
$a xor $b	Xor	TRUE if either $a or $b is TRUE, but not both.
! $a	Not	TRUE if $a is not TRUE.
$a && $b	And	TRUE if both $a and $b are TRUE.
$a || $b	Or	TRUE if either $a or $b is TRUE.

The reason for the two different variations of "and" and "or" operators is that they operate at different precedence.

Operator Precedence

- The precedence of an operator specifies how "tightly" it binds two expressions together. For example, in the expression 1 + 5 * 3, the answer is 16 and not 18 because the multiplication ("*") operator has a higher precedence than the addition ("+") operator. Parentheses may be used to force precedence, if necessary. For instance: (1 + 5) * 3 evaluates to 18. If operator precedence is equal, left to right associativity is used.
- The following table lists the precedence of operators with the highest-precedence operators listed at the top of the table. Operators on the same line have equal precedence, in which case their associativity decides which order to evaluate them in.
- Left Associativity means that the expression is evaluated from left to right, right associativity means the opposite.

Operator Precedence		
Associativity	Operators	Additional Information
non-associative	clone new	clone and new
Left	[	array()
non-associative	++ --	increment/decrement
non-associative	~ - (int) (float) (string) (array) (object) (bool) @	types
non-associative	Instanceof	types

Right	!	logical
Left	* / %	arithmetic
Left	+ - .	arithmetic and string
Left	<< >>	bitwise
non-associative	< <= > >= <>	comparison
non-associative	== != === !==	comparison
Left	&	bitwise and references
Left	^	bitwise
Left	|	bitwise
Left	&&	logical
Left	||	logical
Left	? :	ternary
Right	= += -= *= /= .= %= &= |= ^= <<= >>=	assignment
Left	And	logical
Left	Xor	logical
Left	Or	logical
Left	,	many uses

String Operators

There are two string operators. The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side. Please read Assignment Operators for more information.

- print_r — Prints human-readable information about a variable

syntax:

mixed print_r (mixed $expression [, bool $return])

print_r() displays information about a variable in a way that's readable by humans.

print_r(), var_dump() and var_export() will also show protected and private properties of objects with PHP 5. Static class members will not be shown.

Remember that print_r() will move the array pointer to the end. Use reset() to bring it back to beginning.

Parameters

expression

The expression to be printed.

return

If you would like to capture the output of print_r(), use the return parameter. If this parameter is set to TRUE, print_r() will return its output, instead of printing it (which it does by default).

Return Values

If given a string, integer or float, the value itself will be printed. If given an array, values will be presented in a format that shows keys and elements. Similar notation is used for objects.

➔ **Conditional and looping structure:**

CONDITIONAL STRUCTURE:

In PHP following Conditional structures are used.

IF STATEMENT
IF ELSE STATEMENT
ELSE IF STATEMENT
SWITCH CASE

IF STAEMENT:

- The if construct is one of the most important features of many languages, PHP included.
- It allows for conditional execution of code fragments.
- If expression evaluates to TRUE, PHP will execute statement, and if it evaluates to FALSE - it'll ignore it.
- The following example would display a is bigger than b if $a is bigger than $b:

IF ELSE STATEMENT:

- Often you'd want to execute a statement if a certain condition is met, and a different statement if the condition is not met.
- This is what else is for. else extends an if statement to execute a statement in case the expression in the if statement evaluates to FALSE.

ELSE IF STATEMENT:

- elseif, as its name suggests, is a combination of if and else.

- Like else, it extends an if statement to execute a different statement in case the original if expression evaluates to FALSE. However, unlike else, it will execute that alternative expression only if the elseif conditional expression evaluates to TRUE.

Switch

- The switch statement is similar to a series of IF statements on the same expression.
- In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to.

➔ LOOPING:

WHILE

while loops are the simplest type of loop in PHP.
The basic form of a while statement is:

Syntax:

while (expr)
statement

- The meaning of a while statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the while expression evaluates to TRUE.
- The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time PHP runs the statements in the loop is one iteration).
- Sometimes, if the while expression evaluates to FALSE from the very beginning, the nested statement(s) won't even be run once.
- Like with the if statement, you can group multiple statements within the same while loop

by surrounding a group of statements with curly braces, or by using the alternate syntax:

do-while

- do-while loops are very similar to while loops, except the truth expression is checked at the end of each iteration instead of in the beginning.
- The do...while loop will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

FOR LOOP

The for loop is used when you know how many times you want to execute a statement or a list of statements.

Note:
The for statement has three parameters.

The first parameter initializes variables, the second parameter holds the condition, and the third parameter contains the increments required to implement the loop.
If more than one variable is included in the initialization or the increment parameter, they should be separated by commas. The condition must evaluate to true or false.

foreach

- This simply gives an easy way to iterate over arrays. foreach works only on arrays, and will issue an error when you try to use it on a variable with a different data type or an uninitialized variable.

The first form loops over the array given by array_expression. On each loop, the value of the current element is assigned to $value and the internal array pointer is advanced by one (so on the next loop, you'll be looking at the next element).

break and continue

- PHP gives you the break and continue keywords to control loop operation.
- We already used break previously when we looked at case switching it was used there to exit a switch/case block, and it has the same effect with loops.
- When used inside loops to manipulate the loop behavior, break causes PHP to exit the loop and carry on immediately after it, and continue causes PHP to skip the rest of the current loop iteration and go on to the next.

- When the current number is 3, continue is used to skip the rest of that iteration and go on to Number 4.
- Also, if the number is 7, break is used to exit the loop.

➔ **ARRAY**

- An array is a list variable that is a variable that contains multiple elements indexed by number or string.
- It enables you to store, ordered and access many values under one name.
- Each item is an array is commonly referred to as an element.
- Each element can be accessed directly via its index.
- An index to an array element can be either number or string.
- By default array elements are indexed by number starting at zero (0).
- It is important to remember that the index of last element of numerically index array is always one less that numbers of elements the array contains values can be assigned to an array in two ways:
- With the array function or

E.g.

```
$users= Array(“A”, “B”,”C”, “D”, “E”);
print $users[0];
print $users[1];
```

print $users[2];

- Directly using the array identifiers.

E.g.
$ users[]= “A”;
$ users[]= “B”;
$ users[]= “C”;
$ users[]= “D”;

- Here, we do not need to place any number between the square brackets.
- PHP automatically takes squares of the index no, which saves you from having to workout which is the next available stat.

There are three different kinds of arrays:

- **Numeric array** - An array with a numeric ID key
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

NUMERIC ARRAYS

A numeric array stores each element with a numeric ID key. There are different ways to create a numeric array.

1. In this example the ID key is automatically assigned:
2. In this example we assign the ID key manually:

```
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
```

ASSOCIATIVE ARRAYS

- An associative array, each ID key is associated with a value.
- When storing data about specific named values, a numerical array is not always the best way to do it.
- With associative arrays we can use the values as keys and assign values to them.

MULTIDIMENSIONAL ARRAYS

- In a multidimensional array, each element in the main array can also be an array.
- And each element in the sub-array can be an array, and so on.
- In this example, we create a multidimensional array, with automatically assigned ID keys:

print_r() function

print_r() function is mainly used to print array values. You can also print normal variables with

the help of print_r().

The main difference between var_dump() function and print_r() function is, var_dump() prints value of variable along with its data type and size where as print_r() just prints the value of variable.

If you print array value with print_r() function, it prints key values or index values of array along with the values assigned to the array. It does not print data type and size of the value stored in array

USER DEFINE FUCTION:

A function is a block of code that can be executed whenever we need it.

1. CREATING SIMPLE PHP USER DEFINE FUNCTION:
2. All functions start with the word "function()"
3. Name the function - It should be possible to understand what the function does by its name. The name can start with a letter or underscore (not a number)
4. Add a "{" - The function code starts after the opening curly brace
5. Insert the function code
6. Add a "}" - The function is finished by a closing curly brace

ADDING PARAMETERS TO A FUNCTION:

- Our first function (writeMyName()) is a very simple function.
- It only writes a static string.
- To add more functionality to a function, we can add parameters.
- A parameter is just like a variable.
- You may have noticed the parentheses after the function name, like: writeMyName().
- The parameters are specified inside the parentheses.

VARIABLE LENGTH ARGUMENT FUNCTION:

PHP support 3 different types of variable length argument function.

func_num_args() – Returns the number of argument passed to the function.
func_get_arg() – Returns the item from array specified in argument.
func_get_args() – Returns and array for all arguments.

FUNC_NUM_ARGS():

Purpose: This function returns the number of arguments passed to the function.

Syntax: int
func_num_args(void)

Example:

```
<?php
function f1()
{
   $numargs = func_num_args();
```

```
    echo "Number of arguments: $numargs\n";
}
f1(1, 2, 3);    // Prints 'Number of arguments: 3'
?>
```

FUNC_GET_ARG():
Purpose: Gets the specified argument from a user-defined function's argument list.
This function may be used in conjunction with func_get_args() and func_num_args() to allow user-defined functions to accept variable-length argument lists.

Syntax: mixed func_get_arg($arg_num)

Example:

```
<?php
    function f1()
    {
            $numargs = func_num_args();
    echo "Number of arguments: $numargs<br />\n";
    if ($numargs >= 2)
    {
                    echo "Second argument is: " . func_get_arg(1) . "<br />\n";
            }
    }f1 (1, 2, 3);
    ?>
```

FUNC_GET_ARGS()

Purpose:

- Gets an array of the function's argument list.
- This function may be used in conjunction with func_get_arg() and func_num_args() to allow user-defined functions to accept variable-length argument lists.
  Returns an array in which each element is a copy of the corresponding member of the current user-defined function's argument list.

  Syntax: array func_get_args(void)

  Example:

```
<?php
        function f1(){
                $numargs = func_num_args();
                echo "Number of arguments: $numargs<br />\n";
                If ($numargs >= 2) {
                                echo "Second argument is: " . func_get_arg(1) . "<br />\n";
        }
        $arg_list = func_get_args();
        for ($i = 0; $i < $numargs; $i++) {
                 echo "Argument $i is: " . $arg_list[$i] . "<br />\n";
        }
```

```
}
f1(1, 2, 3);
?>
```

➔ **VARIABLE FUNCTION:**

gettype():

Purpose:
This function returns the data type of variable that is passed to function.
Possible values for the returned string are:

- "Boolean"
- "integer"
- "double"
- "string"
- "array"
- "object"
- "resource"
- "NULL"
- "unknown type"

Syntax: string gettype(mixed $variable name);

Example:

```
<?php
$data = array(1, 1., NULL, new stdClass, 'abc');
foreach ($data as $value)
{
        echo gettype($value), "\n";
}
?>
```

The above example will output something similar to:

```
integer
double
NULL
object
string
```

settype():
Purpose: It is used to change data type of specified variable.

Syntax: bool settype (mixed $var, string type);

- $var – specified the name of variable to be converted in to another data type;

- Type – specified the data type that u want to converted.
- This function returns either TRUE on success and FALSE on failure.

Example

```
<?php
        $f = "5bar"; // string
        $b = true;   // boolean
        settype($f, "integer"); // $f is now 5   (integer)
        settype($b, "string");  // $b is now "1" (string)
?>
```

Isset()

Purpose:

- Determine if a variable is set and is not NULL.
- If a variable has been unset with unset(), it will no longer be set. isset() will return FALSE if testing a variable that has been set to NULL.
- Also note that a NULL byte ("\0") is not equivalent to the PHP NULL constant.

- If multiple parameters are supplied then isset() will return TRUE only if all of the parameters are set. Evaluation goes from left to right and stops as soon as an unset variable is encountered.

Syntax: bool isset (mixed $var [, mixed $var…..]);

Returns TRUE if var exists; FALSE otherwise.

Example:

```
<?php
$var = '';
// This will evaluate to TRUE so the text will be printed.
if (isset($var))
{
   echo "This var is set so I will print.";
}
?>
```

strval():

Purpose:

- Get the string value of a variable.
- User can pass any scalar type variable.
- User can not pass arrays or objects as argument.
- This function returns string value.

Syntax: string strval (mixed $var);

floatval():

Purpose:

- Get the float value of a variable.
- User can pass any scalar type variable.
- User can not pass arrays or objects as argument.
- This function returns float value.

Syntax: string floatval (mixed $var);

Example:

```
<?php
        $var = '122.34343The';
        $float_value_of_var = floatval($var);
        echo $float_value_of_var; // 122.34343
?>
```

intval():

Purpose:

- Get the integer value of a variable.
- User can pass any scalar type variable.
- User can not pass arrays or objects as argument.
- This function returns integer value.

Syntax: int intval (mixed $var [, int $base]

Returns the integer value of var, using the specified base for the conversion (the default is base 10).

Parameters

var - The scalar value being converted to an integer
base - The base for the conversion (default is base 10)

Example:

```
<?php
echo intval(42);              // 42
echo intval(4.2);             // 4
echo intval('42');            // 42
echo intval('+42');           // 42
echo intval('-42');           // -42
?>
```

print_r():

Purpose:

- It is used to displays information about a variable in a way that's readable by humans.
- Remember that print_r() will move the array pointer to the end. Use reset() to bring it back to beginning.

Syntax: mixed print_r (mixed $expression [, bool $return]))

Parameters

expression - The expression to be printed.

return - If you would like to capture the output of print_r(), use the return parameter. If this parameter is set to TRUE, print_r() will return its output, instead of printing it (which it does by default).

Example:

```
<?php
        $a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
        print_r ($a);
?>
```

unset ():

Purpose:

- It is used to destroy the specified variables.
- The behavior of unset() inside of a function can vary depending on what type of variable you are attempting to destroy.
- If a globalized variable is unset() inside of a function, only the local variable is destroyed. The variable in the calling environment will retain the same value as before unset() was called.

Syntax: unset(mixed $var [,mixed $var…])

Example:

```
<?php
        function destroy_f()
        {
                global $f;
                unset($f);
        }
        $f = 'bar';
        destroy_f();
        echo $f;
?>
```

If you would like to unset() a global variable inside of a function, you can use the $GLOBALS

STRING FUNCTION:

chr()

Purpose:

- This function returns the one character of specified ascii value.

Syntax: string chr (ascii value)

Example:

```
<?php
        echo chr(65);
?>
```

Output: A

ord()

Purpose:

- Returns the ASCII value of the first character of string

Syntax: int ord (string $string)

Example:

```
<?php
        echo ord("abc");
?>
```

Output: 97

strtoupper()

Purpose:

- Returns string with all alphabetic characters converted to uppercase.

Syntax: string strtopper (string $string)

Example:

```
<?php
        echo strtoupper("this is example");
?>
```

Output: THIS IS EXAMPLE

strlen()

Purpose:

This function returns the length of specified stirng.

Syntax: int strlen (string $string)

Example:

```
<?php
        echo strlen("first");
```

?>
Output: 5

ltrim()
Purpose:

- This function removes whitespace (or other character) from the left side (beginning) of the string.
- This function returns a string with whitespace stripped from the beginning of str .
- Without the second parameter, ltrim() will strip these characters:

  - " " (ASCII 32 (0x20)), an ordinary space.
  - "\t" (ASCII 9 (0x09)), a tab.
  - "\n" (ASCII 10 (0x0A)), a new line (line feed).
  - "\r" (ASCII 13 (0x0D)), a carriage return.
  - "\0" (ASCII 0 (0x00)), the NUL-byte.
  - "\x0B" (ASCII 11 (0x0B)), a vertical tab.

Example:

```
<?php
        echo ltrim("____GEETANJALI____")."<BR>";
        echo ltrim("_____Welcome");
?>
```

Output: GEETANJALI_____
Welcome

rtrim():

Purpose:

- This function removes whitespace (or other character) from the right side (ending) of the string.
- This function returns a string with whitespace stripped from the ending of str.

Syntax: string rtrim (string $str [, string $charlist])

Example:

```
<?php
        echo rtrim("____GEETANJALI____")."<BR>";
        echo rtrim("_____Welcome");
?>
```

Output:____GEETANJALI
____Welcome

trim():

Purpose:

- This function removes whitespace (or other character) from both side of string (at beginning or at ending)
- This function returns a string with whitespace stripped from the beginning and end of str.

Syntax: string trim (string $str [, string $charlist])

Example:

```
<?php
        echo rtrim("____GEETANJALI____")."<BR>";
        echo rtrim("_____Welcome");
        ?>

        Output:GEETANJALI
         Welcome
```

ubstr

Purpose:

This function returns the portion of string specified by the start and length parameters.

Syntax: string **substr** (string $string , int $start [, int $length])

- If start is non-negative, the returned string will start at the start 'th position in string , counting from zero.
- For instance, in the string 'abcdef', the character at position 0 is 'a', the character at position 2 is 'c', and so forth.
- If start is negative, the returned string will start at the start 'th character from the end of string .
- If string is less than or equal to start characters long, FALSE will be returned.

Example:

```
<?php
$rest = substr("abcdef", -1);    // returns "f"
$rest = substr("abcdef", -2);    // returns "ef"
$rest = substr("abcdef", -3, 1); // returns "d"

$rest = substr("abcdef", 0, -1);  // returns "abcde"
```

```
$rest = substr("abcdef", 2, -1);  // returns "cde"
$rest = substr("abcdef", 4, -4);  // returns ""
$rest = substr("abcdef", -3, -1); // returns "de"

echo substr('abcdef', 1);     // bcdef
echo substr('abcdef', 1, 3);  // bcd
echo substr('abcdef', 0, 4);  // abcd
echo substr('abcdef', 0, 8);  // abcdef
echo substr('abcdef', -1, 1); // f
?>
```

strcmp();

Purpose:
This function is used to compare two specified string.

Syntax: int strcmp (string $str1 , string $str2)

- Returns < 0 if str1 is less than str2 ; > 0 if str1 is greater than str2 , and 0 if they are equal.
- Comparison is case sensitive.

Example:

```
<?php
    echo strcmp("Hello","hello");          //-1
    ?>
```

strrcmp();

Purpose:
This function is used to compare two specified string.

Syntax: int strcmp (string $str1 , string $str2)

Returns < 0 if str1 is less than str2 ; > 0 if str1 is greater than str2 , and 0 if they are equal.
Note: this function is case insensitive.

Example:

```
<?php
$var1 = "Hello";
$var2 = "hello";
if (strcasecmp($var1, $var2) == 0)
{
                    echo '$var1 is equal to $var2 in a case-insensitive string comparison';
```

```
}
?>
```

strpos():

Purpose:

- It is used to find position of first occurrence of a string

Syntax: int **strpos** (string $string, mixed $character [, int $starting pos])

- In this function user have to give main string as first argument.
- Second argument is character that you want to find the position.
- Third argument is the starting position to fine particular character.
- Third argument is optional and by default value is zero.

Example:

```
<?php
$newstring = 'abcdef abcdef';
$pos = strpos($newstring, 'a'); // $pos = 0
 $pos = strpos($newstring, 'a', 1); // $pos = 7, not 0
?>
```

strrpos()

- This function is used to find position of last occurrence of a char in a string

Syntax: int **strrpos** (string $string, mixed $character [, int $starting pos])

- Returns the numeric position of the last occurrence of character in the main string. If a string is passed as the character, then only the first character of that string will be used.
- If character is not found, returns FALSE.

Example:

```
<?php
        $str = 'abcdef abcdef";
        $pos = strrpos($str,'a');  //$pos = 7
?>
```

strstr():

Purpose:

- This Find first occurrence of a string

Syntax: String strstr (string $main string, mixed $sub string [, bool $before_needle])

- Returns part of main string from the first occurrence of sub string to the end of main string.
- This function is case-sensitive. For case-insensitive searches, use stristr().
- If before_needle is TRUE (the default is FALSE), strstr() returns the part of the main string before the first occurrence of the substring.

Example

```
<?php
        $email  = 'name@example.com';
        $domain = strstr($email, '@');
        echo $domain; // prints @example.com
        $user = strstr($email, '@', true);
        echo $user; // prints name
?>
```

➔ **MATH FUNCTION:**

abs():

Purpose:

- This function returns the absolute value of specified number.
- If the argument number is of type float, the return type is also float.

Example:

```
<?php
        $abs = abs(-4.2); // $abs = 4.2; (double/float)
        $abs2 = abs(5);   // $abs2 = 5; (integer)
        $abs3 = abs(-5);  // $abs3 = 5; (integer)
?>
```

ceil():

Purpose:

- This function returns the next highest integer value by rounding up value if necessary.

Syntax: float ceil(float $value);

Example:

```
<?php  echo ceil(4.3);    // 5
       echo ceil(9.999);  // 10
       echo ceil(-3.14);  // -3
?>
```

floor():

Purpose:

This function is used to return nearest smallest integer value of specified number.

Syntax: float floor (float $value);

Example:

```
<?php
       echo floor(4.3);   // 4
       echo floor(9.999); // 9
       echo floor(-3.14); // -4
?>
```

round():

Purpose:

- Returns the rounded value of val to specified precision (number of digits after the decimal point).
- precision can also be negative or zero (default).

Syntax: float round (float $val [, int $precision])

Example:

```
<?php  echo round(3.4);       // 3
       echo round(3.5);       // 4
       echo round(3.6);       // 4
       echo round(3.6, 0);    // 4
       echo round(1.95583, 2);  // 1.96
       echo round(1241757, -3); // 1242000
       echo round(5.045, 2);    // 5.05
       echo round(5.055, 2);    // 5.06
?>
```

fmod()

Purpose:

- Returns the floating point remainder of dividing the dividend (x) by the divisor (y)

Syntax: float fmod (float $x, float $y);

Example:

```
<?php
       $x = 5.7;
       $y = 1.3;
```

```
$r = fmod($x, $y);
?>
```

min():

Purpose:

- min() returns the numerically lowest of the parameter values.
- If the first and only parameter is an array, min() returns the lowest value in that array.
- If at least two parameters are provided, min() returns the smallest of these values.
- PHP will evaluate a non-numeric string as 0 if compared to integer, but still return the string if it's seen as the numerically lowest value.
- If multiple arguments evaluate to 0, min() will return the lowest alphanumerical string value if any strings are given, else a numeric 0 is returned.

Syntax:

mixed min (array $values)
mixed min (mixed $value1 , mixed $value2 [, mixed $value3...])

Example:

```
<?php  echo min(2, 3, 1, 6, 7);  // 1
echo min(array(2, 4, 5)); // 2
echo min(0, 'hello');    // 0
echo min('hello', 0);    // hello
echo min('hello', -1);   // -1
// With multiple arrays, min compares from left to right
// so in our example: 2 == 2, but 4 < 5
$val = min(array(2, 4, 8), array(2, 5, 1)); // array(2, 4, 8)
// If both an array and non-array are given, the array
// is never returned as it's considered the largest
$val = min('string', array(2, 5, 7), 42);   // string              ?>
```

max():

Purpose:

- max() returns the numerically highest of the parameter values.
- If the first and only parameter is an array, max() returns the highest value in that array.
- If at least two parameters are provided, max() returns the biggest of these values.
- PHP will evaluate a non-numeric string as 0 if compared to integer, but still return the string if it's seen as the numerically highest value.
- If multiple arguments evaluate to 0, max() will return a numeric 0 if given, else the alphabetical highest string value will be returned.

Syntax
mixed max (array $values)
mixed max (mixed $value1 , mixed $value2 [, mixed $value3…])

Example:

```
<?php
        echo max(1, 3, 5, 6, 7);  // 7
        echo max(array(2, 4, 5)); // 5
        echo max(0, 'hello');     // 0
        echo max('hello', 0);     // hello

    echo max(-1, 'hello');    // hello
    // With multiple arrays, max compares from left to right
    // so in our example: 2 == 2, but 4 < 5
    $val = max(array(2, 4, 8), array(2, 5, 7)); // array(2, 5, 7)
    // If both an array and non-array are given, the array
    // is always returned as it's seen as the largest
    $val = max('string', array(2, 5, 7), 42);   // array(2, 5, 7)          ?>
```

pow():

Purpose:

- Returns base raised to the power of exp.
- If the power cannot be computed FALSE will be returned instead.

Syntax: number **pow** (number $base , number $exp)

Example:

```
<?php
        echo pow(-1, 20); // 1
        echo pow(0, 0); // 1
        echo pow(2, 3);//8
 ?>
```

sqrt():

Purpose:

- This function returns the square root of specified number.

Syntax: float sqrt(float $arg)

Example:

```
<?php
        // Precision depends on your precision directive
        echo sqrt(9); // 3
```

```
        echo sqrt(10); // 3.16227766 ...
    ?>
```

rand():

Purpose:

- This function is used to generate random integer number.
- If called without the optional min, max arguments rand() returns a integer.
- If you want a random number between 5 and 15 (inclusive), for example, use rand (5, 15).

Syntax:

```
int rand ( void )
int rand ( int $min, int $max)
```

Example:

```
<?php
    echo rand() . "\n";
    echo rand() . "\n";
    echo rand(5, 15);
?>
```

The above example will output something similar to:

7771

22264

12

➔ **DATE FUNCTION:**

Date():-

- The PHP date() function is used to format date or time
- The PHP date() formats a timestamp to a more readable date and time.

syntax:

string date (string $format [, int $timestamp])

- Returns a string formatted according to the given format string using the given integer timestamp or the current time if no timestamp is given. In other words, timestamp is optional and defaults to the value of time().
- The valid range of a timestamp is typically from fri, 13 dec 1901 20:45:54 GMT to true,19 jan 2038 03:14:07 GMT(these are the dates that corruspond to the minimum and maximum values for a 32-bit signed integer.)
- Different formats of PHP date() and gmdate() functions

Parameter Description format Required. Specifies how to return the result:

- d - The day of the month (from 01 to 31)
- D - A textual representation of a day (three letters)
- j - The day of the month without leading zeros (1 to 31)

- l (lowercase 'L') - A full textual representation of a day
- N - The ISO-8601 numeric representation of a day (1 for Monday through 7 for Sunday)
- S - The English ordinal suffix for the day of the month (2 characters st, nd, rd or th. Works well with j)
- w - A numeric representation of the day (0 for Sunday through 6 for Saturday)
- z - The day of the year (from 0 through 365)
- W - The ISO-8601 week number of year (weeks starting on Monday)
- F - A full textual representation of a month (January through December)
- m - A numeric representation of a month (from 01 to 12)
- M - A short textual representation of a month (three letters)
- n - A numeric representation of a month, without leading zeros (1 to 12)
- t - The number of days in the given month
- L - Whether it's a leap year (1 if it is a leap year, 0 otherwise)
- o - The ISO-8601 year number
- Y - A four digit representation of a year
- y - A two digit representation of a year
- a - Lowercase am or pm
- A - Uppercase AM or PM
- B - Swatch Internet time (000 to 999)
- g - 12-hour format of an hour (1 to 12)
- G - 24-hour format of an hour (0 to 23)
- h - 12-hour format of an hour (01 to 12)
- H - 24-hour format of an hour (00 to 23)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds, with leading zeros (00 to 59)
- e - The timezone identifier (Examples: UTC, Atlantic/Azores)
- I (capital i) - Whether the date is in daylights savings time (1 if Daylight Savings Time, 0 otherwise)
- O - Difference to Greenwich time (GMT) in hours (Example: +0100)
- T - Timezone setting of the PHP machine (Examples: EST, MDT)
- Z - Returns 0
- c - The ISO-8601 date (e.g. 2004-02-12T15:19:21+00:00)
- r - The RFC 2822 formatted date (e.g. Thu, 21 Dec 2000 16:01:07 +0200)
- U - The seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)

Example:

```
<?php
        echo("Result with date():<br />");
        echo(date("l") . "<br />");
        echo(date("l dS \of F Y h:i:s A") . "<br />");
        echo("Oct 3,1975 was on a ".date("l", mktime(0,0,0,10,3,1975))."<br />");
        echo(date(DATE_RFC822) . "<br />");
        echo(date(DATE_ATOM,mktime(0,0,0,10,3,1975)) . "<br /><br />");
        echo("Result with gmdate():<br />");
        echo(gmdate("l") . "<br />");
        echo(gmdate("l dS \of F Y h:i:s A") . "<br />");
        echo("Oct 3,1975 was on a ".gmdate("l", mktime(0,0,0,10,3,1975))."<br />");
        echo(gmdate(DATE_RFC822) . "<br />");
        echo(gmdate(DATE_ATOM,mktime(0,0,0,10,3,1975)) . "<br />");
?>
```

The output of above example will be:-

Result with date():
Tuesday
Tuesday 24th of January 2006 02:41:22 PM
Oct 3,1975 was on a Friday
Tue, 24 Jan 2006 14:41:22 CET
1975-10-03T00:00:00+0100
Result with gmdate():
Tuesday
Tuesday 24th of January 2006 01:41:22 PM
Oct 3,1975 was on a Thursday
Tue, 24 Jan 2006 13:41:22 GMT
1975-10-02T23:00:00+0000

Getdate():

- This function gets date and time information.

Syntax: array getdate ([int $timestamp])

- Returns an associative array containing the date information of the timestamp , or the current local time if no timestamp is given.
- Key elements of the returned associative array

Example:

```
<?php
        $today=getdate();
        print_r($today);
?>
```

The output of above code is:-

```
Array
(
        [seconds]=>40
        [minutes]=>58
        [hours]=>21
        [mday]>=17
        [wday]>=2
        [mon]>=6
        [year]>=2003
        [yday]>=167
        [weekday]>=Tuesday
        [month]>=june
        [0]>=1055901520
)
```

checkdate():

- This function is used to validate gregorian date.

Syntax: Bool checkdate(int $month, int $day, int $year)

- returns true if the given date is valid otherwise false.Checks the validity of the date formed by the arguments.
- A date is considered valid if each parameter is properly defined.

time():

Purpose:-

- This function returns current Unix timestamp.

Syntax: int time (void)

- Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

Example:

```
<?php
        $nextWeek=time()+(7 * 24 * 60 * 60); // 7 days; 24 hours; 60 mins; 60secs
        echo 'Now: '. date('Y-m-d') ."\n";
        echo 'Next Week: '. date('Y-m-d', $nextWeek) ."\n";// or using strtotime():
        echo 'Next Week: '. date('Y-m-d', strtotime('+1 week')) ."\n";
?>
```

The output of above code will be:-

```
Now:      2005-03-30
Next Week: 2005-04-06
Next Week: 2005-04-06
```

mktime():

- This function is used to get unix timestamp for a date.

Syntax:

int mktime ([int $hour[,int $minute[,int $second[,int $month[,int $day[,int $year[,int $is_dst]]]]]]])

- Returns the Unix timestamp corresponding to the arguments given. This timestamp is a long integer containing the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.
- Arguments may be left out in order from right to left; any arguments thus omitted will be set to the current value according to the local date and time.

➔ **ARRAY FUNCTION:**

count():-

Purpose:-

- This function is used to count elements in an array, or properties in an object.

Syntax: int count (mixed $var [, int $mode])

- Counts elements in an array, or properties in an object.
- If var is not an array or an object with implemented Countable interface, 1 will be returned. There is one exception, if var is NULL, 0 will be returned.
- count() may return 0 for a variable that isn't set, but it may also return 0 for a variable that has been initialized with an empty array. Use isset() to test if a variable is set.

list():-

Purpose:-

This function is used to Assign variables as if they were an array

Syntax:void list (mixed $varname [, mixed $...])

- Like array(), this is not really a function, but a language construct.
- list() is used to assign a list of variables in one operation.
- list() only works on numerical arrays and assumes the numerical indices start at 0.

in_array():-

Purpose:-

This function is used to Checks if a value exists in an array or not.

Syntax: bool in_array (mixed $needle , array $haystack [, bool $strict])

next():-

Purpose:-

- This function is used to advance the internal array pointer of an array

Syntax: mixed next (array &$array)

- next() behaves like current(), with one difference. It advances the internal array pointer one place forward before returning the element value. That means it returns the next array value and advances the internal array pointer by one.

prev():-

- This function is used to Rewind the internal array pointer

  Syntax: mixed prev (array &$array)

- Rewind the internal array pointer.
- prev() behaves just like next(), except it rewinds the internal array pointer one place instead of advancing it.

end():-

Purpose:-

- This function is used to set the internal pointer of an array to its last element

  Syntax: mixed end (array &$array)

- end() advances array 's internal pointer to the last element, and returns its value.

Example:

```
<?php
    $fruits = array('apple', 'banana', 'cranberry');
    echo end($fruits); // cranberry
?>
```

each():-

Purpose:-

- This function is used to return the current key and value pair from an array and advance the array cursor.

  Syntax:array each (array &$array)

- Return the current key and value pair from an array and advance the array cursor.
- After each() has executed, the array cursor will be left on the next element of the array, or past the last element if it hits the end of the array. You have to use reset() if you want to traverse the array again using each.

Example:

```
<?php
    $foo = array("bob", "fred", "jussi", "jouni", "egon", "marliese");
    $bar = each($foo);
    print_r($bar);
?>
```

sort():-

Purpose:-

- This function is used to sort an array.

  Syntax:bool sort (array &$array [, int $sort_flags])

- This function sorts an array. Elements will be arranged from lowest to highest when this function has completed.

  Example:

```
<?php
        $fruits = array("lemon", "orange", "banana", "apple");
        sort($fruits);
        foreach ($fruits as $key => $val)
        {
                echo "fruits[" . $key . "] = " . $val . "\n";
        }
?>
```

rsort():-

Purpose:-

- This function is used to sort an array in reverse order.

  Syntax:bool rsort (array &$array [, int $sort_flags])

- This function sorts an array in reverse order (highest to lowest).
- This function assigns new keys to the elements in array . It will remove any existing keys that may have been assigned, rather than just reordering the keys

  Example:

```
<?php
        $fruits = array("lemon", "orange", "banana", "apple");
        rsort($fruits);
        foreach ($fruits as $key => $val)
        echo "$key = $val\n";
?>
```

asort():-

Purpose:-

- This function is used to sort an array and maintain the index association.

  Syntax: bool asort (array &$array [, int $sort_flags])

  This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant.

Example:

```
<?php
    $fruits = array("d" => "lemon", "a" => "orange", "b" => "banana", "c" =>
"apple");
    asort($fruits);
    foreach ($fruits as $key => $val)
    {
        echo "$key = $val\n";
    }
    ?>
```

The output of above code will be:-

```
c = apple
b = banana
d = lemon
a = orange
```

arsort():-

Purpose:-

- This function is used to Sort an array in reverse order and maintain index association.

  Syntax:bool arsort (array &$array [, int $sort_flags])

- This function sorts an array such that array indices maintain their correlation with the array elements they are associated with.
- This is used mainly when sorting associative arrays where the actual element order is significant.

  Example:

```
<?php
    $fruits = array("d" => "lemon", "a" => "orange", "b" => "banana",
"c" => "apple");
    arsort($fruits);
    foreach ($fruits as $key => $val)
    {
        echo "$key = $val\n";
    }
?>
```

The output of above code will be:-

```
a = orange
d = lemon
b = banana
c = apple
```

array_merge():-

Purpose:-

This function is used to merge one or more arrays.

Syntax: array array_merge (array $array1 [, array $array2 [, array $…]])

- Merges the elements of one or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.
- If the input arrays have the same string keys, then the later value for that key will overwrite the previous one. If, however, the arrays contain numeric keys, the later value will not overwrite the original value, but will be appended.
- If only one array is given and the array is numerically indexed, the keys get reindexed in a continuous way.

array_reverse():-

Purpose:-

- This function is used to return an array with elements in reverse order.

  Syntax: array array_reverse (array $array [, bool $preserve_keys])

- Takes an input array and returns a new array with the order of the elements reversed.

  Example:

```
<?php
    $input  = array("php", 4.0, array("green", "red"));
    $result = array_reverse($input);
    $result_keyed = array_reverse($input, true);
?>
```

➔ **MISCELLANEOUS FUNCTION:**

define():-

Purpose:-

- This function is used to define the named constant.

Syntax: bool define (string $name , mixed $value [, bool $case_insensitive= false])

Example:

```
<?php
    define("CONSTANT", "Hello world.");
    echo CONSTANT; // outputs "Hello world."
    echo Constant; // outputs "Constant" and issues a notice.
    define("GREETING", "Hello you.", true);
    echo GREETING; // outputs "Hello you."
    echo Greeting; // outputs "Hello you."
?>
```

constant():-

Purpose:-

- This function is used to return the value of a constant.
  Syntax:mixed constant (string $name)
- Return the value of the constant indicated by name .
- constant() is useful if you need to retrieve the value of a constant, but do not know its name. I.e. it is stored in a variable or returned by a function.
- This function works also with class constants.

Example:

```
<?php
        define("MAXSIZE", 100);
        echo MAXSIZE;
        echo constant("MAXSIZE"); // same thing as the previous line
        interface bar
        {
                const test = 'foobar!';
        }
        class foo
        {
                const test = 'foobar!';
        }
        $const = 'test';
        var_dump(constant('bar::'. $const)); // string(7) "foobar!"
        var_dump(constant('foo::'. $const)); // string(7) "foobar!"

?>
```

include():-

Purpose:-

- This function is used to include() statement includes and evaluates the specified file.

Example:

vars.php

```
<?php
        $color = 'green';
        $fruit = 'apple';
?>
```

test.php

```
<?php
        echo "A $color $fruit"; // A
        include 'vars.php';
        echo "A $color $fruit"; // A green apple
?>
```

require():-

Purpose:-

- require() is identical to include() except upon failure it will produce a fatal E_ERROR level error. In other words, it will halt the script whereas include() only emits a warning (E_WARNING) which allows the script to continue.

Example:

```
<?php
        require("teste.php");
        if (class_exists("motherclass"))
        echo "It exists";
?>
```

header():-

Purpose:-

- This function is used to send a raw HTTP header.

Syntax: void header (string $string [, bool $replace= true [, int $http_response_code]])

- header() is used to send a raw HTTP header. See the » HTTP/1.1 specification for more information on HTTP headers.
- Remember that header() must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP. It is a very common error to read code with include(), or require(), functions, or another file access function, and have spaces or empty lines that are output before header() is called. The same problem exists when using a single PHP/HTML file.

Example:

```
<?php
    header("Location: http://www.example.com/"); /* Redirect browser */

    /* Make sure that code below does not get executed when we redirect. */
    exit;
    ?>
```

die():-

Purpose:-

- This function is equivalent to exit.

➔ **FILE HANDLING FUNCTION:**

fopen():-

Purpose:-

- This function opens file or URL.

Syntax:fopen (string $filename , string $mode [, bool $use_include_path [, resource $context]])

- fopen() binds a named resource, specified by filename , to a stream.
- The parameters specifies filename, mode.
- The mode specifies the type of access you require to the stream. It may be any of the following:

mode	Description
'r'	Open for reading only; place the file pointer at the beginning of the file.
'r+'	Open for reading and writing; place the file pointer at the beginning of the file.
'w'	Open for writing only; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
'w+'	Open for reading and writing; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
'a'	Open for writing only; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
'a+'	Open for reading and writing; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
'x'	Create and open for writing only; place the file pointer at the beginning of the file. If the file already exists, the fopen() call will fail by returning FALSE and generating an error of level E_WARNING. If the file does not exist, attempt to create it. This is equivalent to specifying O_EXCL|O_CREAT flags for the underlying open(2) system call.
'x+'	Create and open for reading and writing; place the file pointer at the beginning of the file. If the file already exists, the fopen() call will fail by returning FALSE and generating an error of level E_WARNING. If the file does not exist, attempt to create it. This is equivalent to specifying O_EXCL|O_CREAT flags for the underlying open(2) system call.

Example:

```
<?php
        $handle = fopen("/home/rasmus/file.txt", "r");
        $handle = fopen("/home/rasmus/file.gif", "wb");
        $handle = fopen("http://www.example.com/", "r");
```

```
$handle=fopen("ftp://user:password@example.com/somefile.txt",
"w");
?>
```

fread():-

Purpose:-

- This function is used to read binary file

Syntax: string fread (resource $handle , int $length)

- fread() reads up to length bytes from the file pointer referenced by handle . Reading stops as soon as one of the following conditions is met:
- length bytes have been read
- EOF (end of file) is reached
- a packet becomes available (for network streams)
- 8192 bytes have been read (after opening userspace stream)

Example:

```
<?php
    // get contents of a file into a string
    $filename = "/usr/local/something.txt";
    $handle = fopen($filename, "r");
    $contents = fread($handle, filesize($filename));
    fclose($handle);
?>
```

fwrite():-

Purpose:-

- This function is used to write to binary file

Syntax: int fwrite (resource $handle , string $string [, int $length])

- fwrite() writes the contents of string to the file stream pointed to by handle .

Example:

```
<?php
    $fp = fopen('data.txt', 'w');
    fwrite($fp, '1');
    fwrite($fp, '23');
    fclose($fp);
    // the content of 'data.txt' is now 123 and not 23!
?>
```

fclose():-

Purpose:-

- This function is used to close the file

Syntax: bool fclose (resource $handle)

Example:

```
<?php
    $handle = fopen('somefile.txt', 'r');
    fclose($handle);
?>
```

file_exists():-

Purpose:-

- This function is used to check whether a file or directory exists

Syntax: bool file_exists (string $filename)

- Returns TRUE if the file or directory specified by filename exists; FALSE otherwise.
- This function will return FALSE for symlinks pointing to non-existing files.

Example:

```
<?php
    $filename = '/path/to/foo.txt';
    if (file_exists($filename)) {
    echo "The file $filename exists";
    } else {
     echo "The file $filename does not exist";
    }
?>
```

is_readable():-

Purpose:-

- This function is used to tell whether the filename is readable or not.

Syntax: bool is_readable (string $filename)

Example:

```
<?php
    $filename = 'test.txt';
    if (is_readable($filename))
    {
        echo 'The file is readable';
    } else
    {
         echo 'The file is not readable';
    }
?>
```

is_writable):-

Purpose:-

- This function is used to tell whether the filename is writable or not.

Syntax: bool is_writable (string $filename)

Example:

```
<?php
        $filename = 'test.txt';
        if (is_writable($filename))
        {
                echo 'The file is writable';
        } else
        {
                 echo 'The file is not writable';
        }
?>
```

fgets():-

Purpose:-

- This function is used to get line from file pointer.

Syntax: string fgets (resource $handle [, int $length])

- Returns a string of up to length - 1 bytes read from the file pointed to by handle .
- If an error occurs, returns FALSE.

Example:

```
<?php
        $handle = fopen("/tmp/inputfile.txt", "r");
        if ($handle){
                while (!feof($handle)) {
                    $buffer = fgets($handle, 4096);
                    echo $buffer;
                }
          fclose($handle);
        }
?>
```

fgetc():-

Purpose:-

- This function is used to get character from file pointer.

Syntax: string fgetc (resource $handle)

- Returns a string containing a single character read from the file pointed to by handle. Returns FALSE on EOF.

Example:

```
<?php
 $fp = fopen('somefile.txt', 'r');
 if (!$fp)
  {
    echo 'Could not open file somefile.txt';
 }
 while (false !== ($char = fgetc($fp)))
  {
 echo "$char\n";
 }
 ?>
```

file():-

Purpose:-

- This function is used to read entire file into an array.
  Syntax: array file (string $filename [, int $flags [, resource $context]])

- Reads an entire file into an array.
- You can use file_get_contents() to return the contents of a file as a string.

Example:

```
<?php
// Get a file into an array.  In this example we'll go through HTTP to get
// the HTML source of a URL.
$lines = file('http://www.example.com/');
// Loop through our array, show HTML source as HTML source; and line numbers too.
foreach ($lines as $line_num => $line) {
   echo "Line #<b>{$line_num}</b> : " . htmlspecialchars($line) . "<br />\n";
}
// Another example, let's get a web page into a string.  See also file_get_contents().
$html = implode('', file('http://www.example.com/'));
// Using the optional flags parameter since PHP 5
$trimmed = file('somefile.txt', FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
?>
```

file_get_contents():-

Purpose:-

- This function is used to read entire file into string..

Syntax:

string file_get_contents (string $filename [, int $flags [, resource $context [, int $offset [, int $maxlen]]]]
)

- This function is similar to file(), except that file_get_contents() returns the file in a string, starting at the specified offset up to maxlen bytes. On failure, file_get_contents() will return FALSE.
- file_get_contents() is the preferred way to read the contents of a file into a string. It will use memory mapping techniques if supported by your OS to enhance performance.

file_put_contents():-

Purpose:-

- This function is used to write string to file.

Syntax: int file_put_contents (string $filename , mixed $data [, int $flags [, resource $context]])

- This function is identical to calling fopen(), fwrite() and fclose() successively to write data to a file.
- If filename does not exist, the file is created. Otherwise, the existing file is overwritten, unless the FILE_APPEND flags is set.

ftell():-

Purpose:-

- This function is used to Returns the current position of the file read/write pointer.

Syntax: int ftell (resource $handle)

- Returns the position of the file pointer referenced by handle .

Example:

```
<?php
    // opens a file and read some data
    $fp = fopen("/etc/passwd", "r");
    $data = fgets($fp, 12);
    // where are we ?
    echo ftell($fp); // 11
    fclose($fp);
    ?>
```

fseek():-

Purpose:-

- This function seeks on a file pointer.

Syntax: int fseek (resource $handle , int $offset [, int $whence])

- Sets the file position indicator for the file referenced by handle . The new position, measured in bytes from the beginning of the file, is obtained by adding offset to the position specified by whence .
- whence values are:

  1) SEEK_SET - Set position equal to offset bytes.
  2) SEEK_CUR – Set position to current location plus offset.
  3) SEEK_END – Set position to end-of-file plus offset.

  If whence is not specified, it is assumed to be SEEK_SET.

  Example:

```
<?php
        $fp = fopen('somefile.txt', 'r');
        // read some data
        $data = fgets($fp, 4096);
        // move back to the beginning of the file
        // same as rewind($fp);
        fseek($fp, 0);
?>
```

rewind():-

Purpose:- This function is used to rewind the position of file pointer.

Syntax: bool rewind (resource $handle)

- Sets the file position indicator for handle to the beginning of the file stream.
- If you have opened the file in append ("a" or "a+") mode, any data you write to the file will always be appended, regardless of the file position.

Example:

```
<?php
        $handle = fopen('output.txt', 'r+');
        fwrite($handle, 'Really long sentence.');
        rewind($handle);
        fwrite($handle, 'Foo');
        rewind($handle);
        echo fread($handle, filesize('output.txt'));
        fclose($handle);
        ?>
```

The output of above code will be: Foolly long sentence.

copy():-

Purpose:-

- This function is used to copy the file.

  Syntax: bool copy (string $source , string $dest [, resource $context])

- Makes a copy of the file source to dest .
- If you wish to move a file, use the rename() function.

Example:

```
<?php
    $file = 'example.txt';
    $newfile = 'example.txt.bak';
    if (!copy($file, $newfile)) {
      echo "failed to copy $file...\n";
    }
?>
```

unlink():-

Purpose:-This function is used to delete the file.

Syntax: bool unlink (string $filename [, resource $context])

Example:

```
<?php
    rename("/tmp/tmp_file.txt", "/home/user/login/docs/my_file.txt");
?>
```

Example:

```
<?php
    $fh = fopen('test.html', 'a');
    fwrite($fh, '<h1>Hello world!</h1>');
    fclose($fh);
    mkdir('testdir', 0777);
    unlink('test.html');
    unlink('testdir');
?>
```

rename():-

Purpose:-

- This function renames the file or directory.

  Syntax: bool rename (string $oldname , string $newname [, resource $context])

  Example:

```php
<?php
        rename("/tmp/tmp_file.txt", "/home/user/login/docs/my_file.txt");
?>
```

Unit 3 - Handling Form, Session Tracking & PHP Components

➔ Handling form with GET & POST

- When a form is submitted to a PHP script, the information from that form is automatically made available to the script.
- There are many ways to access this information here explain GET and POST method of form object.

GET Method

- The GET method passes arguments from in page to the next page as a part of the URL query string.
- When used for form handling, GET appends in the indicated variable name and value to the URL designed in the ACTION attribute with a question mark separator.

Each item submitted via GET method is accessed in the handler via the $_GET array.

(1) $HTTP_GET_VARS[form_element_obj_name]
(2) $_GET[form_element_obj_name]

The page and the encoded information are separated by the **?** character.
Example : http://www.test.com/index.htm?name1=value1&name2=value2

Example of GET Method :

```
<html>
  <body>
      <form action = "" method = "GET">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" name="submit" />
    </form>
<?php
          if(isset($_GET['submit'])) {
          if($_GET["name"] || $_GET["age"]) {
    echo "Welcome ". $_GET['name']. "<br />";
    echo "You are ". $_GET['age']. " years old.";
    exit();
  } }
?>
    </body>
</html>
```

POST method

- POST method is the preferred method of form submission.
- The form data set is included in the body of the form when it is forwarded to the processing agent (web server).
- No visible change to the URL will reset according to the different
  data submitted.
  Each item submitted via POST Method is accessed in the handler via the $_POST array.
  (1)$HTTP_POST_VARS().
  (2)$_POST().
- It is more secure then GET because user entered information is never visible in the URL.

Example of POST Method :

```
<html>
  <body>
      <form action = "" method = "POST">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" name="submit" />
    </form>
<?php
         if(isset($_POST['submit'])) {
         if($_POST["name"] || $_POST["age"]) {
    echo "Welcome ". $_POST['name']. "<br />";
    echo "You are ". $_POST['age']. " years old.";
    exit();
  } }
?>
   </body>
</html>
```

➔ **Create Cookies With PHP**

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Syntax

setcookie(name, value, expire, path);

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie.

Example :

```
<?php
$cookie_name="user";
$cookie_value="HN SHUKLA";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");   ?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
        echo "Cookie named '" . $cookie_name . "' is not set!";
}
Else {
        echo "Cookie '" . $cookie_name . "' is set!<br>";
        echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

Note: The setcookie() function must appear BEFORE the <html> tag.

Delete a Cookie

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>
<?php
echo "Cookie 'user' is deleted.";
?>
</body>
</html>
```

➔ PHP SESSION

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.
A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.
A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the **session_start()** function. This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page.
Session variables are stored in associative array called **$_SESSION[]**. These variables can be accessed during lifetime of a session.
Make use of **isset()** function to check if session variable is already set or not.

Example :

```
<?php
session_start();
if( isset( $_SESSION['counter'] ) ) {
$_SESSION['counter'] += 1; }
else { $_SESSION['counter'] = 1; }
$msg = "You have visited this page ". $_SESSION['counter'];
$msg .= "in this session.";
?>
<html>
<head>

<title>Setting up a PHP session</title>
</head>
<body>
<?php
echo ( $msg );
?>
</body>
</html>
```

Destroying a PHP Session

A PHP session can be destroyed by **session_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use **unset()** function to unset a session variable.

Example1 :

```
<?php
unset($_SESSION['counter']);
?>
```

Example2 :

```
<?php
session_destroy();
?>
```

➔ PHP Server Variables

PHP provides various predefined variables. PHP provides set of predefined arrays containing variables from the web server the environment, and user input. Such new arrays are called as superglobal variables.

1. $GLOBALS

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

Example

```
<?php
$x = 75;
$y = 25;
function addition() {
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
```

```
}
addition();
echo $z;
?>
```

2. $_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

Example

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

3. $_GET

PHP $_GET can also be used to collect form data after submitting an HTML form with method="get". $_GET can also collect data sent in the URL.

Example

```
<html>
<body>
<?php
echo "Study " . $_GET['subject'];
?>
</body>
</html>
```

4. $_POST

PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

Example

```
<html>
<body>
<?php
echo "Study " . $_POST['subject'];
?>
</body>
</html>
```

5. $_FILES

$_FILES is a super global variable which can be used to upload files. Here we will see an example in which our php script checks if the form to upload the file is being submitted and generates an message if true.

Example

```
<html>
<body>
<form action="" method="post" enctype="multipart/form-data">
Filename: <input type="file" name="file"><br>
<input type="submit" name="submit" value="submit">
</form>
</body>
</html>
<?php
if($_FILES['file']>0)
{
        echo "You have selected a file to upload";
}
?>
```

6. $_ REQUEST

$_REQUEST is a super global variable which is widely used to collect data after submitting html forms.

Example

```
<html>
<body>
<form name="contact" method="post" action="">
<input size=25 name="name">
<input  type=submit name="send" value="Submit">
</form>
<?php
$name=$_REQUEST['name'];
echo $name;
?>
```

7. $_ COOKIE

Cookies are small text files loaded from a server to a client computer storing some information regarding the client computer, so that when the same page from the server is visited by the user, necessary informations can be collected from the cookie itself, decreasing the latency to open the page.

Example

```
<?php
setrawcookie();
print_r($_COOKIE);
?>
```

8. $_ SESSION

Sessions are wonderful ways to pass variables. All you need to do is start a session by session_start();Then all the variables you store within a $_SESSION, you can access it form anywhere in the server.

Example

```
<?php
session_start();
```

```
$_SESSION[‘hns']='The IT & Management College';
echo $_SESSION[‘hns'];
?>
```

9. $_ PHP_SELF

PHP_SELF is a variable that returns the current script being executed. This variable returns the name and path of the current file (from the root folder). You can use this variable in the action field of the FORM.

Example

```
<form name="form1" method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>" >
```

➔ **PHP Components**

PHP GD Library

PHP provides a feature to draw any images using various kinds of shapes. The area where the shapes are to be drawn is known as canvas.

There are different versions of Graphical Display in PHP.

There are 5 steps for the graphics image to be used in PHP.

1. Create Canvas
2. Allocate background color for shapes.
3. Allocate foreground color for shapes.

4. Draw shapes on the canvas.
5. Place the canvas on the web-page.
6. You should put at the top : header('Content-type: image/png');

Canvas Function

1. Imagecreate()

It is the first step to draw the image. This function is used for creating the canvas. To create the canvas fixed width and height is to be given.

Example

```
<?php
$canvas=imagecreate(500,500);
?>
```

This will create a canvas of width and height 500.

Color Allocation Function

2. **ImagecolorAllocate()**

It is the next step for drawing the image on the canvas. This function is used for allocating the color to background of canvas and the shapes which are drawn. The colors which are to be used are in RGB (Red, Green, Blue) format.

Example

```
<?php
$canvas=imagecreate(500,500);
$backcolor=imagecolorallocate($canvas,0,0,0);
?>
```

This will create a canvas of width and height 500. The background color will be black.

Place image on web-page Function

3. **imagepng() or imagejpeg() or imagegif()**

The last step is to place this canvas on the web-page. This image can be in any format that is either PNG, JPEG or GIF.
If PNG image is to be created, then the function imagepng is used.
If JPEG image is to be created, then the function imagejpeg is used.
If GIF image is to be created, then the function imagegif is used.

Example

```
<?php
header('Content-type: image/png');
$canvas=imagecreate(500,500);
$backcolor=imagecolorallocate($canvas,0,0,0);
imagepng($canvas);
?>
```

Shapes Function

4. **imageline()**

This function is used to draw line on the canvas. It has 6 parameters to be taken as input.

Example

```
<?php
header('Content-type: image/png');
$canvas=imagecreate(500,500);
imagecolorallocate($canvas,15, 142, 210);
$linecolor=imagecolorallocate($canvas,1,1,1);
imageline($canvas,200,200,250,200,$linecolor);
imagepng($canvas);   ?>
```

It will draw a straight line starting from (200,200) till (250,200).

5. **imagerectangle() or imagefilledrectangle()**

Imagerectangle() function is used to draw simple rectangle.
Imagefilledrectangle() function is used to draw a rectangle with the color filled in it. Both the functions take 6 parameters as input.

Example

```
<?php
header('Content-type: image/png');
$canvas=imagecreate(500,500);
imagecolorallocate($canvas,15, 142, 210);
$linecolor=imagecolorallocate($canvas,1,1,1);
```

```
imagefilledrectangle($canvas,200,200,250,250,$linecolor);
imagepng($canvas);
?>
```

6. **Imageellipse() or imagefilledellipse()**

Imageellipse() function is used to draw simple ellipse.
Imagefilledellipse() function is used to draw a ellipse with the color filled in it. Both the functions take 6 parameters as input.

Example

```
<?php
header('Content-type: image/png');
$canvas=imagecreate(500,500);
imagecolorallocate($canvas,15, 142, 210);
$linecolor=imagecolorallocate($canvas,1,1,1);
imagefilledellipse($canvas,250,250,100,100,$linecolor);
imagepng($canvas);
?>
```

7. **Imagepolygon() or imagefilledpolygon()**

Polygon means a shape having starting and ending point same.
imagepolygon() function is used to draw simple polygon.
Imagefilledpolygon() function is used to draw a polygon with the color filled in it.
Both function takes 4 parameters as input.

Example

```
<?php
header('Content-type: image/png');
$canvas=imagecreate(500,500);
imagecolorallocate($canvas,15, 142, 210);
$linecolor=imagecolorallocate($canvas,1,1,1);
$array=array(100,100,150,150,50,150,100,100);
imagefilledpolygon($canvas,$array,3,$color);
imagepng($canvas);
?>
```

8. **imagearc()**

This function is used to draw an arc. It has 8 parameters into it.

Example

```
<?php
header('Content-type: image/png');
$canvas=imagecreate(500,500);
imagecolorallocate($canvas,15, 142, 210);
$linecolor=imagecolorallocate($canvas,1,1,1);
imagearc($canvas, 200,200,40,20,0,270,$color);
imagepng($canvas);
?>
```

Other Function

9. **imagesX(), imagesY()**

imagesX() function will return the width of the image in integer type.
imagesY() function will return the height of the image in integer type.

Example

```
<?php
$canvas=imagecreate(500,500);
echo imagesX($canvas);
echo "<br>";
echo imagesY($canvas);
?>
```

10. **Imagecreatefromjpeg() or imagecreatefrompng()**

These function are useful when working with images that are loaded using a funcion.

Example

```
<?php
header('Content-type: image/jpeg');
$load=imagecreatefromjpeg('IMG_20160904_173026.jpg');
imagejpeg($load);
?>
```

11. **imagecopy()**

This function is used to copy the image from source file into the destination file. It has 8 parameters to be given.

Example

```
<?php
header('Content-type: image/jpeg');
$source=imagecreatefromjpeg('IMG_20160904_173026.jpg');
$dest=imagecreatefromjpeg('IMG_20160824_123645.jpg');
imagecopy($dest,$source,0,0,0,0,1080,1080);
imagejpeg($dest);
?>
```

12. **imagecopymerge()**

This function is similar to that of the imagecopy() but here the other parameter that is the percentage from 1 to 100% is to be given.

Example

```
<?php
header('Content-type: image/jpeg');
$source=imagecreatefromjpeg('IMG_20160904_173026.jpg');
```

```
$dest=imagecreatefromjpeg('IMG_20160824_123645.jpg');
imagecopymerge($dest,$source,0,0,0,0,1080,1080,50);
imagejpeg($dest);
?>
```

13. **imagecreatetruecolor()**

This function is similar to that of the imagecreate() but the difference is that using this function it will create a canvas with true color range.

Example

```
<?php
header('Content-type: image/png');
$canvas=imagecreatetruecolor(500,500);
imagepng($canvas);
?>
```

➔ Regular Expressions

Regular expression means the formula or the general format of any string, or expression.
It used for pattern matching.
It is also known as regexis. It checks any characters or digits which are specified in the format. It is used generally for the validation on Server side scripting.

Advantages :

It prevents the data to be entered in invalid format.
It is a portable language as it has general format.
It is easy to learn and use.

Disadvantages :

The validation which is provided using server side scripting is disadvantageous, as the web-page is refreshed again and again.

Type of regular expression :

1. POSIX Extended Regular Expression
   (Portable Operating System Interface for unix)
2. Perl Compatible Regular Expression

1. POSIX Extended Regular Expression

POSIX or "Portable Operating System Interface for unix" is a collection of standards that define some of the functionality that a (UNIX) operating system should support. When this type of regular expression is used then the function ereg() is require to be used for the pattern matching. It is the oldest type of regular expression in use.

Symbol	Description
^	for starting the format string
$	for ending the format string
[a-z]	only lower case alphabets
[A-Z]	only uper case alphabets
[a-z A-Z]	only alphabets in any form

[0-9]	only digits
[a-z A-Z 0-9]	alphanumeric characters
\	used when any special characters are to be used in string
()	used when more than one value is to be checked
|	for or condition
{n}	is used when the lengths of the digits are known. Where n is an integer value.
{n,m}	n for minimum value and m for maximum value.

When the format which is matched is not found, then it returns as false otherwise it returns true value.

Example :

```
//Validation for date format dd/mm/yyyy.
<?php
$date="19/09/2016";
if(ereg('^[0-9]{2}\/[0-9]{2}\/[0-9]{4}$',$date))
echo "It is valid date";
else
echo "Invalid date";
?>
```

2. **Perl Regular Expression**

This type of regular expression is developed using the Perl language. The function which is used for the given regular expression is preg_match() that is perl regular expression.

Symbol	**Description**
\d	Any number
\D	for Anything other than a number
\s	Any kind of whitespace
\S	Anything other than whitespace
\W	Any word character (including the underscore character)
\w	Anything other than a word character
\A	Beginning of string
\b	Word boundry
\Z	End of string

Example :

```
//Validation for date format dd/mm/yyyy.
<?php
$date="19/09/2016";
if(preg_match('^\d{2}\/\d{2}\/\d{4}$',$date))
echo "It is valid date";
else
echo "Invalid date";
?>
```

General Format using ereg function

Email Address

^[0-9 a-z A-Z]+(.|_)?[0-9 a-z A-Z]*\@(yahoo|gmail|co|in|rediff)\.(com|in)$

Pin code

^[0-9]{5,6}$

Mobile Number
^\+91[0-9]{10}$
General Format using preg_match function

Email Address
^\S+(.|_)?\S*\@(yahoo|gmail|co|in|rediff)\.(com|in)$
Pin code
^\d{5,6}$
Mobile Number
^\+91\d{10}$

➔ **File Uploading**

Sometimes it is required that files are to be uploaded on the server. For this first step is to create HTML Form as here :
HTML FORM

```
<html>
<body>
<form action="upload_file.php" method="post" enctype="multipart/form-data">
File Name : <input type="file" name="file" id="file"><br>
<input type="submit" name="submit" value="Submit">
</form>
</body>
</html>
```

While uploading file, some points should be remembered :

- The enctype attribute of the <form> tag specifies which content-type to use when submitting the form. "multipart/form-data" is used when a form requires binary data, like the contents of a file, to be uploaded.

- The type="file" attribute of the <input> tag specifies that the input should be processed as a file. For example, when viewed in a browser, there will be a browse-button next to the input field.

Next step is to write a PHP code for uploading the file on submitting the form.

PHP Script

```
<?php
If($_FILES['file']['error']>0)
{
        echo "Error: ".$_FILES['file']['error']."<br>";
}
Else
{
        echo "Upload: ".$_FILES['file']['name']."<br>";
        echo "Type: ".$_FILES['file']['type']."<br>";
        echo "Size: ".$_FILES['file']['size']."<br>";
        echo "Stored: ".$_FILES['file']['tmp_name']."<br>";
```

} ?>
By using the global PHP $_FILES array you can upload files from a client computer to the remote server. The first parameter is the form's input name and the second index can be either "name", "type", "size", "tmp_name" or "error".

$_FILES['file']['name'] – the name of the uploaded file
$_FILES['file']['type'] – the type of the uploaded file
$_FILES['file']['size'] – the size in kb of the uploaded file
$_FILES['file']['tmp_name'] – the name of the temporary copy of the file stored on the server
$_FILES['file']['error'] – the error code resulting from the file upload

File Upload with proper extension

```
<?php
$allowedExts=array("gif", "jpeg", "jpg", "png");
$extension=end(explode(".", $_FILES['file']['name']));
If((($_FILES['file']['type']=="image/gif")
|| ($_FILES['file']['type']=="image/jpeg")
|| ($_FILES['file']['type']=="image/jpg")
|| ($_FILES['file']['type']=="image/png"))
&& ($_FILES['file']['size']< 20000)
&& in_array($extension, $allowedExts)
{
        if ($_FILES['file']['error'] > 0) {
                echo "Return Code: ". $_FILES['file']['error']."<br>";
        }
else {
        if (file_exists("images/".$_FILES['file']['name'])) {
                echo $_FILES['file']['name'] . "already exists.";
        }
        else {
        move_uploaded_file($_FILES['file']['tmp_name'], "images/" . $_FILES['file']['name']);
        echo "Uploaded";
        }
        }
} else { echo "Invalid File"; }  ?>
```

➔ **Sending mail using mail()**

PHP makes use of mail() function to send an email. This function requires three mandatory parameters that specify the recipient's email address, the subject of the message and the actual message additionally there are other two optional parameters.

Sending simple text email :

```
<?php
$to="email";
$subject="This is subject";
$message="This is simple text message";
$header="From:email \r\n";
$retval=mail($to,$subject,$message, $header);
if($retval==true) {
echo "Message sent successfully"; }
```

```
else {
echo "Message could not be sent"; }    ?>
```

This is how we send simple text email. To send HTML email code is as below…

➔ **Sending HTML email :**

```
<?php
$to="email";
$subject="This is subject";
$message="<b>This is HTML message<b>";
$message="<h1>This is HTML message<h1>";
$header="From:email \r\n";
$header="Cc:email2 \r\n";
$header="MIME-Version: 1.0 \r\n";
$header="Content-type: text/html\r\n";
$retval=mail($to,$subject,$message, $header);
if($retval==true) {
echo "Message sent successfully"; }
else {
echo "Message could not be sent"; }
?>
```

Go to the link below to get the class file for configuring smtp
https://github.com/Synchro/PHPMailer
Now from this 3 main class files are required to download

1. class.phpmailer.php
2. Class.pop3.php
3. Class.smtp.php

Place all this file in same folder and in that folder create a new php file with code below to send mail using smtp().

```
<?php
require_once('class.phpmailer.php');
$mail=new PHPMailer();
$body=file_get_contents('contents.html');
$body=eregi_replace("[\]",$body);
$mail->IsSMTP();
$mail->Host="mail.yourdomain.com";
$mail->SMTPDebug=2;
//1 = errors and messages
//2= message only
```

➔ **Sending mail using smtp()**

```
$mail->SMTPAuth=true;
$mail->Host="mail.yourdomain.com";
$mail->Port=26;  //for gmail server
$mail->Username="yourname@yourdomain";  //SMTP account username
```

```
$mail->Password="your password";
$mail->SetFrom('name@yourdomain.com','First Last');
$mail->AddReplyTo("name@yourdomain.com","First Last");
$mail->Subject="PHPMailer Test Subject via smtp, basic with authentication";
$mail->AltBody="To view message, use an HTML compatible email viewer.";
$mail->MsgHTML($body);
$address="whoto@otherdomain.com";
$mail->AddAddress($address,"Your Name");
$mail->AddAttachment("images/phpmailer.gif");
if(!$mail->Send()){
echo "Mailer Error:".$mail->ErrorInfo;
} else {
echo "Message sent";
}
?>
```

AJAX

➔ **Introduction to AJAX**

AJAX is a new programming style by which we can receive a quicker response to the request sent for server side files or resources. Requests are sending through the Javascript object to a server file. After execution, we can get updates without page refresh.

AJAX can be expanded as Asynchronous Javascript and XML using which we can get any type of data from the server. It has similar coding styles and rules as like as Javascript, but with advanced feature.

AJAX Life Cycle
The following steps are to be considered as given below :

1. User make initial request against a given URL.
2. Server return the original HTML page.
3. Browser renders page as in memory DOM tree.
4. User activity causes subsequent request against another URL asynchronously, leaving existing DOM tree untouched.
5. Browser returns data to a callback function inside the existing page.
6. Browser parses result and updates in memory DOM with the new data, which is then reflected on screen to the user.
   AJAX is based on the following web elements :
   - JavaScript
   - XML
   - HTML
   - CSS

   The web standards used in AJAX are well defined, and supported by all major browsers.
   AJAX applications are browser and platform independent.

➔ **PHP With AJAX**

How to access PHP file using AJAX programming

1. First, we need to create instance for XMLHTTPRequest using XMLHttpRequest() construct. For older browsers like IE5, IE6, we should use ActiveXObject().
2. And then, Open the server file with required arguments.
3. Send request to the file using XMLHttpRequest instance.
4. Work with the response text generated by server file.

Create instance for XMLHTTPRequest

XMLHttpRequest instance can be created by some Javascript constructs like XMLHttpRequest or ActiveXObject.

For example, the following line of code is used to create such instance.

```
xmlHttpRequest=new XMLHttpRequest();
```

Opening PHP file and sending request

Using XMLHttpRequest instance created in the last step, we should open the file which is at the server. The following code deals with it by passing URL and requesting method as arguments.

```
xmlHttpRequest.open("GET","signin.php?userName="+userName+"&password="+password,true);
xmlHttpRequest.send();
```

Example: User authentication using PHP, AJAX, and MySQL

we are going to do the same thing with AJAX programming to get comparatively quicker response server. The HTML content with AJAX call is as shown below.

login.html

```
<html>
<head>
<title>User Login</title>
<link rel="stylesheet" type="text/css" href="styles.css" />
<script>
function callLogin()
{
var userName = document.getElementById("userName").value;
var password = document.getElementById("password").value;
xmlHttpRequest=new XMLHttpRequest();
xmlHttpRequest.onreadystatechange=function() {
if (xmlHttpRequest.readyState==4 && xmlHttpRequest.status==200) {
document.getElementById("messageBoxId").innerHTML=xmlHttpRequest.responseText;
}
}
xmlHttpRequest.open("GET","signin.php?userName="+userName+"&password="+password,true);
xmlHttpRequest.send();

}
```

```
</script>
</head>
<body>
<form name="frmUser" method="post" action="">
<div id="loginDivId">
<div class="message" id="messageBoxId"></div>
<table border="0" cellpadding="10" cellspacing="1" width="500" align="center">
<tr class="tableheader">
<td align="center" colspan="2">Enter Login Details</td>
</tr>
<tr class="tablerow">
<td align="right">Username</td>
<td><input type="text" name="userName" id="userName"></td>
</tr>
<tr class="tablerow">
<td align="right">Password</td>
<td><input type="password" name="password" id="password"></td>
</tr>
<tr class="tableheader">
<td align="center" colspan="2"><input type="button" name="submit" value="Login"
onClick="callLogin();"></td>
</tr>
</table>
</div>
</form>
</body>
</html>
```

Here, the PHP script is completely separated and called by AJAX on clicking the Login button. Now the PHP script is,

signin.php

```
<?php
if(count($_GET)>0) {
$conn = mysql_connect("localhost","root","");
mysql_select_db("login",$conn);
$result = mysql_query("SELECT * FROM users WHERE userName='" . $_GET["userName"] . "' and password
= '". $_GET["password"]."'");
$count  = mysql_num_rows($result);
if($count==0) {
print "Invalid Username or Password!";
} else {
print "You are successfully authenticated!";
}
mysql_close($conn);
}
?>
```

Working with AJAX as background process

With AJAX, your JavaScript communicates directly with the server, through the JavaScript XMLHttpRequest object. With an HTTP request, a web page can make a request to, and get a response from web server – without reloading the page. The user will stay on the same page, and he or she will not notice that scripts request pages, or send data to a server in the background.

By using the XMLHttpRequest object, a web developer can update a page with data from the server after the page has loaded.

➔ jQuery

jQuery is the lightweight version of JavaScript. Jquery is a fast, small, and rich Java Script library. The purpose of jQuery is to make it much easier to use JavaScript on your website.

Its features are working for things listed below with ready and easy to use API supported for all browsers.

How jQuery works and created

One of the most important aspects of JavaScript and thereby jQuery, is manipulation of the DOM. DOM stands for Document Object Model and is a mechanism for representing and interacting with your HTML, XHTML or XML documents.

jQuery provides a special utility function to select elements. It is called $. jQuery use selectors. Some of the selectors listed here :

$(document); //Active jQuery for object

$('#maintable') // Element with ID 'maintable'

$('p.first') //P tags with class first.

$('p[title="Hello"]') //P tags with title "Hello"

jQuery Events

Mouse Events : - click, dblclick, mousecenter, mouseleave

Keyboard Events : - keypress, keydown, keyup

Form Events :- submit, change, focus, blur

Document/Window Events :- load, resize, scroll, unload

jQuery Syntax for Event Methods :

In jQuery, most DOM events have an equivalent jQuery method. To assign a click event to all paragraphs on a page, you can do it: $("p").click();

The next step is to define what should happen when the event fires. You must pass function to the event : $("p").click(function() { //action goes here });

$(document).ready()

jQuery provides a special utility on the document object, called "ready", allowing to execute code only after the DOM has completely finished loading.

Using $(document).ready(), we can queue up a series of events and have them execute after the DOM is initialized.

Example to show message box on click

```
<html>
<head>
<script src="http://code.jquery.com/jquery-latest.js"> </script>
<script>
$(document).ready(function() {
$("p").click(function() {
```

```
alert("The paragraph was clicked");
});
});
</script>
</head>
<body>
<p>Click on this paragraph.</p>
</body>
</html>
```

Using jQuery with PHP

Step-1 : Create a result.php file.

```
<html>
<head>
<script src="http://code.jquery.com/jquery-latest.js"> </script>
<script>
function getdetails() {
var name=$('#name').val();
var rno=$('#rno').val();
$.ajax({
type: "POST",
url:"details.php",
data: {fname:name,id:rno}
}).done(function(result) {
$("#msg").html("Roll no " + rno +" has "+result);
});
}
</script>
</head>
<body>
<table>
<tr><td>Your Name : </td>
<td><input type="text" name="name" id="name"/></td></tr>
<tr><td>Roll Number : </td>
<td><input type="text" name="rno" id="rno"/></td></tr>
<tr><td></td>
<td><input type="button" name="submit" id="submit" value="Submit" onClick="getdetails()"/> </td> </tr>
</table>
<div id="msg"></div>
</body>
</html>
```

Step-2 : Create details.php

```
<?php
$name=$_POST['fname'];
$rno=$_POST['id'];
$con=mysql_connect("localhost","root","");
$db=mysql_select_db("databasename",$con);
$sql="SELECT result from student where name='".$name."' AND rno='".$rno."'";
$result=mysql_query($sql,$con);
```

```
$row=mysql_fetch_array($result);
echo $row['result'];
?>
```

Unit 4 - Introduction of SQL

➔ Working with Mysqli using PhpMyAdmin

PhpMyAdmin
PhpMyAdmin is one of the most popular applications for Mysqli databases management. It is a free tool written in PHP. Through this software you can create, alter, drop, delete, import and export Mysqli database tables. You can run Mysqli queries, optimize, repair and check tables, change collation and execute other database management commands.
The main PhpMyAdmin features are as follows:

- User-friendly web interface.
- Support for most Mysqli functions like browse, drop, create, copy and alter databases, tables, views, fields and indexes, execute Mysqli queries, manage stored procedures and functions.
- Import data from CSV and SQL files.
- Export data to various formats: CSV, SQL, XML, PDF, ISO/IEC 26300 - OpenDocument Text and Spreadsheet, Word, Excel, LATEX and others.
- Searching globally in a database.

➔ PHP Mysqli Connectivity

The first thing to do is connect to the database.The function to connect to Mysqli is called mysqli_connect. This function returns a resource which is a pointer to the database connection. It's also called a database handle.

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqlii_connect($hostname, $username, $password,$dbname);
?>
```

You should see "Connected to Mysqli" when you run this script. If you can't connect to the server, make sure your password, username and hostname are correct.
Once you've connected, you're going to want to select a database to work with. Let's assume the database is called 'examples'. To start working in this database, you'll need the mysqli_select_db() function:

➔ Mysqli Functions

1. mysqlii_connect()

PHP provides **mysqlii_connect()** function to open a database connection.
Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqlii_connect($hostname, $username, $password,$dbname);
?>
```

2. mysqlii_close()

You can disconnect from Mysqli database anytime using another PHP function **mysqli_close()**.

Syntax

mysqli_close (connection);

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqlii_connect($hostname, $username, $password,$dbname);
mysqlii_close($con);
?>
```

3. mysqlii_error(), mysqlii_errno()

The mysqli_error() function is used to get the error message from the last Mysqli operation.
The mysqli_errno() function is used to get the error code (numerical value) of the error message from the last Mysqli operation.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$con = mysqlii_connect($hostname, $username, $password,$dbname);
echo mysqli_errno($con) . ": " . mysqli_error($con). "<br />";
?>
```

4. mysqlii_query()

The mysqlii_query() is used to execute query on the default database.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqlii_connect($hostname, $username, $password,$dbname);
$result = mysqlii_query("insert into user values('user1','test')");
?>
```

5. mysqlii_fetch_array()

The mysqli_fetch_array() is used to retrieve a row of data as an array from a Mysqli result handle.

Syntax

mysqli_fetch_array(result);

Example

```
<?php
$username = "root";
$password = "";

$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqlii_connect($hostname, $username, $password,$dbname);
$result = mysqlii_query(“select * from examples");
while ($row = mysqlii_fetch_array($result)) {
echo “Name : “ . $row['name‘] . “Mobile“ . $row[‘city’];
}
?>
```

6. mysqlii_num_rows()

The mysqli_num_rows() is used to get the number of rows in a Mysqli result handle.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqlii_connect($hostname, $username, $password,$dbname);
$result = mysqlii_query("select * from examples");
$number_of_rows = mysqlii_num_rows($result);
echo "Number of rows fetched are : ". $number_of_rows;
?>
```

7. mysqli_affected_rows()

The mysqli_affected_rows() function is used to get the number of affected rows by the last Mysqli query. If you run a mysqli query to insert, update, replace or delete records, and want to know how many records are being affected by that query, you have to use mysqli_affected_rows().

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqlii_connect($hostname, $username, $password,$dbname);
$result = mysqlii_query("insert into user values('user1','test')");
echo "Inserted records:".mysqlii_affected_rows();
?>
```

8. mysqlii_fetch_assoc()

The mysqli_fetch_assoc() used to retrieve a row of data as an associative array from a Mysqli result handle.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqlii_connect($hostname, $username, $password,$dbname);
$result = mysqlii_query(“select * from examples");
while ($row = mysqlii_fetch_assoc($result))
{
echo “Name : “ . $row['name‘] . “Mobile“ . $row[‘city’];
}
?>
```

9. mysqli_num_fields()

The mysqli_num_fields() is used to get number of fields in a result.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqli_connect($hostname, $username, $password,$dbname);
$result = mysqli_query("select name,mobile,city from examples where id='2'");
echo mysqli_num_fields($result);
?>
```

10. mysqli_fetch_fields()

The mysqli_fetch_field() function is used to get the column information of a field from a result handle.

Syntax

```
mysqli_fetch_field (result, field_offset);
```

Example

```
<?php
$result = mysqli_query('select * from examples');
/* get information about the column */
$i = 0;
while ($i < mysqli_num_fields($result)) {
$meta = mysqli_fetch_field($result, $i);
echo "<pre>
max_length:   $meta->max_length
name:         $meta->name
not_null:     $meta->not_null
numeric:      $meta->numeric
primary_key:  $meta->primary_key
table:        $meta->table
type:         $meta->type
unique_key:   $meta->unique_key
```

```
</pre>";
$i++;  }
?>
```

11. mysqli_fetch_object()

The mysqli_fetch_object() is used to fetch a row of data as an object from a result handle.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqli_connect($hostname, $username, $password,$dbname);
$result = mysqli_query("select * from register");
echo "<h2>List of the username:</h2>";
while ($row = mysqli_fetch_object($result))
{
        echo $row->username."<br />";
}
?>
```

12. mysqli_fetch_row()

The mysqli_fetch_row() is used to fetch a row of data from a result handle.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqli_connect($hostname, $username, $password,$dbname);
$result = mysqli_query("select * from register");
$row = mysqli_fetch_row($result);
echo $row[1];
?>
```

13. mysqli_insert_id()

The mysqli_insert_id() is used to get the ID generated by the last insert query.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqli_connect($hostname, $username, $password,$dbname);
```

```
mysqli_query("INSERT INTO register (username,password,mobile,city)
values('ram','abcxyz','9816798168','Rajkot')");
echo "Last inserted record has id : ". mysqli_insert_id();
?>
```

14. mysqli_num_fields()

The mysqli_num_fields() is used to get number of fields in a result.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqli_connect($hostname, $username, $password,$dbname);
$result=mysqli_query(“select username,mobile from register”);
echo mysqli_num_fields($result);
?>
```

15. mysqli_result()

The mysqli_result() is used to fetch the contents of a single field from a mysqli query.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
$dbhandle = mysqli_connect($hostname, $username, $password,$dbname);
$result = mysqli_query("select username from register");
$no_result = mysqli_num_rows($result);
echo "<h2>Here is a list of the username :</h2>";
for ($i=0;$i<=$no_result;$i++)
{
echo "<p>".mysqli_result($result,$i)."</p>";
}
?>
```

16. mysqli_list_tables(), mysqli_tablename()

mysqli_list_tables -- List tables in a Mysqli database
mysqli_tablename -- Get table name of field.

Example

```
<?php
$username = "root";
$password = "";
$hostname = "localhost";
$dbname = “database name”;
//connection to the database
```

```
$dbhandle = mysqli_connect($hostname, $username, $password,$dbname);
$result = mysqli_list_tables("examples");
$num_rows = mysqli_num_rows($result);
for ($i = 0; $i < $num_rows; $i++) {
   echo "Table: ", mysqli_tablename($result, $i), "\n";  }
mysqli_free_result($result);
?>
```

17. mysqli_list_fields()

mysqli_list_fields -- List Mysqli table fields.

Syntax

mysqli_list_fields (string database_name, string table_name);

Example

```
<?php
$link = mysqli_connect("localhost","root","")
or die ("Could not connect");
$fields = mysqli_list_fields("examples","register",$link);
$columns = mysqli_num_fields($fields);
for ($i = 0; $i < $columns; $i++) {
echo mysqli_field_name($fields, $i) . "\n";
}
?>
```

18. mysqli_field_type()

mysqli_field_type -- Get the type of the specified field in a result.

Syntax

mysqli_field_type (result, field_offset);

Example

```
<?php
$con=mysqli_connect("localhost","root","");
$selectdb=mysqli_select_db("examples",$con);
$result=mysqli_query("select * from register");
$fields=mysqli_num_fields($result);
$rows=mysqli_num_rows($result);
$table_name=mysqli_field_table($result, 0);
echo $table_name . "' table has " . $fields . " fields and " . $rows . " record(s)<br />";
echo "<p>Here is the list of the fields:</p>";
for ($i=0; $i < $fields; $i++) {
   $type  = mysqli_field_type($result, $i);
   $name  = mysqli_field_name($result, $i);
   $len   = mysqli_field_len($result, $i);
   echo $type . " " . $name . " " . $len ."<br /> ";  }
?>
```

19. mysqli_db_name()

The mysqli_db_name() function retrieve the database name from a call to the mysqli_list_dbs() function.

Syntax

mysqli_db_name(result, row);

Example

```
<?php
$con=mysqli_connect('localhost','root','');
$db_list = mysqli_list_dbs($con);
$i = 0;
$no_of_rows = mysqli_num_rows($db_list);
echo "<h2>List of databases</h2>";
while ($i < $no_of_rows)
{
echo mysqli_db_name($db_list, $i) . "<br />";
$i++;  }
?>
```

20. mysqli_data_seek()

The mysqli_data_seek() function is used to move internal result pointer.

Syntax

```
mysqli_data_seek(result, row_number);
```

Example

```
<?php
$con = mysqli_connect("localhost","root","");
$selecteddb = mysqli_select_db("examples");
$sql = "select * from register";
$result = mysqli_query($sql,$con);
mysqli_data_seek($result,0);
print_r(mysqli_fetch_row($result));
?>
```

Unit 5 - jQuery

What is jQuery?

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto − Write less, do more.
jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.
jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code. Here is the list of important core features supported by jQuery −

- **DOM manipulation** − The jQuery made it easy to select DOM elements, traverse them and modifying their content by using cross-browser open source selector engine called **Sizzle**.
- **Event handling** − The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support** − The jQuery helps you a lot to develop a responsive and feature-rich site using AJAX technology.
- **Animations** − The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight** − The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).
- **Cross Browser Support** − The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Latest Technology** − The jQuery supports CSS3 selectors and basic XPath syntax.

How to use jQuery?
There are two ways to use jQuery.

- **Local Installation** − You can download jQuery library on your local machine and include it in your HTML code.
- **CDN Based Version** − You can include jQuery library into your HTML code directly from Content Delivery Network (CDN).

jQuery Syntax

The jQuery syntax is tailor-made for selecting HTML elements and performing some action on the element(s).
Basic syntax is: $(selector).action()

- A $ sign to define/access jQuery
- A (selector) to "query (or find)" HTML elements
- A jQuery action() to be performed on the element(s)

Examples:
$("p").hide() - hides all <p> elements.

jQuery Selectors

jQuery selectors allow you to select and manipulate HTML element(s).
jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing CSS Selectors, and in addition, it has some own custom selectors.
All selectors in jQuery start with the dollar sign and parentheses: $().

1. The element Selector

The jQuery element selector selects elements based on the element name.

You can select all <p> elements on a page like this:
$("p")
Example

```
$(document).ready(function(){
   $("button").click(function(){
      $("p").hide();
   });
});
```

2. The #id Selector

The jQuery #id selector uses the id attribute of an HTML tag to find the specific element.
An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.
To find an element with a specific id, write a hash character, followed by the id of the HTML element:
$("#test")
Example

```
$(document).ready(function(){
   $("button").click(function(){
      $("#test").hide();    });
});
```

3. The .class Selector

The jQuery class selector finds elements with a specific class.
To find elements with a specific class, write a period character, followed by the name of the class:
$(".test")
Example

```
$(document).ready(function(){
   $("button").click(function(){
      $(".test").hide();
   });
});
```

jQuery Event Methods

Event methods trigger or attach a function to an event handler for the selected elements.

4. **blur()**

The blur event occurs when an element loses focus.
The blur() method triggers the blur event, or attaches a function to run when a blur event occurs.
Syntax
Trigger the blur event for the selected elements:
$(selector).blur()
Attach a function to the blur event:
$(selector).blur(function)

Example

Attach a function to the blur event. The blur event occurs when the <input> field loses focus:

```
$("input").blur(function(){
   alert("This input field has lost its focus.");
});
```

5. **change()**

The change event occurs when the value of an element has been changed (only works on <input>, <textarea> and <select> elements).
The change() method triggers the change event, or attaches a function to run when a change event occurs.

Syntax

Trigger the change event for the selected elements:
$(selector).change()
Attach a function to the change event:
$(selector).change(function)

Example

Alert a text when an <input> field is changed:

```
$("input").change(function(){
  alert("The text has been changed.");
});
```

6. click()

The click event occurs when an element is clicked.
The click() method triggers the click event, or attaches a function to run when a click event occurs.

Syntax

Trigger the click event for the selected elements:
$(selector).click()
Attach a function to the click event:
$(selector).click(function)

Example

Click on a <p> element to alert a text:

```
$("p").click(function(){
  alert("The paragraph was clicked.");
});
```

7. **dblclick()**

The dblclick event occurs when an element is double-clicked.
The dblclick() method triggers the dblclick event, or attaches a function to run when a dblclick event occurs.

Syntax

Trigger the dblclick event for the selected elements:
$(selector).dblclick()
Attach a function to the dblclick event:
$(selector).dblclick(function)

Example

Double-click on a <p> element to alert a text:

```
$("p").dblclick(function(){
  alert("The paragraph was double-clicked");
});
```

8. **focus()**

The focus event occurs when an element gets focus (when selected by a mouse click or by "tab-navigating" to it).

The focus() method triggers the focus event, or attaches a function to run when a focus event occurs.

Syntax

Trigger the focus event for selected elements:

$(selector).focus()

Attach a function to the focus event:

$(selector).focus(function)

Example

Attach a function to the focus event. The focus event occurs when the <input> field gets focus:

```
$("input").focus(function(){
  $("span").css("display", "inline").fadeOut(2000);
});
```

9. **keydown()**

The keydown event occurs when a keyboard key is pressed down.

The keydown() method triggers the keydown event, or attaches a function to run when a keydown event occurs.

Syntax

Trigger the keydown event for the selected elements:

$(selector).keydown()

Attach a function to the keydown event:

$(selector).keydown(function)

Example

Set the background color of an <input> field when a keyboard key is pressed down:

```
$("input").keydown(function(){
  $("input").css("background-color", "yellow");
});
```

10. **keypress()**

The keypress() method triggers the keypress event, or attaches a function to run when a keypress event occurs.

The keypress event is similar to the keydown event. The event occurs when a button is pressed down. However, the keypress event is not fired for all keys (e.g. ALT, CTRL, SHIFT, ESC). Use the keydown() method to also check these keys.

Syntax

Trigger the keypress event for the selected elements:

$(selector).keypress()

Attach a function to the keypress event:

$(selector).keypress(function)

Example

Count the number of key presses in an <input> field:

```
$("input").keypress(function(){
  $("span").text(i += 1);
});
```

11. **keyup()**

The keyup event occurs when a keyboard key is released.

The keyup() method triggers the keyup event, or attaches a function to run when a keyup event occurs.

Syntax

Trigger the keyup event for the selected elements:

```
$(selector).keyup()
```

Attach a function to the keyup event:

```
$(selector).keyup(function)
```

Example

Set the background color of an <input> field when a keyboard key is released:

```
$("input").keyup(function(){
  $("input").css("background-color", "pink");
});
```

12. **load()**

The load() method attaches an event handler to the load event.

The load event occurs when a specified element has been loaded.

This event works with elements associated with a URL (image, script, frame, iframe), and the window object.

Syntax

```
$(selector).load(function)
```

Example

Alert a text when an image is fully loaded:

```
$("img").load(function(){
  alert("Image loaded.");
});
```

13. **resize()**

The resize event occurs when the browser window changes size.

The resize() method triggers the resize event, or attaches a function to run when a resize event occurs.

Syntax

Trigger the resize event for the selected elements:

```
$(selector).resize()
```

Attach a function to the reisize event:

```
$(selector).resize(function)
```

Example

Count the number of times the browser window is resized:

```
$(window).resize(function(){
  $('span').text(x += 1);
});
```

14. **scroll()**

The scroll event occurs when the user scrolls in the specified element.

The scroll event works for all scrollable elements and the window object (browser window).

The scroll() method triggers the scroll event, or attaches a function to run when a scroll event occurs.

Syntax
Trigger the scroll event for the selected elements:
$(selector).scroll()
Attach a function to the scroll event:
$(selector).scroll(function)

Example
Count the number of times the scroll is used for an element:

```
$("div").scroll(function(){
  $("span").text(x += 1);
});
```

15. **submit()**
The submit event occurs when a form is submitted.
This event can only be be used on <form> elements.
The submit() method triggers the submit event, or attaches a function to run when a submit event occurs.
Syntax
Trigger the submit event for the selected elements:
$(selector).submit()
Attach a function to the submit event:
$(selector).submit(function)

Example
Display an alert when a form is submitted:

```
$("form").submit(function(){
  alert("Submitted");
});
```

16. **unload()**
The unload event occurs when the user navigates away from the page.
The unload event is triggered when:

- a link to leave the page is clicked
- a new URL is typed in the address bar
- the forward or back buttons are used
- the browser window is closed
- the page is reloaded

The unload() method specifies what happens when a unload event occurs.
The unload() method should only be used on the window object.
Syntax
$(selector).unload(function)

Example
Alert a message when navigating away from the page:

```
$(window).unload(function(){
  alert("Goodbye!");
});
```

jQuery - Effects

jQuery provides a trivially simple interface for doing various kind of amazing effects. jQuery methods allow us to quickly apply commonly used effects with a minimum configuration.

1. show() and hide()

The commands for showing and hiding elements are pretty much what we would expect − show() to show the elements in a wrapped set and hide() to hide them.

Syntax

Here is the simple syntax for show() method −

[selector].show(speed, [callback]);

Here is the description of all the parameters −

- **speed** − A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- **callback** − This optional parameter represents a function to be executed whenever the animation completes; executes once for each element animated against.

Following is the simple syntax for **hide()** method −

[selector].hide(speed, [callback]);

Here is the description of all the parameters −

- **speed** − A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- **callback** − This optional parameter represents a function to be executed whenever the animation completes; executes once for each element animated against.

Example

Consider the following HTML file with a small JQuery coding −

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $("#show").click(function () {
          $(".mydiv").show( 1000 );
        });

        $("#hide").click(function () {
          $(".mydiv").hide( 1000 );
        });
      });
    </script>
    <style>
      .mydiv{ margin:10px;padding:12px; border:2px solid #666; width:100px; height:100px;}
    </style>
  </head>
  <body>
    <div class = "mydiv">
      This is a SQUARE
    </div>
    <input id = "hide" type = "button" value = "Hide" />
```

```
    <input id = "show" type = "button" value = "Show" />
  </body>
</html>
```

2. **fadeIn()**

The **fadeIn( )** method fades in all matched elements by adjusting their opacity and firing an optional callback after completion.

Syntax

Here is the simple syntax to use this method −

selector.fadeIn(speed, [callback]);

Parameters

Here is the description of all the parameters used by this method −

- **speed** − A string representing one of the three predefined speeds ("slow", "def", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- **callback** − This is optional parameter representing a function to call once the animation is complete.

fadeout()

The **fadeOut( )** method fades out all matched elements by adjusting their opacity to 0, then setting display to "none" and firing an optional callback after completion.

Syntax

Here is the simple syntax to use this method −

selector.fadeOut(speed, [callback]);

Parameters

Here is the description of all the parameters used by this method −

- **speed** − A string representing one of the three predefined speeds ("slow", "def", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- **callback** − This is optional parameter representing a function to call once the animation is complete.

Example

Following is a simple example a simple showing the usage of this method −

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {
        $("#in").click(function(){
          $(".target").fadeIn( 'slow', function(){
            $(".log").text('Fade In Transition Complete');
          });
        });
        $("#out").click(function(){
          $(".target").fadeOut( 'slow', function(){
            $(".log").text('Fade Out Transition Complete');
          });
        });
```

```
      });
    </script>

    <style>
      p {background-color:#bca; width:200px; border:1px solid green;}
    </style>
  </head>
  <body>
    <p>Click on any of the buttons</p>
    <button id = "out"> Fade Out </button>
    <button id = "in"> Fade In</button>
    <div class = "target">
      <img src = "../images/jquery.jpg" alt = "jQuery" />
    </div>
    <div class = "log"></div>
  </body>
</html>
```

3. **slideDown()**

The **slideDown()** method reveals all matched elements by adjusting their height and firing an optional callback after completion.

Syntax

Here is the simple syntax to use this method −

```
selector.slideDown( speed, [callback] );
```

Parameters

Here is the description of all the parameters used by this method −

- **speed** − A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- **callback** − This is optional parameter representing a function to call once the animation is complete.

slideUp()

The **slideUp()** method hides all matched elements by adjusting their height and firing an optional callback after completion.

Syntax

Here is the simple syntax to use this method −

```
selector.slideUp( speed, [callback] );
```

Parameters

Here is the description of all the parameters used by this method −

- **speed** − A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- **callback** − This is optional parameter representing a function to call once the animation is complete.

Example

Following is a simple example a simple showing the usage of this method −

```
<html>
  <head>
    <title>The jQuery Example</title>
```

```
    <script type = "text/javascript"
      src = "https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>

    <script type = "text/javascript" language = "javascript">
      $(document).ready(function() {

        $("#down").click(function(){
          $(".target").slideDown( 'slow', function(){
            $(".log").text('Slide Down Transition Complete');
          });
        });

        $("#up").click(function(){
          $(".target").slideUp( 'slow', function(){
            $(".log").text('Slide Up Transition Complete');
          });
        });

      });
    </script>

    <style>
      p {background-color:#bca; width:200px; border:1px solid green;}
    </style>
  </head>
  <body>
    <p>Click on any of the buttons</p>
    <button id = "up"> Slide Up </button>
    <button id = "down"> Slide Down</button>
    <div class = "target">
      <img src = "../images/jquery.jpg" alt = "jQuery" />
    </div>
    <div class = "log"></div>
  </body>
</html>
```