INTRODUCTION TO PHP:

The origins of PHP date back to 1995 when an independent software development contractor named Rasmus Lerdorf developed a Perl/CGI script that enabled him to know how many visitors were reading his online resume.

His script performed two tasks: logging visitor information, and displaying the count of visitors to the Web page. Because the Web as we know it today was still young at that time, tools such as these were nonexistent, and they prompted e-mails inquiring about Lerdorf's scripts. Lerdorf thus began giving away his toolset, dubbed Personal Home Page (PHP).

The clamor for the PHP toolset prompted Lerdorf to continue developing the language, with perhaps the most notable early change being a new feature for converting data entered in an HTML form into symbolic variables, encouraging exportation into other systems. To accomplish this, he opted to continue development in C code rather than Perl.

Ongoing additions to the PHP toolset culminated in November 1997 with the release of PHP 2.0, or Personal Home Page/Form Interpreter (PHP/FI). As a result of PHP's rising popularity, the 2.0 release was accompanied by a number of enhancements and improvements from programmers worldwide.

The new PHP release was extremely popular, and a core team of developers soon joined Lerdorf. They kept the original concept of incorporating code directly alongside HTML and rewrote the parsing engine, giving birth to PHP 3.0. By the June 1998 release of version 3.0, more than 50,000 users were using PHP to enhance their Web pages.

Development continued at a hectic pace over the next two years, with hundreds of functions being added and the user count growing in leaps and bounds. At the beginning of 1999, Netcraft an Internet research and analysis company, reported a conservative estimate of a user base of more than 1 million, making PHP one of the most popular scripting languages in the world.

Two core developers, Zeev Suraski and Andi Gutmans, took the initiative to completely rethink the way PHP operated, culminating in a rewriting of the PHP parser, dubbed the Zend scripting engine. The result of this work was in the PHP 4 release.

PHP4:

On May 22, 2000, roughly 18 months after the first official announcement of the new development effort, PHP 4.0 was released. Just a few months after the major release, Netcraft estimated that PHP had been installed on more than 3.6 million domains.

PHP 4 added several enterprise-level improvements to the language, including the following:

Improved Resource Handling:

One of version 3.X's primary drawbacks was scalability. This was largely because the designers underestimated how rapidly the language would be adopted for large-scale applications. The language wasn't originally intended to run enterprise-class Web sites, and continued interest in using it for such purposes caused the developers to rethink much of the language's mechanics in this regard.

Object-Oriented Support:

Version 4 incorporated a degree of object-oriented functionality. The new features played an important role in attracting users used to working with traditional object-oriented programming (OOP) languages. Standard class and object development

methodologies were made available in addition to features such as object overloading and run-time class information.

Native session-handling support:

HTTP session handling, available to version 3.X users through the third-party package PHPLIB, was natively incorporated into version 4. This feature offers developers a means for tracking user activity and preferences with unparalleled efficiency and ease.

Encryption:

MCrypt library was incorporated into the default distribution, offering users both full and hash encryption using encryption algorithms including Blowfish, MD5, SHA1, and TripleDES, among others.

ISAPI Support:

ISAPI support offered users the ability to use PHP in conjunction with Microsoft's IIS Web server.

Native java support:

In another boost to PHP's interoperability, support for binding to Java objects from a PHP application was made available in version 4.0. In addition to these features, literally hundreds of functions were added to version 4, greatly enhancing the language's capabilities.

PHP 5:

Version 5 was yet another watershed in the evolution of the PHP language. It contains improvements over existing functionality and adds several features commonly associated with mature programming language architectures:

Vastly improved Object-Oriented Capabilities:

Version 5 includes numerous functional additions such as explicit constructors and destructors, object cloning, class abstraction, variable scope, and interfaces, and a major improvement regarding how PHP handles object management.

Try/Catch Exception Handling:

Devising custom error-handling strategies within structural programming languages is, ironically, error-prone and inconsistent. To remedy this problem, version 5 supports exception handling.

Improved XML AND Web Services support:

XML support is now based on the libxml2 library, and a new and rather promising extension for parsing and manipulating XML, known as SimpleXML, has been introduced. In addition, a SOAP extension is now available.

Native Support for SQLITE:

Always keen on choice, the developers added support for the powerful yet compact SQLite database server. SQLite offers a convenient solution for developers looking for many of the features found in some of the heavyweight database products without incurring the accompanying administrative overhead.

With the release of version 5, PHP's popularity hit what was at the time a historical high, having been installed on almost 19 million domains, according to Netcraft.

PHP 6:

PHP 6 beta version was scheduled to be released by the conclusion of 2007. A list of highlights is found here:

UNICODE SUPPORT:

Native Unicode support has been added, making it much easier to build and maintain multilingual applications.

Security Improvements:

A considerable number of security-minded improvements have been made that should greatly decrease the prevalence of security related gaffes that to be frank aren't so much a fault of the language, but are due to inexperienced programmers running with scissors, so to speak.

New Language Features and Constructs:

A number of new syntax features have been added, including, most notably, a 64-bit integer type, a revamped foreach looping construct for multidimensional arrays, and support for labeled breaks.

According to Netcraft, PHP has been installed on more than 20 million domains.

General Language Features:

Every user has his or her own specific reason for using PHP to implement a mission critical application, although one could argue that such motives tend to fall into four key categories: Practicality, Power, Possibility, and Price.

Practicality:

From the very start, the PHP language was created with practicality in mind. After all, Lerdorf's original intention was not to design an entirely new language, but to resolve a problem that had no readily available solution.

Furthermore, much of PHP's early evolution was not the result of the explicit intention to improve the language itself, but rather to increase its utility to the user. The result is a language that allows the user to build powerful applications even with a minimum of knowledge. For instance, a useful PHP script can consist of as little as one line; unlike C, there is no need for the mandatory inclusion of libraries.

PHP is a *loosely typed* language, meaning there is no need to explicitly create, typecast, or destroy a variable, although you are not prevented from doing so. PHP handles such matters internally, creating variables on the fly as they are called in a script, and employing a best-guess formula for automatically typecasting variables.

PHP will also automatically destroy variables and return resources to the system when the script completes.

Power:

PHP developers have more than 180 libraries at their disposal, collectively containing well over 1,000 functions. Although you're likely aware of PHP's ability to interface with databases, manipulate form information, and create pages dynamically, you might not know that PHP can also do the following:

1. Create and manipulate Adobe Flash and Portable Document Format (PDF) files.
2. Evaluate a password for guessability by comparing it to language dictionaries and easily broken patterns.
3. Parse even the most complex of strings using the POSIX and Perl-based regular expression libraries.
4. Authenticate users against login credentials stored in flat files, databases, and even Microsoft's Active Directory.
5. Communicate with a wide variety of protocols, including LDAP, IMAP, POP3, NNTP, and DNS, among others.
6. Tightly integrate with a wide array of credit-card processing solutions.

Possibility:

PHP developers are rarely bound to any single implementation solution. On the contrary, a user is typically fraught with choices offered by the language.

For example, consider PHP's array of database support options. Native support is offered for more than 25 database products, including Adabas D, dBase, Empress, FilePro, FrontBase, Hyperwave, IBM DB2, Informix, Ingres, InterBase, mSQL, Microsoft SQL Server, MySQL, Oracle, Ovrimos, PostgreSQL, Solid, Sybase, UNIX dbm, and Velocis.

PHP's flexible string-parsing capabilities offer users of differing skill sets the opportunity to not only immediately begin performing complex string operations but also to quickly port programs of similar functionality over to PHP.

Price:

PHP is available free of charge! Since its inception, PHP has been without usage, modification, and redistribution restrictions. In recent years, software meeting such open licensing qualifications has been referred to as *open source* software. Open source software and the Internet go together like bread and butter.

DRAWBACKS OF OTHER TECHNOLOGIES LIKE SERVLETS AND JSP:

SERVLETS:

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So Servlets are trusted.
- The full functionality of the Java class libraries is available to a Servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

DRAWBACKS OF SERVLETS:

1. *In many Java Servlet-based applications, processing the request and generating the response are both handled by a single Servlet class.*
2. *Thorough Java programming knowledge is needed to develop and maintain all aspects of the application, since the processing code and the HTML elements are lumped together.*
3. *Changing the look and feel of the application, or adding support for a new type of client, requires the Servlet code to be updated and recompiled*
4. *To develop the web page layout, the generated HTML must be manually embedded into the Servlet code, a process which is time consuming, error prone, and extremely boring.*

JSP:

Java Server Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A Java Server Pages component is a type of Java Servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

DRAWBACKS OF JSP:

1. *JSP pages require about double the disk space to hold the page.*
   Reason: *Because JSP pages are translated into class files, the server has to store the resultant class files with the JSP pages.*

2. *JSP pages must be compiled on the server when first accessed. This initial compilation produces a noticeable delay when accessing the JSP page for the first time.*
   Reason: *The developer may compile JSP pages and place them on the server in compiled form (as one or more class files) to speed up the initial page access. The JSP developer may need to bring down the server to make the changed classes corresponding to the changed JSP page.*

3. *Debugging JSP programs is still a major challenge. You have to remember that a jsp page will be first converted to a .java file and then compiled. Thus, any errors will be pointing to line numbers in the converted .java file and not the .jsp page.*

4. *Database connectivity is not as easy as it should be.*

5. *JSP makes it tempting to put Java code in the web page, even though that's considered bad design. Or JSP can actually demand putting Java code in the page.*

6. *Doing even a simple task such as header and footer includes is overly difficult with JSP.*

7. *Looping is overly difficult in JSP.*

8. *JSP requires a compiler be shipped with the web server.*

9. *JSP consumes extra hard drive space and extra memory space.*

DOWNLOADING, INSTALLING AND CONFIGURING PHP:

Downloading PHP:

Download the latest stable version from the PHP Web site i.e. “www.php.net”. To decrease download time, choose from the approximately 100 mirrors residing in more than 50 countries, a list of which is available here: “http://www.php.net/mirrors.php”. Once you’ve chosen the closest mirror, navigate to download page and choose one of the available distributions:

Source:

If Linux is your target server platform, or if you plan to compile from source for the Windows platform, choose this distribution format. Building from source on Windows isn’t recommended. Unless your situation warrants very special circumstances, the prebuilt Windows binary will suit your needs just fine. This distribution is compressed in Bzip2 and Gzip formats.

Windows zip package:

If you plan to use PHP in conjunction with Apache on Windows, you should download this distribution because it's the focus of the later installation instructions.

Windows installer:

This version offers a convenient Windows installer interface for installing and configuring PHP, and support for automatically configuring the IIS, PWS, and Xitami servers. Although you could use this version in conjunction with Apache, it is not recommended. Instead, use the Windows zip package version.

Installing Apache & PHP on Windows:

1. Start the Apache installer by double-clicking the apache_X.X.XX-win32-x86-no_ssl.msi icon. The X's in this file name represent the latest stable version numbers of the distributions you downloaded.

2. The installation process begins with a welcome screen. Take a moment to read the screen and then click next.

3. The license agreement is displayed next. Carefully read through the license. Assuming that you agree with the license stipulations, click next.

4. A screen containing various items pertinent to the Apache server is displayed next. Take a moment to read through this information and then click next.

5. You will be prompted for various items pertinent to the server's operation, including network domain, server name, & administrator's e-mail address. If you know this information, fill it in now; otherwise, just enter localhost for the first two items and put in any e-mail address for the last. You can always change this information later in the httpd.conf file.

6. You'll also be prompted as to whether Apache should run as a service for all users or only for the current user. If you want Apache to automatically start with the operating system, which is recommended, then choose to install Apache as a service for all users. When you're finished, click next.

7. You are prompted for a Setup Type: Typical or Custom. Unless there is a specific reason you don't want the Apache documentation installed, choose Typical and click next. Otherwise, choose Custom, click next, and on the next screen, uncheck the Apache Documentation option.

8. You're prompted for the Destination folder. By default, this is C:\Program Files\Apache Group. Consider changing this to C:\, which will create an installation directory C:\apache2\. Regardless of what you choose, keep in mind that the latter is used here for the sake of convention. Click next.

9. Click Install to complete the installation. That's it for Apache. Next you'll install PHP.

10. Unzip the PHP package, placing the contents into C:\php6\. You're free to choose any installation directory you please, but avoid choosing a path that contains spaces.

11. Navigate to C:\apache2\conf and open httpd.conf for editing.

12. Add the following three lines to the httpd.conf file. Consider adding them directly below the block of LoadModule entries located in the bottom of the Global Environment section:

    LoadModule php6_module c:/php6/php6apache2.dll
    AddType application/x-httpd-php .php
    PHPIniDir "c:\php6"

13. Rename the php.ini-dist file to php.ini and save it to the C:\php6 directory. The php.ini file contains hundreds of directives that are responsible for tweaking PHP's behavior.

14. If you're using Windows NT, 2000, XP, or Vista, navigate to Start → Settings → Control Panel → Administrative Tools → Services.

15. Locate Apache in the list and make sure that it is started. If it is not started, highlight the label and click start the Service, located to the left of the label. If it is started, highlight the label and click Restart the Service, so that the changes made to the httpd.conf file take effect. Next, right-click Apache and choose Properties. Ensure that the startup type is set to Automatic.

INSTALLING IIS AND PHP ON WINDOWS

To install IIS on Windows XP, you will need your Windows XP Professional CD with you and you will need to be able to log in as an administrator on your computer.

Step 1: Control Panel

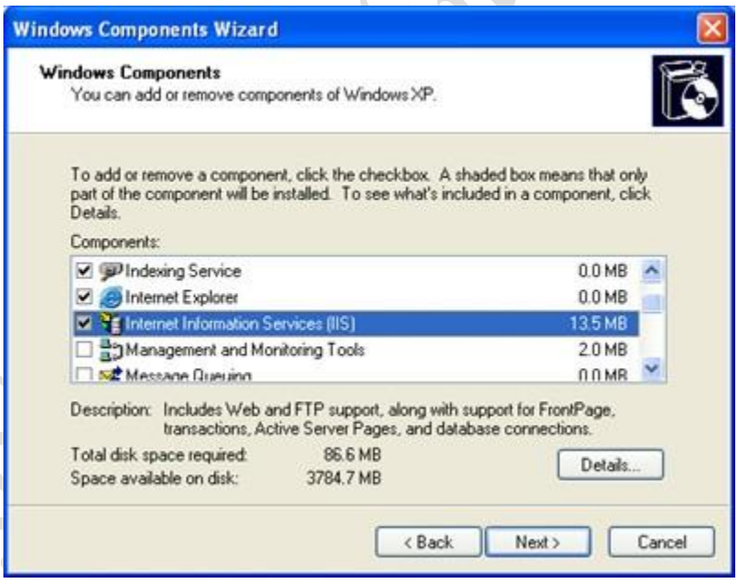
Go to Control Panel -> Add Remove Programs.

Step 2: Windows Configuration

Click Add/Remove Windows Components

Step 3: Select IIS

The Windows Component wizard will appear. You need to select the Internet Information Service (IIS) check box and then click on next.



Step 4: Insert your XP Professional CD

Now you will probably be asked to insert your windows XP professional CD.

Step 5: Installing

Once your Windows XP Professional CD is available you will see a window indicating that Internet Information Service is being installed.

Step 6: Finished. Next install PHP.

Configuring PHP:

A total of 45 extensions are bundled with PHP 5.1 and 5.2, a number that was pared to 35 extensions with the current alpha version of PHP 6. However, to actually use any of these extensions, you need to uncomment the appropriate line within the *php.ini* file.

For example, if you'd like to enable PHP's XML-RPC extension, you need to make a few minor adjustments to your php.ini file:

1. Open the *php.ini* file and locate the ***extension_dir*** directive and assign it C:\php\ext\. If you installed PHP in another directory, modify this path accordingly.
2. Locate the line ***;extension=php_xmlrpc.dll***. Uncomment this line by removing the preceding semicolon. Save and close the file.
3. Restart the web server and the extension is ready to use from within PHP.

When enabling these extensions, you may occasionally need to install other software.

Run-Time Configuration:

It's possible to change PHP's behavior at run time on both Windows and Linux through the php.ini file. PHP's most commonly used configuration directives, introducing the purpose, scope, and default value of each.

The php.ini file is PHP's global configuration file, much like httpd.conf is to Apache. This file addresses 12 different aspects of PHP's behavior:

1	Language Options	7	Data Handling
2	Safe Mode	8	Paths and Directories
3	Syntax Highlighting	9	File Uploads
4	Miscellaneous	10	Fopen Wrappers
5	Resource Limits	11	Dynamic Extensions
6	Error Handling and Logging	12	Module Settings

The php.ini file is a simple text file, consisting solely of comments and the directives and their corresponding values. Here's a sample snippet from the file:

```
;
; Allow the <? tag
;
short_open_tag = Off
```

Lines beginning with a semicolon are comments; the parameter short_open_tag is assigned the value Off.

When PHP is running as an Apache module, you can modify many of the directives through either the httpd.conf file or the .htaccess file. This is accomplished by prefixing directive/value assignment with one of the following keywords:

1. php_value: Sets the value of the specified directive.
2. php_flag: Sets the value of the specified Boolean directive.
3. php_admin_value: Sets the value of the specified directive. This differs from php_value in that it cannot be used within an .htaccess file and cannot be overridden within virtual hosts or .htaccess.
4. php_admin_flag: Sets the value of the specified directive. This differs from php_value in that it cannot be used within an .htaccess file and cannot be overridden within virtual hosts or .htaccess.

For example, to disable the short tags directive and prevent others from overriding it, add the following line to your httpd.conf file: ***php_admin_flag short_open_tag Off***

Configuration Directive Scope:

Configuration directives cannot be modified anywhere, because of a variety of reasons, mostly security related. Each directive is assigned a scope, and the directive can be modified only within that scope. In total, there are four scopes:

1. PHP_INI_PERDIR: Directive can be modified within the php.ini, httpd.conf, or .htaccess files
2. PHP_INI_SYSTEM: Directive can be modified within the php.ini and httpd.conf files.
3. PHP_INI_USER: Directive can be modified within user scripts.
4. PHP_INI_ALL: Directive can be modified anywhere.

PHP's Configuration Directives:

- ***Language Options:***

The directives located in this section determine some of the language's most basic behavior.

engine = *On* | *Off*
Scope: PHP_INI_ALL; Default value: On

This parameter is responsible for determining whether the PHP engine is available. Turning it off prevents you from using PHP at all. Obviously, you should leave this enabled if you plan to use PHP.

zend.ze1_compatibility_mode = *On* | *Off*
Scope: PHP_INI_ALL; Default value: Off

Some three years after PHP 5.0 was released, PHP 4.X is still in widespread use. One of the reasons for the protracted upgrade cycle is due to some significant object-oriented incompatibilities between PHP 4 and 5. The zend.ze1_compatibility_mode directive attempts to revert several of these changes in PHP 5, raising the possibility that PHP 4 applications can continue to run without change in version 5.

short_open_tag = *On* | *Off*
Scope: PHP_INI_ALL; Default value: On

PHP script components are enclosed within escape syntax. There are four different escape formats, the shortest of which is known as *short open tags*, which looks like this:

```
<?
echo "Some PHP statement";
?>
```

You may recognize that this syntax is shared with XML, which could cause issues in certain environments. Thus, a means for disabling this particular format has been provided. When short_open_tag is enabled (On), short tags are allowed; when disabled (Off), they are not.

asp_tags = *On* | *Off*
Scope: PHP_INI_ALL; Default value: Off

PHP supports ASP-style script delimiters, which look like this:

```
<%
echo "Some PHP statement";
%>
```

If you're coming from an ASP background and prefer to continue using this delimiter syntax, you can do so by enabling this tag.

precision = *integer*
Scope: PHP_INI_ALL; Default value: 12

PHP supports a wide variety of data types, including floating-point numbers. The precision parameter specifies the number of significant digits displayed in a floating point number representation. Note that this value is set to 14 digits on Win32 systems and to 12 digits on Linux.

implicit_flush = *On* | *Off*
Scope: PHP_INI_SYSTEM; Default value: Off

Enabling implicit_flush results in automatically clearing, or flushing, the output buffer of its contents after each call to print() or echo(), and completing each embedded HTML block. This might be useful in an instance where the server requires an unusually long period of time to compile results or perform certain calculations.

allow_call_time_pass_reference = *On* | *Off*
Scope: PHP_INI_SYSTEM; Default value: On

Function arguments can be passed in two ways: by value and by reference. Exactly how each argument is passed to a function at function call time can be specified in the function definition, which is the recommended means for doing so. However, you can force all arguments to be passed by reference at function call time by enabling allow_call_time_pass_reference.

- ***Safe Mode:***

When you deploy PHP in a multiuser environment, such as that found on an ISP's shared server, you might want to limit its functionality. Enabling safe mode opens up the possibility for activating a number of other restrictions via other PHP configuration directives.

safe_mode = *On* | *Off*
Scope: PHP_INI_SYSTEM; Default value: Off

Enabling this directive results in PHP being run under the afore mentioned constraints.

safe_mode_gid = *On* | *Off*
Scope: PHP_INI_SYSTEM; Default value: Off

When safe mode is enabled, an enabled safe_mode_gid enforces a GID (group ID) check when opening files. When safe_mode_gid is disabled, a more restrictive UID (user ID) check is enforced.

safe_mode_include_dir = *string*
Scope: PHP_INI_SYSTEM; Default value: NULL

The safe_mode_include_dir provides a safe haven from the UID/GID checks enforced when safe_mode and potentially safe_mode_gid are enabled. UID/GID checks are ignored when files are opened from the assigned directory.

safe_mode_exec_dir = *string*
Scope: PHP_INI_SYSTEM; Default value: NULL

When safe mode is enabled, the safe_mode_exec_dir parameter restricts execution of executables via the exec() function to the assigned directory.

- ***Syntax Highlighting:***

PHP can display and highlight source code. You can control the color of strings, comments, keywords, the background, default text, and HTML components of the highlighted source through the following six directives. Each can be assigned an RGB, hexadecimal, or keyword representation of each color. For example, the color we commonly refer to as *black* can be represented as rgb(0,0,0), #000000, or black, respectively.

highlight.string = *string*
Scope: PHP_INI_ALL; Default value: #DD0000

highlight.comment = *string*
Scope: PHP_INI_ALL; Default value: #FF9900

highlight.keyword = *string*
Scope: PHP_INI_ALL; Default value: #007700

highlight.bg = *string*
Scope: PHP_INI_ALL; Default value: #FFFFFF

highlight.default = *string*
Scope: PHP_INI_ALL; Default value: #0000BB

highlight.html = *string*
Scope: PHP_INI_ALL; Default value: #000000

- ***Miscellaneous:***

The Miscellaneous category consists of a single directive, *expose_php*.

expose_php = *On* | *Off*
Scope: PHP_INI_SYSTEM; Default value: On

Each scrap of information that a potential attacker can gather about a Web server increases the chances that he will successfully compromise it. One simple way to obtain key information about server characteristics is via the server signature.

For example Apache will broadcast the following information within each response header by default:

Apache/2.2.0 (Unix) PHP/6.0.0 PHP/6.0.0-dev Server at www.example.com Port 80

Disabling expose_php prevents the Web server signature (if enabled) from broadcasting the fact that PHP is installed.

- ***Resource Limits:***

We must be careful to ensure that scripts do not monopolize server resources as a result of either programmer- or user-initiated actions. Three particular areas where such overconsumption is prevalent are script execution time, script input processing time, and memory. Each can be controlled via the following three directives.

max_execution_time = *integer*
Scope: PHP_INI_ALL; Default value: 30

The max_execution_time parameter places an upper limit on the amount of time, in seconds, that a PHP script can execute. Setting this parameter to 0 disables any maximum limit.

max_input_time = *integer*
Scope: PHP_INI_ALL; Default value: 60

The max_input_time parameter places a limit on the amount of time, in seconds, that a PHP script devotes to parsing request data. This parameter is particularly important when you upload large files using PHP's file upload feature.

memory_limit = *integer*M
Scope: PHP_INI_ALL; Default value: 8M

The memory_limit parameter determines the maximum amount of memory, in megabytes, that can be allocated to a PHP script.

- ***Data Handling:***

The parameters introduced in this section affect the way that PHP handles external variables— that is, variables passed into the script via some outside source.

GET, POST, cookies, the operating system, and the server are all possible candidates for providing external data.

arg_separator.output = *string*
Scope: PHP_INI_ALL; Default value: &

PHP is capable of automatically generating URLs and uses the standard ampersand (&) to separate input variables. However, if you need to override this convention, you can do so by using the arg_separator.output directive.

arg_separator.input = *string*
Scope: PHP_INI_ALL; Default value: ;&

The ampersand (&) is the standard character used to separate input variables passed in via the POST or GET methods. Although unlikely, should you need to override this convention within your PHP applications, you can do so by using the arg_separator.input directive.

variables_order = *string*
Scope: PHP_INI_ALL; Default value: EGPCS

The variables_order directive determines the order in which the ENVIRONMENT, GET, POST, COOKIE, and SERVER variables are parsed. While seemingly irrelevant, if *register_globals* is enabled (not recommended), the ordering of these values could result in unexpected results due to later variables overwriting those parsed earlier in the process.

- ***Paths and Directories:***

It introduces directives that determine PHP's default path settings. These paths are used for including libraries and extensions, as well as for determining user Web directories and Web document roots.

include_path = *string*
Scope: PHP_INI_ALL; Default value: NULL

The path to which this parameter is set serves as the base path used by functions such as include(), require(), and fopen_with_path().

doc_root = *string*
Scope: PHP_INI_SYSTEM; Default value: NULL

This parameter determines the default from which all PHP scripts will be served. This parameter is used only if it is not empty.

enable_dl = *On* | *Off*
Scope: PHP_INI_SYSTEM; Default value: On

The enable_dl() function allows a user to load a PHP extension at run time—that is, during a script's execution.

- ***Fopen Wrappers:***

This section contains five directives pertinent to the access and manipulation of remote files.

allow_url_fopen = *On* | *Off*
Scope: PHP_INI_ALL; Default value: On

Enabling allow_url_fopen allows PHP to treat remote files almost as if they were local. When enabled, a PHP script can access and modify files residing on remote servers, if the files have the correct permissions.

from = *string*
Scope: PHP_INI_ALL; Default value: NULL

The title of “from” directive is perhaps misleading in that it actually determines the password, rather than the identity, of the anonymous user used to perform FTP connections. Therefore, if from is set like this:

from = "jason@example.com"

the username anonymous and password jason@example.com will be passed to the server when authentication is requested.

user_agent = *string*
Scope: PHP_INI_ALL; Default value: NULL

PHP always sends a content header along with its processed output, including a user agent attribute. This directive determines the value of that attribute.

default_socket_timeout = *integer*
Scope: PHP_INI_ALL; Default value: 60

This directive determines the time-out value of a socket-based stream, in seconds.

auto_detect_line_endings = *On* | *Off*
Scope: PHP_INI_ALL; Default value: Off

One never-ending source of developer frustration is derived from the end-of-line (EOL) character because of the varying syntax employed by different operating systems.

- ***Dynamic Extensions:***

This section contains a single directive, extension.

extension = *string*
Scope: PHP_INI_ALL; Default value: NULL

The extension directive is used to dynamically load a particular module. On the Win32 operating system, a module might be loaded like this:

extension = php_java.dll

PROGRAMMING IN WEB ENVIRONMENT:

Web development is a broad term for the work involved in developing a web site for the Internet (World Wide Web) or an intranet (a private network).

Web development can range from developing the simplest static single page of plain text to the most complex web-based internet applications, electronic businesses, and social network services.

A more comprehensive list of tasks to which web development commonly refers, may include web design, web content development, client liaison, client-side/server-side scripting, web server and network security configuration, and e-commerce development.

Among web professionals, "web development" usually refers to the main non-design aspects of building web sites: *writing markup and coding (scripting)*.

Web content development is the process of researching, writing, gathering, organizing, and editing information for publication on web sites. Web site content may consist of prose, graphics, pictures, recordings, movies or other digital assets that could be distributed by a hypertext transfer protocol server, and viewed by a web browser.

Web Development can be split into many areas and a typical and basic web development hierarchy might consist of:

Client side coding:

Client side coding such as XHTML is executed and stored on a local client (in a web browser) whereas server side code is not available to a client and is executed on a web server which generates the appropriate XHTML which is then sent to the client.

The nature of client side coding allows you to alter the HTML on a local client and refresh the pages with updated content (locally), web designers must bear in mind the importance and relevance to security with their server side scripts.

If a server side script accepts content from a locally modified client side script, the web development of that page is poorly sanitized with relation to security.

With the help of client side programming or coding we can do the following:

1. Make interactive WebPages.
2. Make stuff happen dynamically on the web page.
3. Interact with temporary storage and local storage (cookies, local storage)
4. Sends request to server and retrieve data from it.

Client-side scripting generally refers to the class of computer programs on the web that are executed client-side, by the user's web browser, instead of server-side (on the web server).

This type of computer programming is an important part of the Dynamic HTML (DHTML) concept, enabling web pages to be scripted; that is, to have different and changing content depending on user input, environmental conditions (such as the time of day), or other variables.

Client-side scripts are often embedded within an HTML or XHTML document. Client-side scripts may also contain instructions for the browser to follow in response to certain user actions, (e.g., clicking a button). Often, these instructions can be followed without further communication with the server.

Client-side scripts do not require additional software on the server however, they do require that the user's web browser understands the scripting language in which they are written. It is therefore impractical for an author to write scripts in a language that is not supported by popular web browsers.

Due to security restrictions, client-side scripts may not be allowed to access the user's computer beyond the web browser application. Techniques like ActiveX controls can be used to sidestep this restriction.

Some of the client side languages are

1. Java script (essentially the client side scripting languages)
2. Ajax (used for implementing java script & XML without effecting the web page itself)
3. HTML &
4. CSS

Server side coding:

Server-side programming is the general name for the kinds of programs which are run on the Server.

With the help of server side programming or coding we can do the following:

1. Process user input.
2. Display pages.
3. Structure web applications.
4. Interact with permanent storage (SQL, files).

Server-side scripting is a technique used in website design which involves embedding scripts in an HTML source code which results in a user's (client's) request to the server website being handled by a script running server-side before the server responds to the client's request.

Server-side scripting differs from client-side scripting where embedded scripts, such as JavaScript, are run client-side in the web browser.

Server-side scripts require that their language's interpreter be installed on the server, and produce the same output regardless of the client's browser, operating system, or other system details.

Some of the server side languages are

1. JSP
2. PHP
3. ASP

ANATOMY OF PHP:

The PHP code is enclosed in special start and end processing instructions <?php and ?> that allow you to jump into and out of "PHP mode."

The skeleton of PHP is given as follows:

```
<?php
        //Statements
?>
```

The above shows the opening (<?php) and closing (?>) tag elements used to embed PHP functions and code in a HTML document, i.e. web page.

There are a number of ways to embed PHP within a page, but this is the most "portable," in that it works on nearly every web server—as long as the server supports PHP (typically a document's filename also needs to end with the extension .php, so the server recognizes it as a PHP document).

Anything within this tag is parsed and handled by the PHP interpreter, which runs on the web server (the interpreter is the PHP engine that figures out what the various functions and code do, and returns their output).

Everything outside of a pair of opening and closing tags is ignored by the PHP parser which allows PHP files to have mixed content. This allows PHP to be embedded in HTML documents, for example to create templates.

```
<p> This is going to be ignored by PHP and displayed by the browser. </p>
<?php
echo 'While this is going to be parsed.';
?>
<p>This will also be ignored by PHP and displayed by the browser.</p>
```

This works as expected, because when the PHP interpreter hits the ?> closing tags, it simply starts outputting whatever it finds (except for an immediately following newline) until it hits another opening tag unless in the middle of a conditional statement in which case the interpreter will determine the outcome of the conditional before making a decision of what which to skip over. See the next example.

```
<?php if ($expression == true): ?>
  This will show if the expression is true.
<?php else: ?>
```

Otherwise this will show.

<?php endif; ?>

In this example PHP will skip the blocks where the condition is not met, even though they are outside of the PHP open/close tags, PHP skips them according to the condition since the PHP interpreter will jump over blocks contained within a condition what is not met.

Note: Recursive acronym for PHP is *Hypertext Preprocessor.*