

# Subject: Web Development using PHP

Submitted by: Sh. Vipul Pant, Lect. Comp. Engg., Govt. Polytechnic for Women,  
Sirsa

## What is HTML?

---

*HTML is the language of the internet. It's what web pages are written in. HTML stands for "hypertext mark-up language".* HTML and XHTML are the languages used to construct Web pages. They are really the same language, except that XHTML is more formal. A good analogy to understand these is that they are basically like the difference between using slang English and proper English. Slang English is like HTML, whereas XHTML is the more proper, structured version of the language.

In the future, it is likely that Web browsers will expect that your Web pages are designed with the proper grammar and not alternative versions of that language.

We will be using HTML5 throughout this course. HTML5 is the new HTML standard. However, it is still being developed and not all browsers support the new features consistently. For the purposes of this class, and as beginners, we shouldn't bump into too many of these variations.

### HTML5 will:

- Provide better error handling
- Provide new elements and attributes
- Allow your code to be device independent
- Have a **much** simpler doctype
- Reduce the need for plugins - like Flash

### Do You Need to Know HTML to Create a Website?

- No - there are many website-building programs on the market that don't require knowledge of HTML at all. You tell the program what you want, and the program creates the HTML for you. This is one of the nice features of Dreamweaver. You can click a few buttons to make some formatting choices, and all the coding will be done for you.
- However, it is definitely to your benefit to understand how HTML works, so you can take full advantage of everything possible in web design. And, no matter how good the program is, there will be times where you will need to "tweak the code" to get it to do

exactly what you want to do. Throughout our class, we will be examining the code of our pages so it is essential that you can at least recognize HTML.

## What Do You Need to Write HTML?

- Technically, you just need a browser and a text-editing program. It's best to avoid word processing programs when writing HTML because they will often add additional "stuff" to your code. If you are working on a PC, Notepad or Notepad++ work well and are most likely already on your computer. If you are working on a MAC, TextWrangler is a program that works well.

When you type a web address into your browser's address bar, you are asking for a server to show you a web page. For example, if you type mcmenamins.com into your browser, the server must decide which page from the McMenamins directory it should display. By default, servers are typically configured to display the file "index.html" (or "index.htm" or "index.php", etc.). This means that the home page or main HTML file for any directory should be named "index.html" (without the quotes, of course!)

You will be creating several sites this term. Some will be set up with home pages, and others will simply be stand alone files that have a specific filename other than index.html. Be sure to follow the instructions for each assignment and always name your files as instructed.

## Tags

In HTML you work with tags, which are identified with angle brackets <>. Each tag has an opener and a closer. For example, if you want to format a paragraph, you use a <p> tag at the start of the new paragraph and a </p> tag at the end of the paragraph. Notice, the closing tag is the same as the opening tag with the addition of the forward slash /.

The basic structure of an HTML document includes tags, which surround content and apply meaning to it. ALL HTML tags should be closed. Although older versions of HTML lazily allowed some tags not to be closed, latest standards require all tags to be closed. This is a good habit to get into anyway.

**<p>**This is a sentence formatted with the HTML paragraph tags.**</p>**

All HTML5 tags have an opening tag and a closing tag which are indicated with brackets <>, such as <html> and need to have a closing tag, such as </html>. They indicate where things start and end on the code. The first tag we see is the **<html> tag** which kicks things off and tells the browser that everything between that and the **</html> closing tag** is an HTML document. The stuff between **<body>** and **</body>** is the main content of the document that will appear in the browser window.

All you need to remember is that all tags must be closed and most (those with content between them) are in the format of opening tag → content → closing tag.

## EMPTY TAGS

Not all tags have closing tags like this (<html></html>). Some tags, which do not wrap around content will close themselves or is called empty tags. The horizontal rule tag for example, looks like this : <hr />. Empty tags are tags that does not have a closing tag </ >, they are the only exception of the tag rules. There are 5 empty tags that you should at least know:

- <br /> --- break tag. If you hit shift-enter, it will create a <br /> tag for single line. If you hit enter, it will create a <p> tag for double-space line.
- <img /> --- image tag.
- <link /> --- used to link to an external stylesheet file.
- <hr /> --- horizontal rule tag.
- <meta /> --- used to display information about the webpage. It can contain what language or description or keyword about the webpage for the search engine.

## ATTRIBUTES

Some tags can have attributes, which are extra bits of information that appear inside the opening tag, separated by a space after the tag. Attributes usually followed by value, which is always inside quotation marks. They may look like this: <opening-tag attribute="value">Element</closing-tag>.

Example of HTML code: <a href="http://www.pcc.edu">PCC Home</a>

That code is described as the anchor tag <a> followed by the attribute -- href, then the value inside the quote -- http://www.pcc.edu. PCC Home is the element, what actually shows up on the browser. Don't forget to close the tag with </a>.

## ELEMENTS

Elements are not tags, but represented by tags in the code as a presentation on the web page.

For example: <title>Calisthenics 1 | Your Name</title>

Elements of the code above would be: Calisthenics 1 | Your Name, everything that is in between the opening and closing tags.

Since this is a class focused on using Dreamweaver to create web pages, we will not be spending a lot of time learning how to hand-code websites. We'll leave that for CAS 206 (which you should definitely take next!). However, there are certain tags that you **NEED TO KNOW** now - or at least be able to recognize them when looking at the code of your web page.

## HTML vs. XHTML

---

The main differences between XHTML and HTML are that in XHTML (*not necessarily in this order*):

1. Tags must be closed. If you start with a <p> tag, then at the end of that paragraph there should be a </p> tag.
2. Tags must be properly nested, such as when used in lists or inline style.
3. Tags and attribute names must be in lower case letters.
4. All attribute values must be in quotes.
5. A Doctype declaration should appear in the first line to clarify which version of the markup language you are using.
6. Empty tags like <hr /> and <br /> should contain a slash at the end.

Basic Web page elements normally consist of things shown below. The **mandatory minimum tags (in Bold)** are what you must include in an XHTML Web page.

**<html>** --- marks the beginning of the Web page

**<head>** --- contains elements that are not part of the main Web page, such as title and meta elements

<title> --- specifies text that appears in the title bar of the Web browser opening the page  
</title>

<meta http-equiv="Content-Type" content="text/html; charset="utf-8" /> --- contains information about the page and keywords to be used in the search engine

<link href="assets/whatever.css" rel="stylesheet" type="text/css" /> --- link to an external CSS file

<style type="text/css"> --- contains the embedded (internal) stylesheet  
p { color: #00f; }  
</style>

<script src="whatever.js"> --- normally link to a javascript file or contain javascript itself  
</script>

**</head>**

**<body>** --- includes contents that are visible in the main window of a Web browser

**<h1></h1>** --- represents the highest-level heading on the page. Headings go from largest (h1) to smallest (h6)

**<p></p>** --- marks a paragraph of text

**<strong></strong>** --- bolds text

**<em></em>** --- italicizes text

**<br />** --- inserts a line break

**<ul></ul>** --- Creates an unordered (bulleted) list

**<ol></ol>** --- Creates an ordered (numbered) list

**<li></li>** --- Surrounds a list item in either an ordered list or an unordered list

**<a href="URL"></a>** --- Creates a hyperlink

**<img></img>** --- Surrounds a file location where an image file is located - and displays the image!

**</body>** --- marks the end of the content

**</html>** --- marks the end of the Web page

When you start a new Web page in Dreamweaver, it gives you these tags along with a Doctype, **<meta>** tag and **<title>** tag.

While not absolutely required, the **<title>** tag should be embedded within the head section and is important to most Web designers.

## **Common HTML Tags you should know about**

---

- Div **<div> </div>** tag --> divides a page into a series of blocks.
- Paragraph **<p> </p>** tag --> creates a double-space break on a page.
- Break **<br />** tag --> forces a single-space break on a page.
- Nonbreaking space **&nbsp;**; --> insert a space that will be displayed by the browser. Often used as a temporary text placeholder.
- Blockquote **<blockquote> </blockquote>** tag --> indents text from both left and right margins, and can be nested for deeper indents.

- Ordered list `<ol> <li>list item</li> </ol>` --> creates a list of numbered items.
- Unordered list `<ul> <li>list item</li> </ul>` --> creates a list of bulleted items.
- Strong `<strong> </strong>` tag --> replacing the `<b>` tag or bold style to text.
- Emphasis `<em> </em>` tag --> replacing the `<i>` tag or italic style to text.

## 1. Absolutely Essential

**< !DOCTYPE...> The DOCTYPE preprocessor information (needed for XHTML) and**

**<html > </html> ..... The HTML tag**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
```

```
<!DOCTYPE
<html lang='en'>
<meta charset="UTF-8"> AFTER the head tag
```

html>

Copy one of the two above into the start of your page.  
Either way your document must end with **</html>**

Everything in between the `<html>` and `</html>` is interpreted as (X)HTML.

As you saw just above, in HTML5 you must specify the lang attribute, but in XHTML it is optional.

**<head> </head> ..... The head tag**

Again, opening and closing tags.

The header contains the title and will often contain your JavaScript code.

Often the header also contains meta-tags (keywords about the content of your

page to make it easier for search engines to find it.)

Any text between the tags will be in **bold** face.

There will be a blank line after your headings.

Heading sizes go from **<h1> </h1>** (biggest) down to **<h6> </h6>** (smallest).

## **<\title> </title>** ..... **The title tag**

Opening and closing tags.

The title is what is displayed at the bottom of your browser.

It should be informative.

Do not add spaces between the tags and the title:

```
<title>The right way to make a title </title>
```

```
<title> The wrong way to make a title </title>
```

## **<body> </body>** ..... **The body tag**

The body has everything that's not in the header.

It comes after the header, so that by the time the body is executed anything in the header has been read.

## **<!-- -->** **The Comment tag**

Anything between these tags is ignored by HTML.

This is where you put important information to document the code :

- Your name
- The date you wrote this code and the date of any subsequent revisions
- References - This code from such and such a book, page ....etc.

You will also use to enclose JavaScript code, so that HTML doesn't try to execute it.

## 2. Lining Up Text

`<p> </p>` ..... **Paragraph tags**

These mark the beginning and end of a paragraph.

Each paragraph will automatically start on a new line, with one blank line inserted after the last paragraph.

Of course, these tags come as an opening and closing pair.

`<br />` ..... **Line break tag**

This inserts a line feed (start new line).

There is no closing tag required in HTML, but the closing slash is needed in XHTML.

### **Alignment:**

The following have been deprecated in HTML5 and XHTML5, although they are still available in HTML4.01 and XHTML1.

`<center> </center>` ..... **Center alignment tag**

You may also use ALIGN to align a heading or paragraph:

```
<h1 align="center">Here is my centered heading</h1>
```

Alignment ends with the heading. Note quotes around "center". Other blocks (paragraphs, etc.) also allowed you to set the alignment.

But why use something which is not available in HTML5 when there is a perfectly good way to align items which works in all the versions of HTML and XHTML...

```
<p style="text-align:right">
```

Now comes a long and boring paragraph, right aligned.



`</p >`

`<div style="text-align: center">` Everything in here is centered until you come to...`</div>`

This is useful to center several paragraphs, heading, etc. at once.

Note: The default is left aligned for everything except headings, where the default is center.

text-align: may be followed by left, right or center.

In addition to controlling layout, a common use of text-align is to right-align a column of numbers.

`<blockquote>` `</blockquote>` ..... **Blockquotes**

For long quotes. The quote will be indented or italicized or otherwise set off.

`<pre>` `</pre>` .... **Preformatted text**

Everything in between will appear exactly as you typed it - indenting, paragraphs, etc. Useful for quoted material, poetry, etc.

`<hr />` ..... **Horizontal Rule**

This draws a line across your page.

You may specify the length as a percent of the page :

`<hr width="70%" />`

or a certain number of pixels, with or without an alignment:

`<hr width="100" align="left" />`

You may also specify the height (in pixels) by using the **SIZE** attribute, and make it solid color, or any other color (see next section)..

`<hr width="60%" size="6" noshade />`

You will probably collect some fancy horizontal rules for your pages

### 3. Colors and Fonts

`<b> </b>` ..... The bold face tag

`<i> </i>` ..... The italics tag

`<sup> </sup>`..... The superscripts tag

`<sub> </sub>`..... The subscripts tag

The following is no longer available in HTML5:

`<u> </u>` ..... The underline tag

See below for how to do this with CSS.

Using CSS these would be accomplished with:

`<span style="font-weight: bold"> ... </span>`

`<span style="font-style: italic"> ... </span>`

`<span style="text-decoration: underline"> ...</span>`

`<span style="vertical-align: super"> ... <span>`

`<span style="vertical-align: sub"> ... </span>`

In general, it is better to use `<strong> ... </strong>` than `<b> ...</b>`, and it is better to use `<em> ...</em>` than `<i> ... </i>`. ('em' stands for emphasis.) This is because readers for the visually impaired can render 'strong' and 'em' but not b(old) and i(talics).

### Font manipulation

Fonts have a font-face (e.g. Arial, Courier, etc.), a font-size, font-weight (e.g. bold), a font-style (e.g. italic).

Text attributes are used to set alignment (`text-align`), color (`text-color`) and decoration (`text-decoration` can have the values `underline`, `overline`, `line-through` or `blink`).

To have a paragraph in bold red with the Arial font and in the font three times as large as usual you would write:

```
<p style="font-face:Arial; font-weight:bold; font-size:3em;text-color:red"> ...</p>
```

The `<font>` tage of XHTML1 and HTML4 is no longer available in (X)HTML5. Accordingly, you may no longer use code such as:

```
<font> </font>..... Font tags
```

These tags are used to specify a particular font - size, face, color in the body.

Size, face and color are the attributes (properties) you are specifying in the font tag.

When the font tag closes, those attributes end.

```
<font size="7">This is the biggest text available.</font >
```

```
<font size ="3">This is the default size for text .</font >
```

```
<font size ="1">This is the smallest text available.</font >
```

```
<font size ="+1">Increases size by 1 unit</font >
```

Note: For headlines it is better (more reliable) to use `h1`, `h2`, etc.

You may also specify the typeface - but the face must be available on the user's computer.

```
<font face="helvetica">This is in Helvetica.</font>
```

Note: Not all browsers support this, and different browsers/versions may have different faces available, or different names for the same face (e.g. Times, Times Roman, Times New Roman.)

```
face="Times, times, Times Roman, times roman, Times New Roman, times new roman"
```

will look for these 6 faces (in that order), and then go to the default face.

Using CSS, font-size is changed with

`<span style="font-size: value">...</span>` where value may be absolute - e.g. 10pt, or relative to the previous - e.g. 120%, or specified with words such as xx-small, thru xx-large. For details, see the CSS notes or <http://www.htmlhelp.com/reference/css/font/font-size.html>

Using CSS, font families are specified with

`<span style="font-family: courier, Times, serif"> ... </span>`

NOTE: These style instructions can also go in heading or paragraph tags.

You may also combine these: `<p style="font: bold italic 12pt arial">..</p>`

## Colors

Finally, you may specify colors. You should always try to use browser-safe colors.

Colors are described by a set of three hexadecimal numbers. Each of the numbers is of the form hh.

Since there are three such numbers, the whole thing looks like hhhhhh.

Each of the h's is 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E, or F.

The three numbers specify the level of the Red, Green and Blue lights which make up the whole color.

Here are some common browser-safe colors:

Red #FF0000

Green #00FF00

Blue #0000FF

White #000000

Black #FFFFFF

The following list of colors is in the transitional but not the strict DTD of XHTML1 and is also available for styling with CSS (hence in HTML5).

There are also 16 widely known color names with their RGB values:

Black = #000000    Green = #008000  
Silver = #C0C0C0    Lime = #00FF00  
Gray = #808080    Olive = #808000  
White = #FFFFFF    Yellow = #FFFF00  
Maroon = #800000    Navy = #000080  
Red = #FF0000    Blue = #0000FF  
Purple = #800080    Teal = #008080  
Fuchsia = #FF00FF    Aqua = #00FFFF

If you wish your text to be blue then you enter:

```
<span style="text-color:#0000FF">Here is my blue text. </span>
```

The # sign alerts HTML that a hexadecimal number is following.

HTML (Netscape, Firefox and Internet Explorer and probably the other browsers) also recognizes a few color names:

**Black, White, Green, Maroon, Olive, Navy, Purple, Gray,  
Red, Yellow, Blue, Teal, Lime, Aqua, Fuchsia, Silver**

If you wish the background of your page to be black (not recommended) and all

your text to be white, then set the background with the body selector and use the background-color property and the text-color property in your style sheet.

You may no longer say

```
<body bgcolor="#FFFFFF" text="#000000">
Your body goes here
</body>
```

As all attributes of the body tag have been removed in HTML5.

#### 4. Lists

**<ul> </ul>..... Unordered List tag** (Unordered means not numbered).

The list is indented, and you may nest lists to get levels of indentation.

If the list is not bulleted then end each line with a **<br />**.

```
<ul>
  My first item <br />
  My second item< br />
  My third item< br />
  My last item
</ul>
```

**<li>..... List Item tag**

If you want your list to have bullets, put **<li>** in front of each item.

The line feed is inserted automatically before each **<li>**, so omit the **<br />**'s.

```
<ul>
  <li>My first item </li>
  <li>My second item</li>
  <li>My third item</li>
  <li>My last item</li>
</ul>
```

It is also possible to style the bullets in a list using

```
<ul style="list-style-type:none"> and the <li>, </li>
```

The value of none in list-style-type will give no bullets. Other possible values are disc, circle (the default) and square.

**<ol> </ol>... Ordered List tag (Numbered lists)**

Ordered lists are numbered sequentially.

Put an <li> before each item. The numbers and new lines are automatic.

Ordered lists may be nested, and you may mix ordered and ordered lists.

```
<ol>
<li>My first item</li>
<li>My second item</li>
<li>My third item</li>
<li>My last item</li>
</ol>
```

You may also specify how an ordered list is numbered/lettered using list-style-type.

For example,

```
<ol style="list-style-type:upper-alpha">
```

Will produce a list with items enumerated by A, B, C etc.

## 5. Links

### Absolute Links or Links to Other Pages

```
<a href="http://the_URL">Words to Underline</a>     The anchor tag - absolute
```

The text in between the two tags is underlined. When the user clicks on it the browser transfers to the URL in the first tag.

```
<a href="http://www.simmons.edu/~menzin">My Favorite Professor</a>
```

This example (above) is an absolute reference.

Notice that it gives both the protocol (HTTP ---- as opposed to FTP etc.) and the complete address.

Notice that the complete address is enclosed in quotation marks.

There is a convention that when a path name is listed (as above) without a file name at the end, then the browser will look for a file called **index.htm** or **index.html**. So your opening page should be named index.

There is also a convention that user directories (those that start ~username) will have all their public files in a directory called public\_html.

In other words, when a viewer clicks on the text in the example, her browser will actually get the file `www.simmons.edu/~menzin/public_html/index.htm`

In this case (the absolute URL) the URL completely defines where the browser is to go.

### Links to Places on the Same Page

`<a href ="#NamedSpot">Words to Underline to go up or down the page</a>`

`<a name ="NamedSpot" id="NamedSpot">Where link will go</a>`

**The anchor tag – same page (using the NAME attribute)**

In order to link somewhere else on the same page you need two anchor tags –

`<a name="ShortNameForTheSpot">Text to link to</a>`

defines a name for the place you wish to go to.

`<a href="#ShortNameForTheSpot">Text to click on to go there</a>`

does the actual linking.

Notice that both the **a name=** tag and the **a href=** tag have the address in quotation marks.

**In XHTML1 and in HTML4 you did not need the id= part, but beginning in HTML5 you need the id= and further if there is both a name (for legacy browsers) and an id then they have the same value.**

Notice the use of # inside the anchor where the linking is done ---this alerts the browser to look for a named place, not an absolute or (see below) relative reference.



Your link may go either up or down the page. See the links8a.html and links8b.html examples.

You may also combine links to other pages and links to named spots on those other pages. For example, let us suppose that you have built a page at with the URL

SomeComputer/MyBook/Intro.html

And that somewhere in that file you have a named anchor

```
<a name="contents">Table of Contents</a>
```

Then, on some other page, if you wish to link to the Table of Contents you would code:

```
<a href="http://SomeComputer/MyBook/Intro.html#contents">MyBook's Table of Contents</a>
```

Notice that there is the usual anchor with an href (in quotes) but that the #namedSpot comes at the end of the URL.

### **Relative Links or Links to Other Pages on the Same Site**

```
<a href="OtherFileInSameDirectory.htm">Check Out My Other Pages</a>
```

In this case you will link to a different file (one named OtherFileInSameDirectory.htm).

Relative links allow you to keep all related files in the same directory or folder. If you decide someday to move the whole folder to another computer or another spot on that computer,

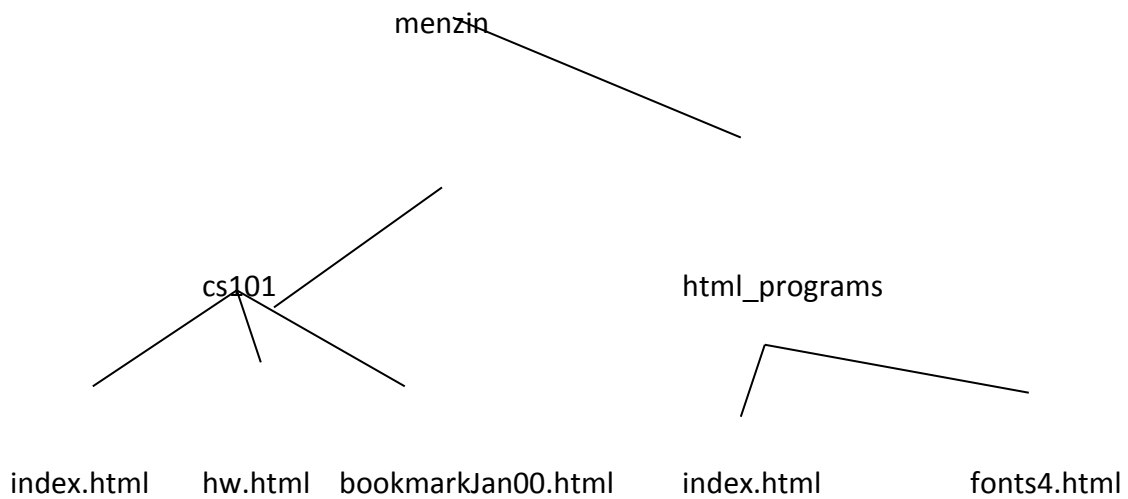
then the relative hrefs will still work, but absolute ones will need to be re-typed.

As usual, there are no spaces in URL file names, and file names are case-sensitive.

Relative references may be combined with named anchors, too, as above.

It is possible to do a limited amount of navigation in a directory using relative URLs.

Suppose that I have a directory (folder) named menzin and in it I have sub-directories named cs101 and html\_programs. Further, suppose that my html\_programs directory has a file called fonts4.h



In `html_programs/index.html`, a link to the `fonts4` file is **`href="fonts4.htm"`**

In `cs101/index.html` a link to `hw.html` is **`href="hw.html"`**

To get to the `html_programs/fonts4.html` file from `cs101/hw.html`, I first need to go up to the `html_programs` directory, and then to the `fonts4` file.

The `../` means go up one level in the directory tree. So the link is

**`href="../html_programs/fonts4.html"`**

The `../` gets us from the `cs101` directory to the `menzin` directory.

From there we go to the `html_programs` directory, and in it to the `fonts4` file.

We will see this again with graphics links.

You may insert a link to your email with:

`<a href="mailto:menzin@simmons.edu">Or contact me by e-mail</a>`

## 6. Tables

In HTML tables are used for creating charts and tables, but are no longer recommended for controlling page layout. Traditionally, a table with two columns (which need not have the same width) is one way to create the familiar side-bar with links to other parts of a web site. Today, using CSS is the preferred way to achieve this result.

Position on the page/page layout may be controlled with CSS. This is discussed in the CSS notes. Controlling position with CSS works better for pages which may be 'read' in many formats (e.g. on hand-held devices), but has the disadvantage that an external style-sheet is not always downloaded from a web page (i.e. the layout is not saved). It is the preferred method for laying out pages.

### `<table>` `</table>`            **The Table Tag**

Every table begins and ends with these tags.

A table has rows (which run left to right) and columns (which go up and down, just as on a building).

A table is described by reading across the first row, then reading across the next row, etc.

All rows of a table are of the same width.

### `<tr>` `</tr>`            **The Table Row Tag**

`<tr>` marks the beginning of a row's description.

```
<table>
```

```
  <tr>
```

```
    The description of the entries in the first row goes here
```

```
  </tr>
```

```
  <tr>
```

```
    The description of the entries in the second row goes here
```

</tr>

<tr>

The description of the entries in the third row goes here

</tr>

</table>

Notice that I have indented the table rows. Table descriptions can get complex (you can even put a table inside another table!) and it is a good idea to do this!

**<th> </th>            The Table Header Tag**

**<td> </td>            The Table Data Item Tag**

Each entry in a table is either a header (which is in bold) or a data item. The beginning and end of each entry is surrounded by these tags.

**Beginning with HTML5, all attributes of tables (border, cellspacing, cellpadding, and width. etc.), table rows and table cells must be set through CSS.**

You may specify **width** in <table> or in each column.

For the whole table (specified in the table tag):

<table style="width:70%">.....</table>    The table takes up 70% of the page.

<table style="width:200px">.....</table>    The table is 200 pixels wide.

For a table column (specified in a table cell):

<th style="width:20%">...</th>    This column is 20% of the width of the table.

You may do this for some or all columns (once for each column, typically in the first row)

<th style="width:50px">...</th>    The column is 50 pixels wide.

You may specify **alignment within each cell or row**.

`<th style="text-align:left"> </th>`A **th** or **td** or **tr** may be aligned left or right or center.

`<td style="vertical-align:top"> </td>` A **th** or **td** or **tr** may be vertically aligned top, middle, bottom.

Or you may specify that all the cell elements be aligned a certain way by putting the table inside div tags:

```
<div style="text-align:center">
  <table>
    :
    :
  </table>
</div>
```

You may **align a table** for purposes of wrapping text.

`<table style="text-align:left">...</table>` Puts the table on the left side of the page, and the text to the right.

The only choices are left and right.

You may put a **caption** on the top or bottom (default) of a table:

```
<table>
  <caption style="text-align: top">Data for Our Fascinating Study</caption>
  <tr>
    :
    :
  </tr>
</table>
```

See the various tables pages for examples, and examples of coloring both all the background and individual cells.

`<table style="background-color="red">.....</table>`

An entire table with a red background

`<td style="background-color:blue">     </td>`     A blue cell

`<table style="border:5;border-color="green">.....</table>` For St. Patrick's Day.

To create space around your cell contents:

`<table style="cellpadding:5">`     **Cellpadding** is the space between the edge of the cell and its contents.

`<table style="cellspacing:5">`     **Cellspacing** is the space between cells.

Sometimes you want a cell to stretch across several columns (e.g. for a heading) or down several rows.

```
<tr style="text-align:center">
  <th>This is the first column.</th>
  <th colspan="3">This occupies the next 3 columns.</th>
  <th>This is the last column</th>
</tr>
```

If you are doing something complex, it is a good idea to make a simple sketch of it before you start coding. That way when you have a column or row span you will remember which cells have already been taken described.

Remember: If you have an empty cell and you want it to be colored, put a `<br />` in it.

## 7. Inserting Graphics:

**Please read the pages I e-mailed you about gif's and jpeg's and about large files.**

``     **The Image Tag**

Let us suppose you wish to insert a clip art file that is in the same directory as this html page, and that the file is named StopSign.gif At the place where you wish the image to go you code:

``Here is the text that goes next to it

You may refer to the file using absolute or relative addressing (as for links).

```
What a big stop sign!
```

Obviously, if you change the height and width to a different ratio than your original gif or jpeg you will distort the image (which you may choose to do.)

Inside the img tag you may **align** the image to go on the left (or right) of the accompanying text.

```

```

NOTE: As of HTML5 you are supposed to always set the border. While the border attribute may still be used inside the <img> tag, it is preferred to set the border with CSS, as above.

I have a long explanation that I want near the icon, which is to the left of the icon.

For simple images, I may align it top, middle or bottom with my line of text, by styling the vertical-align property.

Whenever you see this sign 

```

```

 you should stop.

The hspace attribute will place space between your text and your graphic.

```
Here goes lots of text
```

The 

```
<br style="clear:both" />
```

 will clear all alignments. You should be warned that the align tag does not always work the way you wish it to (especially when you have a lot of text to next to your image.) Using a table for layout is a more reliable way to control appearance of your page. See the CSS Notes for more information.

See [InsertingGraphics.html](#) for examples.

You may (of course!) include the image in an anchor tag:

```
<a href="http://web.simmons.edu/~menzin/cs101"><img="smiley.gif" />To the source!</a>
```

Finally

```
<body style="background-image:url("awfulStuff.gif")>
```

will cause the entire background of your page to be tiled with the gif you specified.

**NOTE:** You should always include the **alt** attribute `` to get a written description for visually impaired users (and those too impatient to wait for the image to load) and for search engines.



## Brief History of PHP Language

PHP (recursive acronym for "PHP: Hypertext Preprocessor") is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML

PHP was developed to specifically address needs of the web to provide dynamic content on websites

Unlike other development languages commonly used for dynamic content (Perl, C++), PHP was designed specifically with the web in mind (it had no other master, per se)

Because of this specific design, common web-based activities, such as the processing of forms data and the correct rendering of HTML content (both inside and outside of forms), are much easier than with adapted languages

Because of PHP's close relationship to HTML, PHP can be embedded inside an HTML-based document, unlike other languages that do not inherently understand HTML and therefore must treat HTML as text that has to be displayed inside the confines of the languages print statements. PHP can literally switch between PHP and HTML inside a single document, making it so large areas of pure HTML can be managed normally

PHP has been extended as a language to include a huge library of commonly-available procedures and classes (including database manipulation, mail management, secure connections, and graphics manipulation to mention just a few) that has made it extremely powerful in a variety of environments and disciplines

Although PHP was designed to be a web-based language to display content via a browser through a server, the power and the usefulness of the language has expanded its uses beyond just the web, and it now can be found in both local command-line interface (CLI) environments as well as local graphical interface (GUI) environments

---

## HTML Background

HTML (Hypertext Markup Language) was developed to address the need to easily display content via a web-browser

It is a "markup" language (unlike a typical programming language), in that its commands (tags) are designed to assist in the formatting and layout of textual data

It by definition is a "static" language, in that the content displayed using the standard HTML language will always look the same -- it will not change over time or by who accessed it

Due to this major limiting factor of the language in this modern world of dynamic, data-driven websites, a variety of extensions to HTML and related programming languages have been developed:

- Javascript
- Microsoft's ASP (Active Server Pages)
- Java Applets and Applications
- PHP
- others...

[Examples of HTML tags and pages](#)

---

## Common PHP Resources

---

### Basic PHP Concepts

PHP borrowed its primary syntax from C++ and C

Many of the programming techniques you've previously learned will work in PHP (assignments, comparisons, loops, procedures) with little to no syntax difference

There are, however, major changes in how data is manipulated in relationship to C/C++

C/C++ are type-specific languages, requiring the user to define a specific, singular

type for each variable that they use

PHP commonly assigns its variables "by value", meaning a variable takes on the value of the source variable (expression) and is assigned to the destination variable. A variable can therefore change its type "on the fly". Therefore variables are not declared (as they are in most type-specific languages like C++)

PHP is an interpreted language, in that the PHP interpreter program reads the PHP source code, translates the code and executes it at the same time. With C++ on the other hand, the C++ compiler translates your C++ code into a binary executable, eliminating the translation of the source each time the code executes

Initially this interpreted nature of PHP sounds like a disadvantage; on the contrary, the interpreted nature of PHP provides some very interesting and useful programming techniques that are not possible in compiled languages

---

## Using PHP in a Webpage

PHP source code is embedded in an HTML-based document, and is identified by special delimiting tags,

```
<?php content ?>
```

similar to Javascript and Java applets.

```
<H2>My Webpage</H2>
This is my webpage.

<?php
  echo "This is written in PHP.\n";
?>
```

How this will appear in a browser:

```
My Webpage

This is my webpage. This is written in PHP.
```

You can switch between HTML and PHP as many times as you like within a document:

```
HTML content

<?php PHP content ?>

HTML content

<?php PHP content ?>

HTML content
```

---

## Webpage Setup Using PHP

Two Approaches:

- Using .php filename extension on source file.
- Including PHP script call inside source file along with naming the source file with .cgi extension and making source file executable (UNIX environment).

---

Approach One:

Name your source file with a .php extension:

sample.php  
index.php

(This requires proper setup on server so it understands what to do with files with this extension.)

---

Approach Two:

Including call to PHP script inside source file:

Source file: sample.cgi

```
#!/usr/local/bin/php

<H2>My Webpage</H2>
This is my webpage.

<?php
    echo "This is written in PHP.\n";
?>
```

and making source file executable:

```
% chmod +x sample.cgi
```

---

The major difference between the two approaches is how the files are accessed by the webserver:

When using the .php extension, the script runs as the standard webserver user (commonly the user-id nobody or www-data). Therefore if the script attempts to access/create files, the programmer needs to make certain that the file permissions are set correctly.

When using the .cgi extension, the script runs as the owner of the script (you), so any files created/changed by the script will automatically be accessible by you.

Approach Two is the approach used in your department UNIX account on the students.csci.unt.edu server.

---

## Variables and Types in PHP

Although variables are *not* declared to be type-specific in PHP, PHP still has a common set of data types:

boolean integer float string array object resource NULL

Determining the current type of a variable:

A series of type-testing functions exist to determine the current type of variables:

`gettype(varname)`

returns type name, such as 'string'

`is_int()` `is_integer()` `is_long()` `is_null()` `is_numeric()`

`is_object()` `is_real()` `is_string()` `is_scalar()` `is_bool()`

`empty()` `isset()`

[PHP type comparison tables](#)

Special debugging / variable-display functions:

`print_r()` `var_dump()` `var_export()`

---

Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is case-sensitive.

A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed as:

```
'[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'
```

To assign values or expressions to variables, the standard assignment equal-sign operator ( = ) is used

```
$var = "Bob";  
$Var = "Joe"; // different variable  
$Long_variable_numeric_name = 47;  
  
$4site = 'not yet'; // invalid; starts with a number  
$_4site = 'not yet'; // valid; starts with an underscore  
$täyte = 'mansikka'; // valid; 'ä' is (Extended) ASCII 228
```

---

## Special Relationship Between Strings and Variables

String constants can be defined in one of three common ways:

Inside Single Quotes: 'One type of string'

Inside Double Quotes: "Another type of string"

Using special "heredoc" syntax (*discussed later*)

---

Data inside Single-quoted strings are taken *literally*; ie., everything is treated exactly as it is typed

Data inside Double-quoted strings are treated in a special way in relationship to variable references and other standard formatting characters:

If a variable is referenced inside a double-quoted string, its value is automatically substituted.

"Escaped" characters are interpreted: [Table of "Escaped" characters](#)

```
$var1 = 'This is a test';  
$var2 = 27;  
  
$var3 = "$var1 $var2\n"; // "This is a test 27  
"  
  
$var4 = '$var1 $var2\n'; // '$var1 $var2\n'
```

If variable names can be clearly delineated in the double-quoted string syntax (it is not obvious where the variable name ends and literal text following the variable name begins), a variable name can be surrounded with curly-braces { }, or separated from the rest of the text using *concatenation* (the period . operator):

```
$var1 = 'ABC';  
$var2 = "Value is $var1xyz"; // "Value is "  
$var3 = "Value is {$var1}xyz"; // "Value is ABCxyz"  
$var4 = 'Value is ' . $var1 . 'xyz'; // "Value is ABCxyz"
```

---

All string constants (single- or double-quoted) can be automatically continued onto multiple lines:

```
$var1 = 'This is a long variable
```

```
that is continued onto multiple lines.';

$var2 = "This is a long variable with another
variables defined inside it: $var1\n";
```

---

## Displaying Values

Values can be displayed (output) using three methods: echo, print(), and printf()

---

```
echo string arg1 [, string argn...];
```

echo outputs all values following it. It is not actually a function (it is a language construct) so you are not required to use parentheses with it.

```
echo "This is a test\n";

$var1 = 'Test string';
$var2 = 75;

echo "The value of var1 is $var1\n";
echo "The value of var2 is $var2\n";
echo "Multiple variables displayed: $var1 $var2\n";
echo "This is a value that
is written on multiple lines,
including variable $var2 references.
";
echo 'This',$var2,'that'; // "This75that"
```

---

print() is in some ways similar to echo, although it can be used as a function and could be included in more complicated expressions

```
echo "This is a test\n";

$var1 = 'Test string';
```



```
$var2 = 75;

print "The value of var1 is $var1\n";
print ("The value of var2 is $var2\n");

$ret = print "Hello World"; // $ret will equal 1
```

### [Discussion of differences between echo and print](#)

---

printf() is one member of a family of string formatting functions. It is based on the syntax of the [sprintf\(\)](#) function.

```
$var1 = 123.456;
$var2 = 255;
$var3 = 'text';

printf("<pre>%d %05d %5.2f %'*-10s %o %b %x</pre>",
    $var1,$var2,$var1,$var3,$var2,$var2,$var2);
```

```
123 00255 123.46 text***** 377 11111111 ff
```

---

## PHP Conditional Statements

In many ways PHP's methods of handling conditional statements (if) is exactly the same as C/C++. All of the if-related logical operators are the same, although they've added a couple of more for convenience:

```
< > <= >= == != ! && ||
AND OR
```

With the addition of the word versions of AND and OR, conditional statements can now be written more like English:

```
if ($num1 < $num2 AND $num3 == $num4)
```

```
if ($a == 'Sample' OR $data < 200)
```

---

## PHP Arrays

An array in PHP is actually an ordered map.

A map is a type that maps values to keys. You can use it as a real array, or a list (vector), hashtable, dictionary, collection, stack, queue and probably more. Because you can have another PHP array as a value, you can also quite easily simulate trees.

An array's index (key) can simply be an integer value, which is equivalent to C++ arrays.

To reference an element in an array, you also use the same notation as in C++.

Elements are added dynamically -- when an index is specified, if it doesn't already exist, it will be added.

PHP Arrays also differ from C++ arrays in that each value can be a different type.

```
$num[4] = 256;  
$num[10] = 'some text';  
$num[20] = $count + 20;  
echo $num[10];  
  
echo $num[5]; // may produce error  
  
$num[] = -25; // same as $num[21]
```

A shorthand notation can be used to assign values to an array in a single statement using the `array()` function:

```
$elements = array (1,6,'text',-4,0.123,50+$count);  
// 0 1 2 3 4 5  
// note these are values, not indices
```

---

## Associative Arrays

PHP Arrays can use either integer or string indices. They can be mixed inside the same array. PHP does not maintain different typed arrays for integer or string indices; there is only one array type.

```
$num = 10;  
$elements['test'] = 23;  
$elements[5] = 'stuff';  
$elements[$num] = 'more stuff';
```

When using the array() function (and several other places in the language), the key/value element pair can be written using the special key => value notation.

```
$elements = array ( 4 => 'text', 'str' => 23);
```

---

## Accessing All Elements in an Associative Array

Since an Associative Array can have a mixture of index types, a normal for-loop will not work to access each position in an Associative Array. A special construct foreach exists to simplify this operation:

```
foreach (array_expression as $value)  
    statement  
  
foreach (array_expression as $key => $value)  
    statement
```

```
$A1 = array ('x','test',3,-16,'stuff',array(1,2,3));  
$A2 = array (10=>20, 'test'=>'data', 'counter'=>12);  
  
foreach ($A1 as $value) echo "$value ";  
echo "<br><br>\n";  
foreach ($A2 as $key => $value)  
    echo "$key => $value<br>\n";
```

```
x test 3 -16 stuff Array
```

```
10 => 20  
test => data  
counter => 12
```

---

### Determining the size of an Array

`sizeof(arrayname)` or `count(arrayname)`

```
$A1 = array ('x','test',3,-16,'stuff',array(1,2,3));  
$A2 = array (10=>20, 'test'=>'data', 'counter'=>12);  
  
echo sizeof($A1) . ' ' . sizeof($A2); // 6 3  
echo count($A1) . ' ' . count($A2); // 6 3
```

---

### Common Associative Arrays

<code>\$_POST</code>	fields from form tags
<code>\$_GET</code>	fields from URL arguments
<code>\$_SERVER</code>	common system-oriented information
<code>\$_COOKIES</code>	fields from browser cookies
<code>\$_SESSION</code>	fields for user authentication
<code>\$GLOBALS</code>	all global variables

---

### Working with Forms Data in PHP

Form fields and their values are stored in the PHP `$_POST[]` super-

global associative array.

Depending upon the current configuration of PHP on your server, all form fields may also be stored as individual global variables.

Because of this convention, you should maintain a *variable name* standard for the naming of your form fields.

```
<input type=text name="NameField" value="Tom Jones" />

<textarea name="InformationAndComments" rows=5 cols=60>
This is some data
</textarea>

-----

echo $_POST['NameField'] . "<br />\n";
echo $_POST['InformationAndComments'] . "<br />\n";
```

You should use caution when defining form field names that do not adhere to the standard PHP variable naming conventions. When you define fields in this fashion in your HTML, PHP will "attempt" to convert the field names into a compatible PHP variable name.

```
<input type=text name="Name Field" value="Tom Jones" />

<textarea name="Information &%^$@#/ Comments" rows=5
cols=60>
This is some data
</textarea>

-----

print_r($_POST);
```

```
Array
(
    [Name_Field] => Tom Jones
```

```
[Information_+%^$@#/_Comments] => This is some data
)
```

*NOTE the conversion of spaces into underscores*

Although using the global array references

```
$_POST['Information_+%^$@#/_Comments'] and
$GLOBALS['Information_+%^$@#/_Comments']
```

will work, an attempt to reference `$_Information_+%^$@#/_Comments` will result in a syntax error.

If you plan on using this type of complicated naming convention for form fields, you should not plan on referencing the fields as global variables. Most current configurations of PHP have this option turned off by default.

---

## Techniques for detecting Forms Submission

It is common that a programmer will design a forms-based webpage so that it consists of a pure HTML-based webpage, and a separate PHP-based script that processes the forms data.

When the forms-based page itself contains PHP-generated information, such as remembering field values from a previously submitted form, maintaining separate scripts becomes tedious.

It is very simple, however, to determine if a page is referenced via a URL reference or is called by a script. A variety of techniques can be used to do this.

```
$_SERVER['REQUEST_METHOD']
```

This variable returns either 'GET' or 'POST', indicating the method the page was referenced. To determine if a script is called by pressing a form submission button, this simple test could be used:

```
if ($_SERVER['REQUEST_METHOD'] == 'POST')
```

Another method would be to merely determine the size of the global `$_POST` array. If it has one or more indices, there was at least one form field passed, indicating the script had to be called from the posting of a form.

```
if (count($_POST) > 0)
```

---

This therefore creates a very simple model to combine the display of a form, and the processing of the submitted form, all within the same script file:

```
<?php
if (count($_POST) == 0) {
    // display the initial display of the form here
}
else {
    // process the submitted forms data here
}
?>
```

---

### **Schemes for Submission Button Naming and Access**

Forms can have any number of submission buttons, and therefore can cause different actions depending upon which button is actually pressed. There are two common techniques that can be used to easily identify which action you wish to perform based on the actual button pressed.

---

#### **Same Name, Different Values**

The first technique is to name each selection button the same

name, and then specify a different value. In the PHP code, you could test the value of the corresponding `$_POST` element, which will indicate which button was actually pressed.

Button 1	Button 2	Please, press me, w on't you?
----------	----------	-------------------------------

---

```
<input type=submit name=Dolt value="Button 1">
<input type=submit name=Dolt value="Button 2">
<input type=submit name=Dolt
    value="Please, press me, won't you?">

switch ($_POST['Dolt']) {
case 'Button 1' :
    // button 1 code
    break;
case 'Button 2' :
    // button 2 code
    break;
case 'Please, press me, won't you?' :
    // "Please, press me, won't you" code
    break;
} // end switch
```

The minor disadvantage of this approach is that since the submit-type field always uses the value as the text displayed in the button, this may require long comparisons since the text must match exactly.

---

### Different Names

The second approach is to choose a different, unique name for each selection button field. In your PHP code that processes the form, you would merely test for the presence of each of the submit fields; if one is present, it was the one that was pressed. Similar to checkboxes and radio buttons, submit buttons that aren't actually pressed send no data to the script, and therefore do not appear in the resultant `$_POST` array.

```
<input type=submit name=Button1 value="Button 1">
```



```
<input type=submit name=Button2 value="Button 2">
<input type=submit name=Button3
      value="Please, press me, won't you?">
```

---

```
if (isset($_POST['Button1'])) {
    // button 1 code
}
elseif (isset($_POST['Button2'])) {
    // button 2 code
}
elseif (isset($_POST['Button3'])) {
    // button 3 code
}
```

---

## Dealing with Multiple Selections

Multiple selection form fields pose an interesting problem based on how PHP processing form fields in general.

```
<form method=post action="showFields.cgi">
<select name="Options" size=5 multiple>
  <option>Option 1</option>
  <option>Option 2</option>
  <option>Option 3</option>
  <option>Option 4</option>
  <option>Option 5</option>
</select>
<input type=submit name=GO value=GO>
</form>
```

---

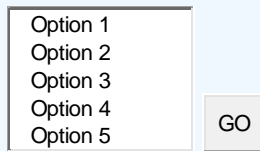
Since PHP stores all form fields and their values as indices in the `$_POST` array, it can't deal with multiple fields being sent with

exactly the same name.

PHP therefore uses a special array notation to represent multiple selections:

```
<form method=post action="showFields.cgi">
<select name="Options[]" size=5 multiple>

<option>Option 1</option>
<option>Option 2</option>
<option>Option 3</option>
<option>Option 4</option>
<option>Option 5</option>
</select>
<input type=submit name=GO value=GO>
</form>
```



---

```
$_POST = Array
(
  [Options] => Array
  (
    [0] => Option 2
    [1] => Option 3
  )

  [GO] => GO
)
```

This approach can also be used for other form fields as well. In addition, index values can be used rather than automatically generating a new, numerically indexed element.

```
<form method=post action="showFields.cgi">
<input name="f1[Fred]" value="fred">
<input name="f1[John]" value="john">
<input name="f1[Alias for John]" value="sam"><br />
```

```



```

fred	john	sam
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="submit" value="DO IT"/>		

---

```

$_POST = Array
(
    [f1] => Array
        (
            [Fred] => fred
            [John] => john
            [Alias for John] => sam
        )

    [GO] => DO IT
)

```

Multiple-dimensional arrays are also possible using this notation:

```

<form method=post action="showFields.cgi">


```

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="submit" value="DO IT"/>		

```
$_POST = Array
(
    [f1] => Array
        (
            [0] => Array
                (
                    [1] => set
                )
            [1] => Array
                (
                    [2] => set
                )
        )
    [GO] => DO IT
)
```

**PHP**  
**Certain Form Fields**

**Simplification**

**of**

```
<SELECT NAME="table2">
<OPTION Value=2>2</OPTION>
<OPTION Value=3>3</OPTION>
<OPTION Value=4>4</OPTION>
<OPTION Value=5>5</OPTION>
<OPTION Value=6>6</OPTION>
<OPTION Value=7>7</OPTION>
<OPTION Value=8>8</OPTION>
<OPTION Value=9>9</OPTION>
<OPTION Value=10>10</OPTION>
<OPTION Value=11>11</OPTION>
<OPTION Value=12>12</OPTION>
</SELECT>
<SELECT NAME="Table2">
<?php
```

```
for ($i=2; $i <= 12; $i++)
    echo " <OPTION Value=$i>$i</OPTION>\n";
?>
</SELECT>
```

---

```
2 <input type=radio name="table3" value="2"
checked>
3 <input type=radio name="table3" value="3">
4 <input type=radio name="table3" value="4">
5 <input type=radio name="table3" value="5">
6 <input type=radio name="table3" value="6">
7 <input type=radio name="table3" value="7">
8 <input type=radio name="table3" value="8">
9 <input type=radio name="table3" value="9">
10 <input type=radio name="table3" value="10">
11 <input type=radio name="table3" value="11">
12 <input type=radio name="table3" value="12">
<?php
$checked = 5;
for ($i=2; $i <= 12; $i++)
    echo "$i <input type=radio name=\"table3\"
value=$i\" .
        (($i == $checked) ? ' checked' : ") . ">\n";
?>
```

---

## URL Parameters - GET method

Additional data can be passed to a script via parameters indicated on the URL line:

<http://server/scriptname.cgi?parameters>

These parameters normally come in two possible formats:

- keyword=value pairs with multiple values separated with ampersands (&); to include spaces, substitute plus signs (+)

- simple character sequences with multiple values separated with plus signs (+)

```
scriptname.cgi?this=that&name=value+with+spaces
```

```
scriptname.cgi?value1+value2+value3
```

Depending upon which of these formats is used for the parameter data, different PHP super-global variables can be used:

`$_GET` used with name=value pairs

`$_GET` will be an associative array with the names being the indices

---

`$_SERVER` used with simple character sequences

`$_SERVER['argc']` contains the number of character sequences;

`$_SERVER['argv']` contains an array of the actual character sequence values

```
scriptname.cgi?this=that&name=value+with+spaces
```

```
$_GET = Array
(
    [this] => that
    [name] => value with spaces
)
```

---

```
scriptname.cgi?value1+value2+value3
```

```
$_SERVER['argc'] = 3

$_SERVER['argv'] = Array
(
    [0] => value1
    [1] => value2
    [2] => value3
)
```

---

### Combining POST data with URL arguments

Even with posting forms data, it is also possible to include URL arguments on the ACTION=field on the form. This data will be passed to the executing script just as with the GET method.

```
<form                                method=POST
action="script.cgi?value1+value2">
<input name=Field1 value="Field1 data">
<input type=submit name=Button value="Press
Me">
</form>
```

---

```
$_POST = Array
{
    [Field1] = Field1 data
    [Button] = Press Me
}

$_SERVER['argc'] = 2

$_SERVER['argv'] = Array
(
    [0] => value1
    [1] => value2
)
```

---

## Submitting Form Data with GET method

Although less-commonly used, the submit method for form data can also be GET rather than the normal POST.

```
<form method=GET action="script.cgi">
<input name=Field1 value="Field1 data">
<input type=submit name=Button value="Press
Me">
</form>
```

---

When script is called, the URL will appear as:  
script.cgi?Field1=Field1+data&Button=Press+Me

```
$_GET = Array
(
    [Field1] => Field1 data
    [Button] => Press Me
)
```

```
$_POST = Array
{
}
```

In earlier versions of servers, URL arguments were limited to ~100 characters, basically eliminating the practical use of the GET posting method, especially when TEXTAREA fields were involved. Today this limitation has virtually been eliminated, and it no longer is considered a limitation of the GET posting method.

However, since the resultant \$\_GET array when using the GET posting method has the same appearance as the \$\_POST array when using the POST posting method, the GET posting method is considered (by DonR) to be unnecessary.

Code example demonstrating the various ways a script can be called, along with various parameter-passing



techniques:

---

### URL encoding of Special Characters

Since the decoding of the URL must include special separating characters (for example, + for space, & for separating GET fields), what happens when you would like to use those characters (or other non-alphabetic characters) as data within the URL argument?

You have to use a special hexadecimal-based encoding notation to represent these special characters. Their general format is:

*%hexvalue*

Some examples are:

%2B	-	plus	sign
%26	-	ampersand	
%3D	-	equal sign	

```
script.cgi?parm=This+has+a+plussign+%2B

$_GET = Array
{
  [parm] = This has a plussign +
}
```

When a URL argument is produced inside a PHP script, the PHP function `urlencode()` should be used to properly encode any non-alphabetic characters found in the argument:

```
<?php

$data = '# @ $ +';
echo '<a href="script.cgi?parm=' . urlencode($data) .
```

```
">;
```

```
?>
```

---

```
<a href="script.cgi?parm=%23+@+%24+%2B">
```

---