HTML

Primer & Reference Guide

This PDF is produced with the assistance of the community, in the expectation of helping users to more expertly create HTML pages. It has been cross-checked with the W3C☞ DTD☞ & other pages & sites. The author is only too human, so do proof-check & report back any errors found so that it can improve.

Credits:
For this PDF, you can thank the work of *Alex Kemp*.
26 August 2013
http://www.modem-help.co.uk/☞

Contact via forums:
http://forums.modem-help.co.uk/☞

Download this PDF:
non-modem/zigzag/

Feedback:
HTML Reference PDF thread (Modem-Help forums)

Redistribution:
This PDF may be freely copied and / or re-distributed, on the understanding that the Copyright Notices & Trademark Acknowledgements on this page, plus the links on this & other pages, are not changed. An acknowledgement of source plus link back to Modem-Help would also be greatly appreciated.

Version History:

26 August 2013:	started
28 April 2014:	v1 released:- html 4.01 + css1

Contents:

Foreword

In August 2013 Carly & Martin—the guiding minds & hands behind the regeneration of the ZigZag fields in St. Anns—took everyone into (what used to be, now long abandoned) *The Beacon* public house & said *"If we want, this can be a community space"*. My thought was:- 'what could I do?'...

The previous Christmas I had given my grandchildren a cut-out from The Times (the London newspaper). It was an 'easy-howto-code-HTML'. To my surprise Micaela was enthusiastic; Mickey was 12 and had been into ballet since she was 3; she was the last person that I thought would want to know how to code HTML. I clearly needed to update my thinking.

Back at the *The Beacon* I asked the local kids:- would they be interested in learning how to code? It did seem that they were more interested in learning *"How to hack"*, but at least one adult was also enthusiastic, so I put my brains into action, and this PDF is the first result.

I could not find a comprehensive reference for HTML nor CSS so decided to produce my own. This first version is HTML4.01 + CSS1. From one point of view that is prehistoric but, as HTML5 isn't even ratified yet, it is also the most up-to-date HTML that we have got. CSS2,3 will follow in a later version.

I hope that you find it useful.

Alex Kemp
27 April 2014

Primer

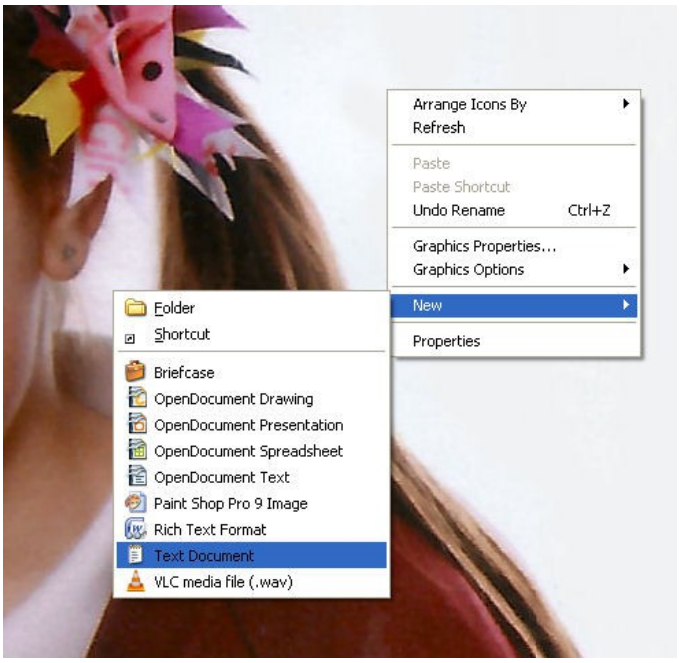
Create a Web-page

You will need:

1. A plain-text Editor
2. A web Browser.

Any text editor will do, and any browser, under any OS; you do NOT need to be connected to the Internet. The examples below are in TextPad + Firefox v23 under Windows XP.

Create an empty HTML page

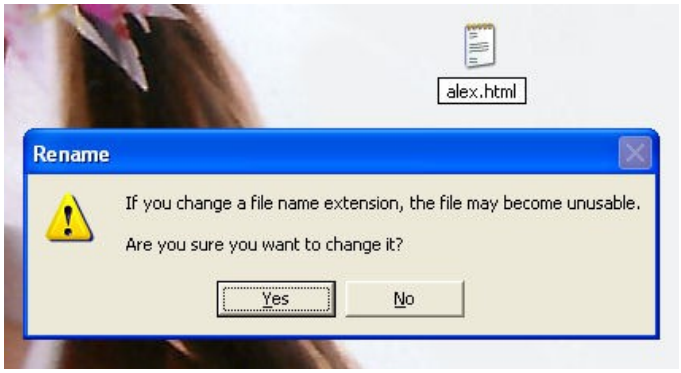


There are many ways to do this; the picture at right shows just one.

Start with a click from the right mouse-button on the Desktop (or in an Explorer window) under Windows XP.

Pressing <Enter> will give a new document called '`New Text Document.txt`'.

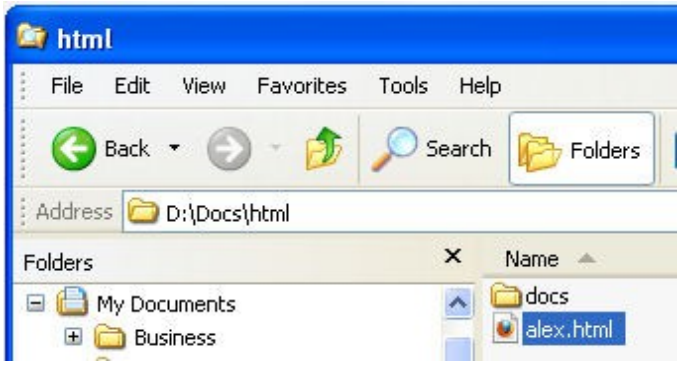
Now change the document name to: '`<your-name>.html`'.



Windows will now complain, because you have changed the document name from 'something.txt' to 'something.html'.

You are allowed to swear (but only under your breath if others are in the room).

Press <Yes>.



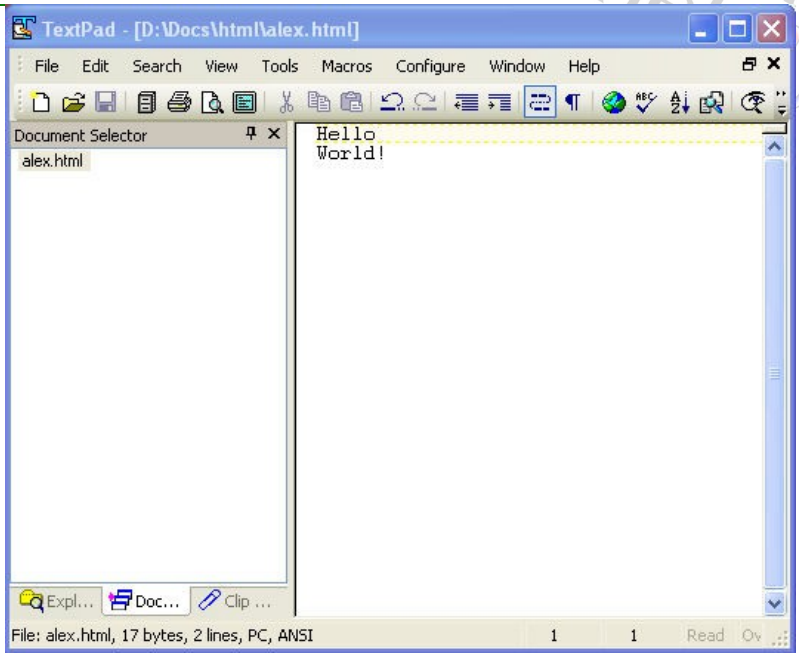
Here the new file has been put somewhere easy to find:

Open in a Text Editor

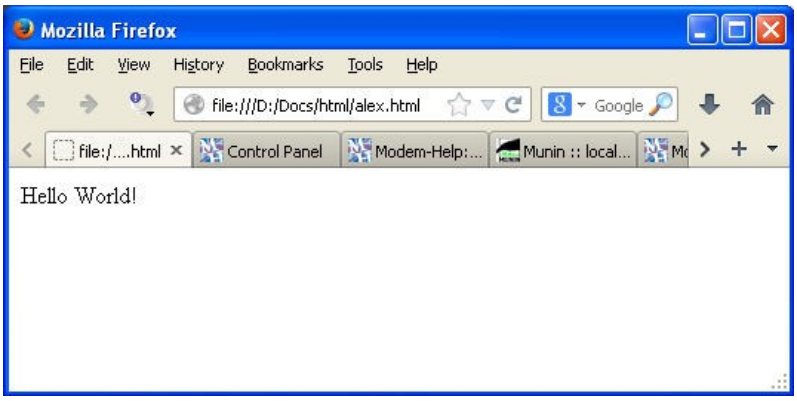
Type in the words as shown at right ('Hello World!') – notice that each word is on a separate line (you can also try putting more words in, and more than one space between each word)

Save the file, then open in a Web Browser.

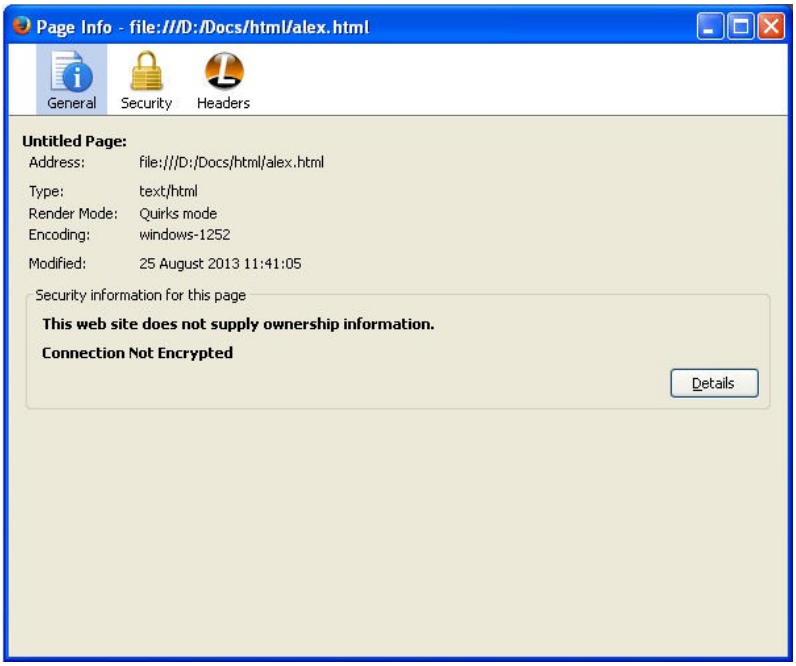
(TextPad has a 'View|In Web Browser' menu option, whilst Firefox has a 'File|Open File...' menu option)



Well done! That is your very first Web Page (that was easier than you thought).



We have some distance to go, yet. If you right-click in the Firefox window & choose *'View Page Info'* you will see the dialog as at right; the important line at this moment is *"Render Mode: Quirks mode"*. We do not want that in any situation, no sir, but it will be a little while before it changes.

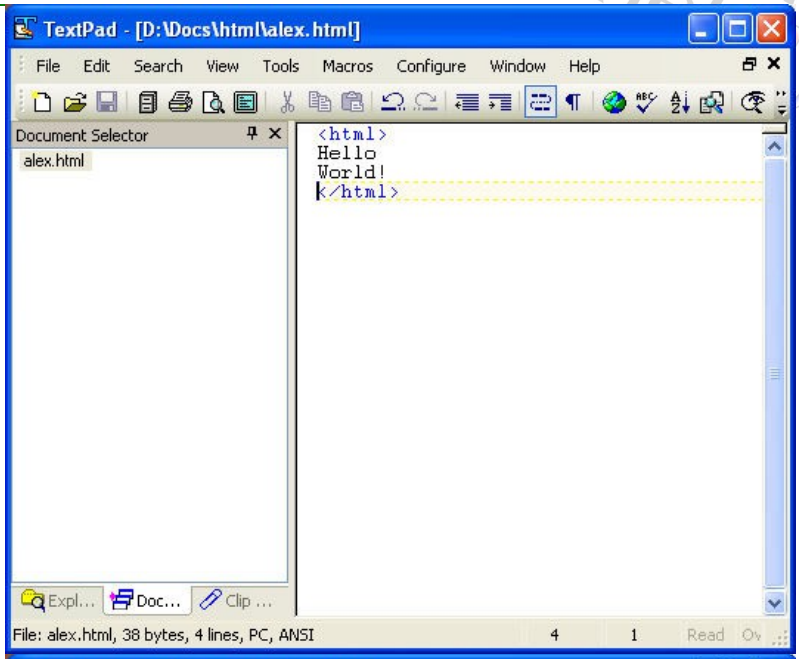


Add <html> Tags

The 'm' in 'html' stands for "markup", and as at right you should add a set of 'html' markup tags to your document.

These tags tell the browser that what follows is HTML (an 'xml' tag would indicate the XML language, etc.).

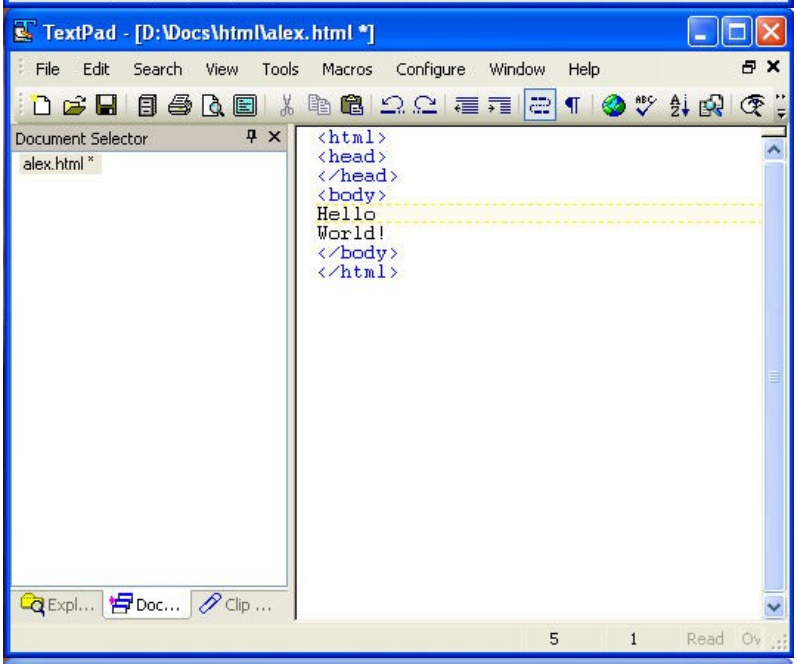
(you can save these changes & refresh the Browser, but there will be zero change at this point)



Add head, body tags

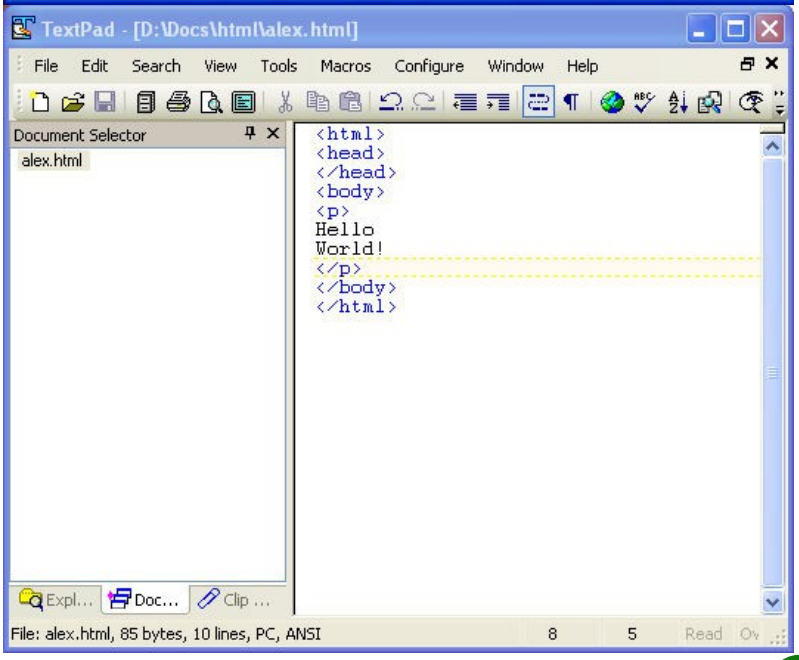
Each HTML document should contain both a *HEAD* plus a *BODY* section.

The `<head>` tags immediately follow the opening html tag, whilst the `<body>` tags surround the text entered earlier.



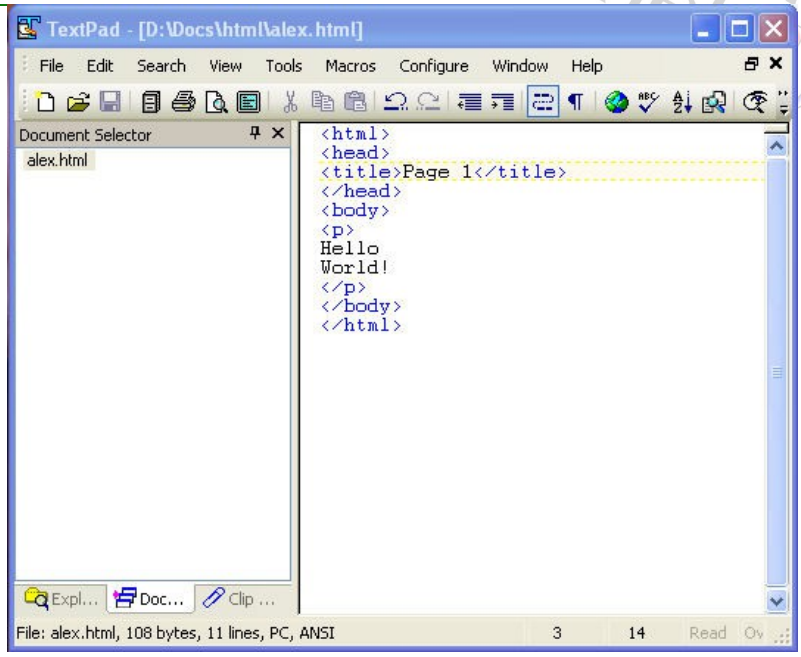
Add paragraph tags

These are placed to surround the body-text entered originally.



Add a Title

Although most of the sections within the HTML head are not normally directly seen, the *title* section is the major exception to this rule.



Once again, save the text file & refresh the Browser window:

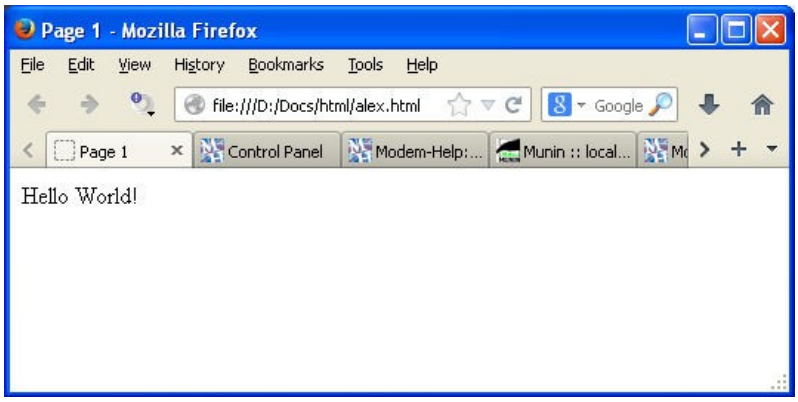
how to refresh:

- press 'f5' key, or
- press circ arrow (far right, address-bar)

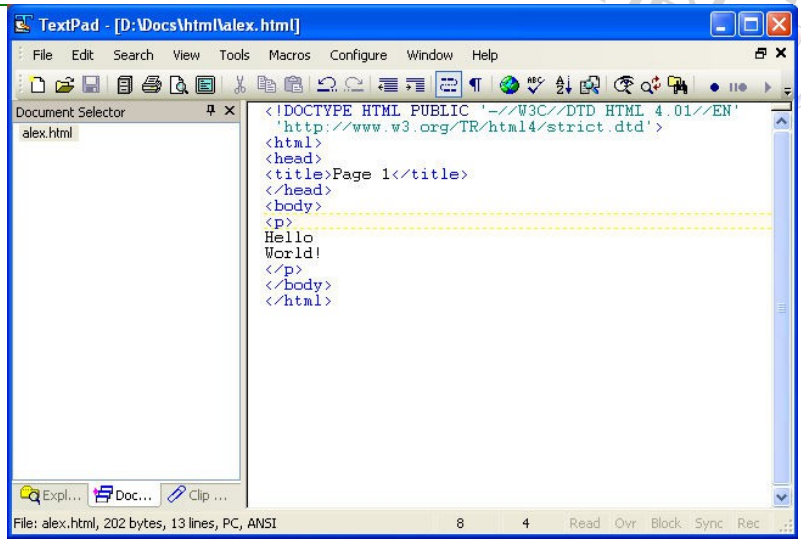
how to force-refresh:

- press <Ctrl> + 'f5' key (both keys at same time), or
- press <Shift> + circ arrow (both keys at same time)

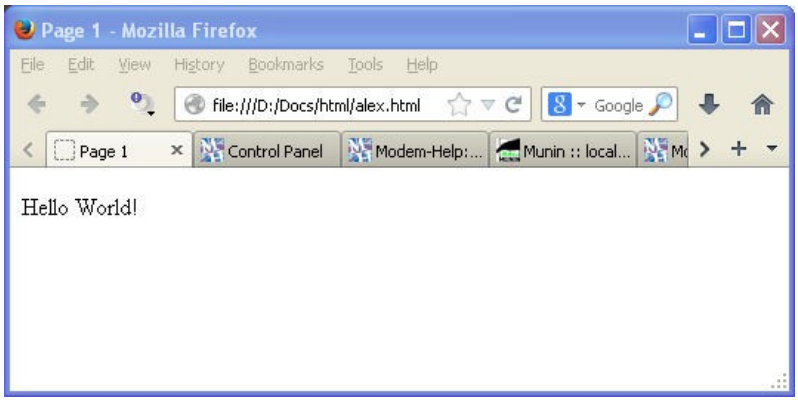
(the difference between 'refresh' & 'force-refresh' is that the latter does not allow the local Browser cache to be used)



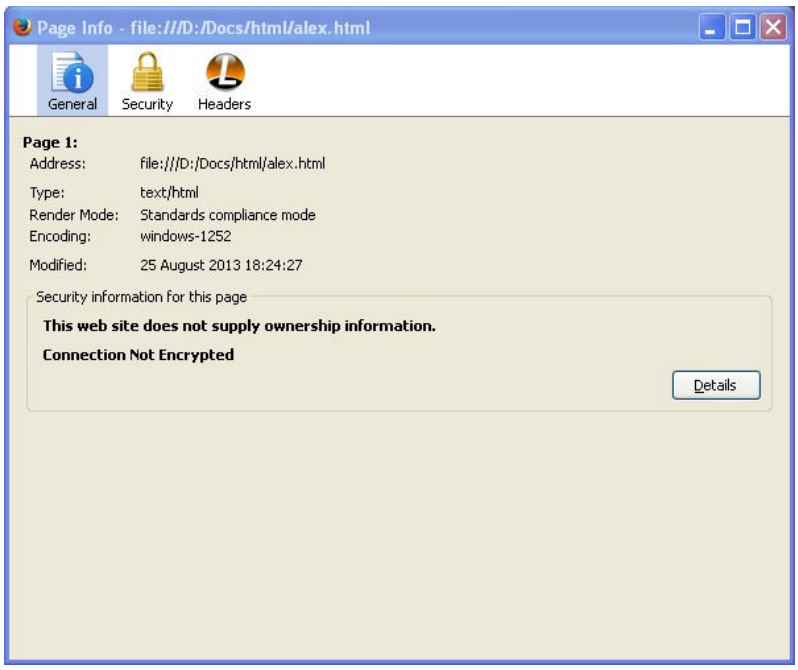
Add a DTD



This is a *Document Type Declaration* (DTD), and it declares to the Browser's parser as to which specific type of XML/SGML document this is (an HTML document is one of the *Standard Generalized Markup Language* (SGML) family of markup languages).



If you save the file, then refresh the Browser you will see a small change in the window (the text moves down a little). The default Style Sheet is the reason for the text movement – that will be explained later.



Importantly, the *Page Info* has also finally changed. According to this statement, page display for all elements should now be perfectly reliable (*"Render Mode: Standards compliance mode"*). Hooray!

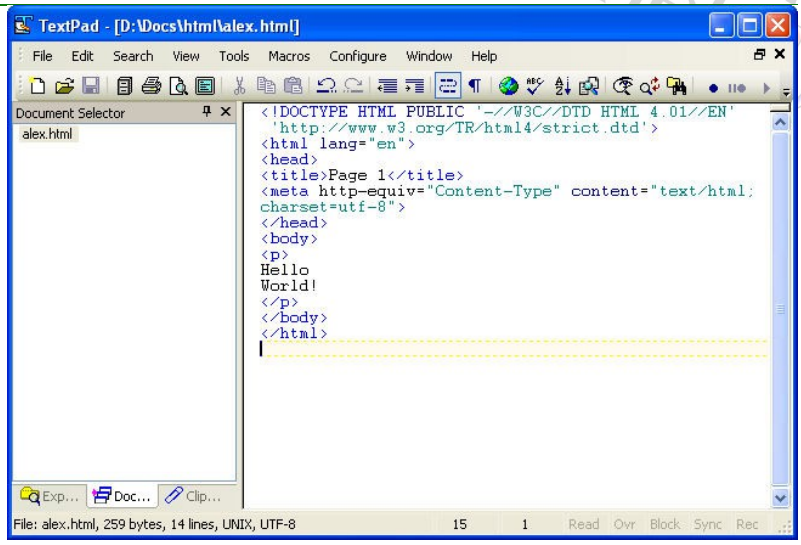
Now look at the next line: (*"Encoding: windows-1252"*). That is because *TextPad* was set to save the file as:

- Code set: ANSI
- File type: PC

...plus my Windows XP is set for the windows-1252 code-page☞ (*"ANSI Latin 1; Western European (Windows)"*) (yours may well be different).

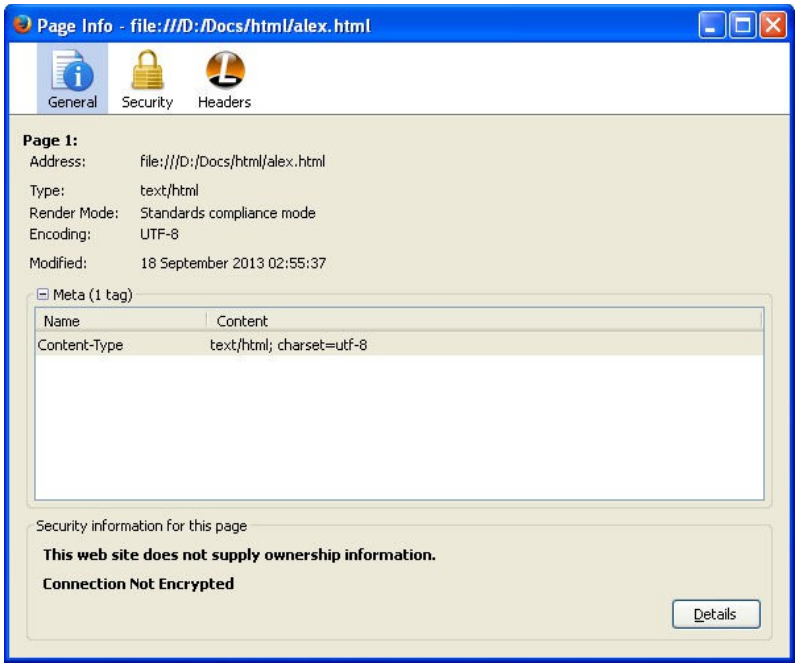
Well, that all sounds very complicated, and it is! What is *easy* to understand is that saving in a Windows' format may be difficult for users of Linux, Mac & other computers, plus saving in a *"Western European"* format may be difficult for people in a different region. Which they are, and it is easy to fix both, so let's now do something about that.

Change your language



You can see at right that we have added ' lang="en"' to the html element (which defines the default language for the whole document).

We have also added a mysterious `meta` element in the `head`; this defines the charset of the content to be 'utf-8'. More on that later. The *meta* statement is on one line (2 in the screen-shot), and 2 actual lines should also be fine in the html.



The *Page Information* has now also changed; the 'Encoding' is now 'UTF-8', and Firefox also shows us the *Meta* info from the file.

I also changed the file format when I saved it in TextPad to be 'UNIX' and 'UTF-8'. The last one does not actually change anything to the file nor it's display (for the geeks: all the current characters are identical to 7-bit ASCII, which itself is identical to those same characters in UTF8). The first makes the file a little bit smaller (geeks: line-endings change from '<CR><LF>' to just '<LF>').

UTF8

UTF8 is a rather wonderful & clever charset that will let you write *any character from any language in the world* (no kidding). On the up-side, it means that, from now on, you can write in your own language (content-only: all the HTML elements are US-ASCII, and no other *charset* must be used for them), and that language can be any of the hundreds of languages that exist on the surface of this planet (and one or two that do not – yes, *Klingons*, I'm looking at you). Importantly, in 2008 utf-8 became the most-used charset on the web☞ (as reported by Google), which means that your web-work will be future-proofed. All modern browsers can read & display utf-8 correctly, often not true for other applications on the OS, and especially under Windows.

Text Editor UTF8 issues

The main downside is that you have to make sure that your text-editor will allow you to write & display in plain-text utf-8. It is mostly a setup issue, though Windows users may have problems (TextPad does not handle utf-8 properly) (I shall only be producing ASCII text). Another feature to beware of is to make sure that you are using a *Plain*-Text UTF8 editor. Many editors—such as the one that I am producing this PDF on—will introduce codes into the text (notice that all the "quotations" use *curly* quotes, as just one example) (UTF8 has *curly* quotes but your editor may not be using them). This is also often a setup issue, and can sometimes be fixed by a plain-text *Save* option.

The final point to note here is that nothing has changed in the browser display due to our changes. Our future options have opened up, but nothing else has changed.

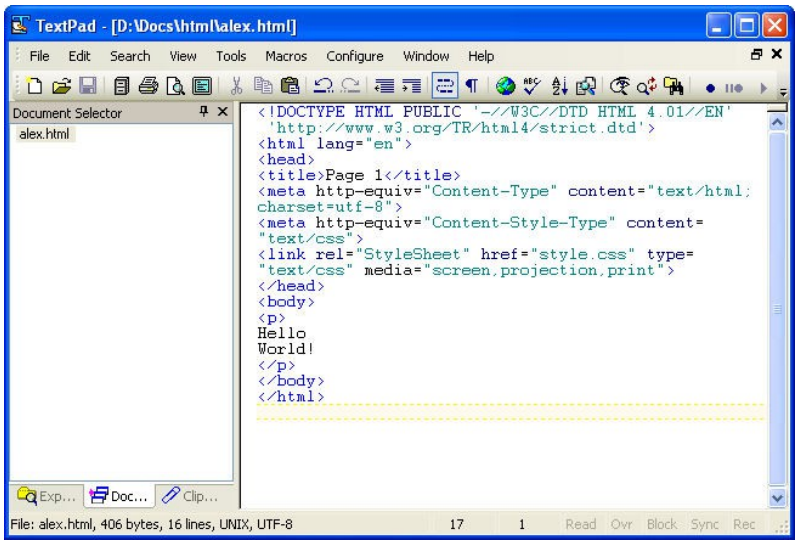
Add some style

What we have done so far is good but a little… dull. And I'm not talking just about the content (that is up to you) – it is how it looks; the *presentation* of that content. In a (HTML) word, the *style*.

10 or 20 years ago you would have been advised at this point to begin littering your html markup with FONT & COLOR attributes. No longer. This is the 21st Century, and today we do it with *style*.

We need 2 additions to the html file in this, your first web-page (can you spot them?):-

1. Another of those mystery `meta` elements, which always go in the `head`. This one is to declare the 'Content-Style-Type' *type*.
2. The file that will provide *StyleSheet* information for the entire file, which is done with a `link` element (also in the `head`).



Have you noticed what's missing? Well, yes, it's a good idea to also have the *StyleSheet* file in place; nothing will happen to the display until it is there. You may think at first that there is little that we can do with this file as it currently stands; after all, there is only *"Hello World!"* inside of a `p` element. But you would be wrong! (you have forgotten the `body` , which is *always* there—even if not not defined—and that governs all sorts of things).

A StyleSheet added

Be warned! *StyleSheets* can become compulsive, because they are so much fun! The trick with *StyleSheets* is to first get one with all the *style* for all the elements that you may use—and one that you like, obviously—and put the commands into the `head` to load it, and then forget it. Otherwise, you can spend hours & hours & *hours* messing about. Or perhaps that's just me... Save the StyleSheet below as plain-text in the same dir as your html file.

This one starts with a comment. Comments begin with a *slash dot* ('`/*`') and finish with a *dot slash* ('`*/`'). *Everything* in between—including tabs, newlines and the *slash dot* & *dot slash* is removed by the browser from the stylesheet before use.

Next is the `body`, and then the `p` element style code. Then follow some elements that we do not yet have in our page but may do.

Here is some brief info on *StyleSheets* to help you begin to play with this one:-

{curly brackets}

A pair of curly brackets ('{}') following an element mean that *all* the style-statements between the brackets apply to that element. The semi-colon (';') at the end of each statement is *required;* the newlines & spaces are optional.

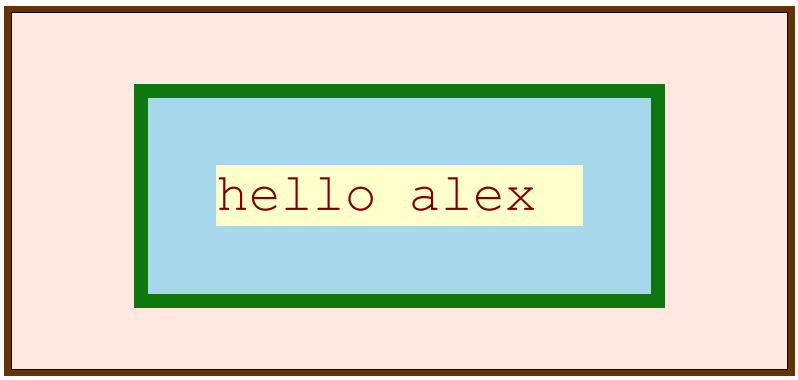
2 elements together (eg '`p input`') apply only when the second one is contained inside the first (an `input` inside a `p` element). Then, those *style* statements override both any `p` style *and* any bare `input` style. Comma-separated elements ('`p input,option`') means *style* applies to both "'`p input`' + '`p option`'". Be warned: style can get *really* complicated.

```
/* style.css
       StyleSheet for html Primers

   Alex Kemp
*/
body {
       width: 100%;
       background: #ffffe8; /* ne6mac default is grey
*/
       color: #000000;
       font-family: helvetica, sans-serif;
       margin: 0;  /* ie default !=0 */
       padding: 0; /* Opera default !=0 */
}
p {
       font-size: 1.0em;
       line-height: 1.8em;
       margin: 1.2px 0px 1.2px 0px;
}
p input,option,select,textarea { /* iepc textarea
always in serif monospace font, regardless */
       font-size: 1.0em;
       line-height: 1.3em;
}
div {
       background: white;
       color: black;
       padding: 0 3px;
}
body div {
       border: 0.74em solid #ffffe8;
}
/*typography*/
a {
       background: none transparent;
       TEXT-DECORATION: none;
}
a:link       { color: #0000ff; }
a:visited    { color: #e105ba; }
a:active           { color: #ff0000; }
a:hover            { color: #ff0000; TEXT-
DECORATION: underline; }
acronym { /* borrowed from php.net */
       border-bottom: 1px dashed #00cc00;
}
/* -- Now instructions purely for printing -
essentially, turn background to white -- */
@media print {

body {
       background: white;
       color: #000000;
}
body div {
       border: 0.74em solid white;
}
```

Getting boxy...

The W3C—the folks responsible for specifying both HTML *and* STYLE—have specified a 'Box Model' that is, frankly, nuts. Well, it's all a touch silly; Microsoft tried to fix it with Internet Explorer 5 on the PC☞, but no-one else would accept that, and now we are all stuck with it. Let's see if I can make sense of it for you...

Let's imagine my grandchildren have decided to post me a present (yes please!) and here it is: a green box with something nice in the middle. Sensibly, they have put it in a brown cardboard box with lots of protective wrapping.



Now, this is how a *StyleSheet* specifies the green box in the middle; it says that the green box has a *border*, a *margin* & a *padding* (plus a *width* & *height*):

- border: The border is the green of the green-box itself. It has:
  - width (a length unit, or: thin | medium | thick)
  - style (none | dotted | dashed | solid | double | groove | ridge | inset | outset)
  - color
- margin: Margin is the (pink) space *outside* the green box, between it and the brown box that contains it. It has:
  - (just a value) (a length unit)
- padding: Padding is the (blue) space *inside* the green box, between it & the yellow contents. It has:
  - (just a value) (a length unit)

That all seems perfectly reasonable, and you are probably wondering why I called the W3C 'Box Model' "nuts". The answer comes when I ask you a question:

- What is the width of the green box?

If your answer is: *"the distance from the left to the right of the outside of the green box"*, or perhaps if you are trying to be clever: *"the width of the inside of the brown box minus the green-box margin"* then I would agree with you. But the W3C does not. The W3C answer is:

- The width of the contents (the yellow space).

Thus, for '`green-box width:`' you would put the length of the content-space. For the W3C *"padding and border are to be added to the width"* in order to find the dimensions of the green box. One of these days I may get over that☞, but for the last 15+ years I've thought that they are nuts. However, none of us have any choice, we must *all* go nuts the W3C way.

PS
The brown box also has a *border*, a *margin* & a *padding*, but I've ignored all that so far (as if they are all zero) to try to give everyone a chance to first understand the basic principles.

Reference Guide:- HTML

Block-Level Elements

Description: Most of the elements permitted within the html body are designated to have a *Content Model* that is either Block-Level or Inline. That affects how a browser will render them upon the canvas of the *display device* (a monitor, mobile, projector, etc.).

A Block-Level element is rendered according to the W3C Box Model. In brief, it normally begins upon a newline, but see the Box Model for full details.

Elements:

address	blockquote	
button	block or inline; when inline must NOT contain block-level elements	
caption		
dd	(can contain block-level elements)	
del	block or inline; when inline must NOT contain block-level elements	
div		
dl	(can contain block-level elements)	
dt	fieldset	form
frameset	(can contain block-level elements)	
h1	h2	h3
h4	h5	h6
hr		
ins	block or inline; when inline must NOT contain block-level elements	
legend	li	
map	block or inline; when inline must NOT contain block-level elements	
noframes	noscript	
object	block or inline; when inline must NOT contain block-level elements	
ol	p	pre
script	block or inline; when inline must NOT contain block-level elements	
table		
tbody	(can contain block-level elements)	
td	(can contain block-level elements)	
tfoot	(can contain block-level elements)	
th	(can contain block-level elements)	
thead	(can contain block-level elements)	
tr	(can contain block-level elements)	
ul		

Inline Elements

Description: Most of the elements permitted within the html body are designated to have a *Content Model* that is either Block-Level or Inline. That affects how a browser will render them upon the canvas of the *display device* (a monitor, mobile, projector, etc.).

An Inline element is rendered as part of the flow of text, with the element beginning at the next character position within that flow.

Elements:

a	abbr	acronym
b	bdo	big
br		
button	block or inline; when inline (within another inline element or a p) must NOT contain block-level elements	
cite	code	
del	block or inline; when inline (within another inline element or a p) must NOT contain block-level elements	
dfn	em	img
input		
ins	block or inline; when inline (within another inline element or a p) must NOT contain block-level elements	
kbd	label	
map	block or inline; when inline (within another inline element or a p) must NOT contain block-level elements	
object	block or inline; when inline (within another inline element or a p) must NOT contain block-level elements	
optgroup	option	q
samp		
script	block or inline; when inline (within another inline element or a p) must NOT contain block-level elements	
select	small	span
strong	sub	sup
textarea	tt	u
var		

Other Elements

Description: There are some in every gathering!

Most HTML elements are either Block-Level, Inline or can be either; this page gathers together those that are neither. Most of this miscellany are elements that, for one reason or another, are not rendered directly upon the canvas.

Elements:

area
base
body
col
colgroup
doctype
frame
head
html
link
meta
param
style
title

Generic Attributes 4.01

Description: These attributes are (mostly) available for every element. See the individual element page for specifics.

core-attributes:

`id`	document-wide unique id (new to HTML4)
`class`	space-separated list of classes (new to HTML4)
`style`	associated style info (new to HTML4)
`title`	advisory title (normally rendered in a visual browser as a *'tooltip'*)

language-attributes:

`lang`	2-char language codes☞
`dir`	(ltr|rtl) direction for weak/neutral text

event-attributes:

`onclick`	a pointer button was clicked
`ondblclick`	a pointer button was double clicked
`onmousedown`	a pointer button was pressed down
`onmouseup`	a pointer button was released
`onmouseover`	a pointer was moved onto
`onmousemove`	a pointer was moved within
`onmouseout`	a pointer was moved away
`onkeypress`	a key was pressed and released
`onkeydown`	a key was pressed down
`onkeyup`	a key was released

Character Entities

Description: These are academic if you use the UTF-8 *charset*.

> HTML2.0 introduced '"', '&', '<' and '>'. HTML3.2 added Latin-1 *entities;* these are US-ASCII (7-bit character) short-codes to represent 8-bit characters in the Latin-1 charset. HTML4 extended those with "Mathematical, Greek and Symbolic" + "Special" entities. Browser support was always patchy (apart from the original HTML2 foursome), and now UTF-8 allows entry of *any* character directly (but you may need to know what these entities are).

Entity tables:

- HTML2 + Latin-1
- Mathematical, Greek and Symbolic
- Special

Notes for the tables:

- 'Alpha'
  The entity representation is (case-sensitive):

  ```
  &Alpha;
  ```

  So, prepend '&' and append ';'. An example is:

  ```
   
  ```
  (a non-breaking space)

- 'Dec'
  (decimal) The entity representation is:

  ```
  &#Dec;
  ```

  So, prepend '&#' and append ';'. An example is:

  ```
   
  ```
  (a non-breaking space)

- 'Hex'
  (hexadecimal) The entity representation is (case-insensitive):

  ```
  &#xHex;
  ```

  So, prepend '&#x' and append ';'. An example is:

  ```
   
  ```
  (a non-breaking space)

  ' ' == ' ' == 'U+00A0' (the unicode/UTF-8 code)

HTML2 + Latin-1 Entities

Entity			Name	Glyph
Alpha	Dec	Hex		
quot	034	0022	quotation mark	"
amp	038	0026	ampersand	&
lt	060	003c	left angle-bracket (‘less than’ symbol)	<
gt	062	003e	right angle-bracket (‘greater than’ symbol)	>
nbsp	160	00a0	no-break space = non-breaking space	
iexcl	161	00a1	inverted exclamation mark	¡
cent	162	00a2	cent sign	¢
pound	163	00a3	pound sign	£
curren	164	00a4	currency sign	¤
yen	165	00a5	yen sign = yuan sign	¥
brvbar	166	00a6	broken bar = broken vertical bar	¦
sect	167	00a7	section sign	§
uml	168	00a8	diaeresis = spacing diaeresis	¨
copy	169	00a9	copyright sign	©
ordf	170	00aa	feminine ordinal indicator	ª
laquo	171	00ab	left-pointing double angle quotation mark = left pointing guillemet	«
not	172	00ac	not sign	¬
shy	173	00ad	soft hyphen = discretionary hyphen	
reg	174	00ae	registered sign = registered trade mark sign	®
macr	175	00af	macron = spacing macron = overline = APL overbar	¯
deg	176	00b0	degree sign	°
plusmn	177	00b1	plus-minus sign = plus-or-minus sign	±
sup2	178	00b2	superscript two = superscript digit two = squared	²
sup3	179	00b3	superscript three = superscript digit three = cubed	³
acute	180	00b4	acute accent = spacing acute	´
micro	181	00b5	micro sign	µ
para	182	00b6	pilcrow sign = paragraph sign	¶
middot	183	00b7	middle dot = Georgian comma = Greek middle dot	·
cedil	184	00b8	cedilla = spacing cedilla	¸
sup1	185	00b9	superscript one = superscript digit one	¹
ordm	186	00ba	masculine ordinal indicator	º
raquo	187	00bb	right-pointing double angle quotation mark = right pointing guillemet	»
frac14	188	00bc	vulgar fraction one quarter = fraction one quarter	¼
frac12	189	00bd	vulgar fraction one half = fraction one half	½
frac34	190	00be	vulgar fraction three quarters = fraction three quarters	¾
iquest	191	00bf	inverted question mark = turned question mark	¿
Agrave	192	00c0	latin capital letter A with grave = latin capital letter A grave	À
Aacute	193	00c1	latin capital letter A with acute	Á

Entity			Name	Glyph
Alpha	Dec	Hex		
Acirc	194	00c2	latin capital letter A with circumflex	**Â**
Atilde	195	00c3	latin capital letter A with tilde	**Ã**
Auml	196	00c4	latin capital letter A with diaeresis	**Ä**
Aring	197	00c5	latin capital letter A with ring above = latin capital letter A ring	**Å**
AElig	198	00c6	latin capital letter AE = latin capital ligature AE	**Æ**
Ccedil	199	00c7	latin capital letter C with cedilla	**Ç**
Egrave	200	00c8	latin capital letter E with grave	**È**
Eacute	201	00c9	latin capital letter E with acute	**É**
Ecirc	202	00ca	latin capital letter E with circumflex	**Ê**
Euml	203	00cb	latin capital letter E with diaeresis	**Ë**
Igrave	204	00cc	latin capital letter I with grave	**Ì**
Iacute	205	00cd	latin capital letter I with acute	**Í**
Icirc	206	00ce	latin capital letter I with circumflex	**Î**
Iuml	207	00cf	latin capital letter I with diaeresis	**Ï**
ETH	208	00d0	latin capital letter ETH	**Ð**
Ntilde	209	00d1	latin capital letter N with tilde	**Ñ**
Ograve	210	00d2	latin capital letter O with grave	**Ò**
Oacute	211	00d3	latin capital letter O with acute	**Ó**
Ocirc	212	00d4	latin capital letter O with circumflex	**Ô**
Otilde	213	00d5	latin capital letter O with tilde	**Õ**
Ouml	214	00d6	latin capital letter O with diaeresis	**Ö**
times	215	00d7	multiplication sign	**×**
Oslash	216	00d8	latin capital letter O with stroke = latin capital letter O slash	**Ø**
Ugrave	217	00d9	latin capital letter U with grave	**Ù**
Uacute	218	00da	latin capital letter U with acute	**Ú**
Ucirc	219	00db	latin capital letter U with circumflex	**Û**
Uuml	220	00dc	latin capital letter U with diaeresis	**Ü**
Yacute	221	00dd	latin capital letter Y with acute	**Ý**
THORN	222	00de	latin capital letter THORN	**Þ**
szlig	223	00df	latin small letter sharp s = ess-zed	**ß**
agrave	224	00e0	latin small letter a with grave = latin small letter a grave	**à**
aacute	225	00e1	latin small letter a with acute	**á**
acirc	226	00e2	latin small letter a with circumflex	**â**
atilde	227	00e3	latin small letter a with tilde	**ã**
auml	228	00e4	latin small letter a with diaeresis	**ä**
aring	229	00e5	latin small letter a with ring above = latin small letter a ring	**å**
aelig	230	00e6	latin small letter ae = latin small ligature ae	**æ**
ccedil	231	00e7	latin small letter c with cedilla	**ç**
egrave	232	00e8	latin small letter e with grave	**è**
eacute	233	00e9	latin small letter e with acute	**é**
ecirc	234	00ea	latin small letter e with circumflex	**ê**
euml	235	00eb	latin small letter e with diaeresis	**ë**
igrave	236	00ec	latin small letter i with grave	**ì**

Entity			Name	Glyph
Alpha	Dec	Hex		
iacute	237	00ed	latin small letter i with acute	**í**
icirc	238	00ee	latin small letter i with circumflex	**î**
iuml	239	00ef	latin small letter i with diaeresis	**ï**
eth	240	00f0	latin small letter eth	**ð**
ntilde	241	00f1	latin small letter n with tilde	**ñ**
ograve	242	00f2	latin small letter o with grave	**ò**
oacute	243	00f3	latin small letter o with acute	**ó**
ocirc	244	00f4	latin small letter o with circumflex	**ô**
otilde	245	00f5	latin small letter o with tilde	**õ**
ouml	246	00f6	latin small letter o with diaeresis	**ö**
divide	247	00f7	division sign	**÷**
oslash	248	00f8	latin small letter o with stroke = latin small letter o slash	**ø**
ugrave	249	00f9	latin small letter u with grave	**ù**
uacute	250	00fa	latin small letter u with acute	**ú**
ucirc	251	00fb	latin small letter u with circumflex	**û**
uuml	252	00fc	latin small letter u with diaeresis	**ü**
yacute	253	00fd	latin small letter y with acute	**ý**
thorn	254	00fe	latin small letter thorn	**þ**
yuml	255	00ff	latin small letter y with diaeresis	**ÿ**

Mathematical, Greek and Symbolic Entities

Entity			Name	Glyph
Alpha	Dec	Hex		
fnof	402	0192	latin small f with hook = function = florin	**ƒ**
Alpha	913	0391	Greek capital letter alpha	**Α**
Beta	914	0392	Greek capital letter beta	**Β**
Gamma	915	0393	Greek capital letter gamma	**Γ**
Delta	916	0394	Greek capital letter delta	**Δ**
Epsilon	917	0395	Greek capital letter epsilon	**Ε**
Zeta	918	0396	Greek capital letter zeta	**Ζ**
Eta	919	0397	Greek capital letter eta	**Η**
Theta	920	0398	Greek capital letter theta	**Θ**
Iota	921	0399	Greek capital letter iota	**Ι**
Kappa	922	039a	Greek capital letter kappa	**Κ**
Lambda	923	039b	Greek capital letter lamda	**Λ**
Mu	924	039c	Greek capital letter mu	**Μ**
Nu	925	039d	Greek capital letter nu	**Ν**
Xi	926	039e	Greek capital letter xi	**Ξ**
Omicron	927	039f	Greek capital letter omicron	**Ο**
Pi	928	03a0	Greek capital letter pi	**Π**
Rho	929	03a1	Greek capital letter rho	**Ρ**
	930	03a2		**(n/a)**
Sigma	931	03a3	Greek capital letter sigma	**Σ**
Tau	932	03a4	Greek capital letter tau	**Τ**
Upsilon	933	03a5	Greek capital letter upsilon	**Υ**
Phi	934	03a6	Greek capital letter phi	**Φ**
Chi	935	03a7	Greek capital letter chi	**Χ**
Psi	936	03a8	Greek capital letter psi	**Ψ**
Omega	937	03a9	Greek capital letter omega	**Ω**
alpha	945	03b1	Greek small letter alpha	**α**
beta	946	03b2	Greek small letter beta	**β**
gamma	947	03b3	Greek small letter gamma	**γ**
delta	948	03b4	Greek small letter delta	**δ**
epsilon	949	03b5	Greek small letter epsilon	**ε**
zeta	950	03b6	Greek small letter zeta	**ζ**
eta	951	03b7	Greek small letter eta	**η**
theta	952	03b8	Greek small letter theta	**θ**
iota	953	03b9	Greek small letter iota	**ι**
kappa	954	03ba	Greek small letter kappa	**κ**
lambda	955	03bb	Greek small letter lamda	**λ**
mu	956	03bc	Greek small letter mu	**μ**
nu	957	03bd	Greek small letter nu	**ν**
xi	958	03be	Greek small letter xi	**ξ**

Entity			Name	Glyph
Alpha	Dec	Hex		
omicron	959	03bf	Greek small letter omicron	ο
pi	960	03c0	Greek small letter pi	π
rho	961	03c1	Greek small letter rho	ρ
sigmaf	962	03c2	Greek small letter final sigma	ς
sigma	963	03c3	Greek small letter sigma	σ
tau	964	03c4	Greek small letter tau	τ
upsilon	965	03c5	Greek small letter upsilon	υ
phi	966	03c6	Greek small letter phi	φ
chi	967	03c7	Greek small letter chi	χ
psi	968	03c8	Greek small letter psi	ψ
omega	969	03c9	Greek small letter omega	ω
thetasym	977	03d1	Greek small letter theta symbol	ϑ
upsih	978	03d2	Greek upsilon with hook symbol	ϒ
piv	982	03d6	Greek pi symbol	ϖ
bull	8226	2022	bullet = black small circle	•
hellip	8230	2026	horizontal ellipsis = three dot leader	…
prime	8242	2032	prime = minutes = feet	′
Prime	8243	2033	double prime = seconds = inches	″
oline	8254	203e	overline = spacing overscore	‾
frasl	8260	2044	fraction slash	⁄
image	8465	2111	blackletter capital I = imaginary part	ℑ
weierp	8472	2118	script capital P = power set = Weierstrass p	℘
real	8476	211c	blackletter capital R = real part symbol	ℜ
trade	8482	2122	trade mark sign	™
alefsym	8501	2135	alef symbol = first transfinite cardinal	ℵ
larr	8592	2190	leftwards arrow	←
uarr	8593	2191	upwards arrow	↑
rarr	8594	2192	rightwards arrow	→
darr	8595	2193	downwards arrow	↓
harr	8596	2194	left right arrow	↔
	8597	2195		↕
	8598	2196		↖
	8599	2197		↗
	8600	2198		↘
	8601	2199		↙
	8624	21b0		↰
	8625	21b1		↱
	8626	21b2		↲
	8627	21b3		↳
	8628	21b4		↴
crarr	8629	21b5	downwards arrow with corner leftwards = carriage return	↵
larr	8656	21d0	leftwards double arrow	⇐
uarr	8657	21d1	upwards double arrow	⇑

Entity			Name	Glyph
Alpha	Dec	Hex		
rarr	8658	21d2	rightwards double arrow	⇒
darr	8659	21d3	downwards double arrow	⇓
harr	8650	21d4	left right double arrow	⇔
	8651	21d5		⇕
	8652	21d6		⇖
	8653	21d7		⇗
	8654	21d8		⇘
	8655	21d9		⇙
forall	8704	2200	for all	∀
	8705	2201		∁
part	8706	2202	partial differential	∂
exist	8707	2203	there exists	∃
	8708	2204		∄
empty	8709	2205	empty set = null set = diameter	∅
	8710	2206		∆
nabla	8711	2207	nabla = backward difference	∇
isin	8712	2208	element of	∈
notin	8713	2209	not an element of	∉
	8714	220a		∊
ni	8715	220b	contains as member	∋
	8716	220c		∌
	8717	220d		∍
	8718	220e		∎
prod	8719	220f	n-ary product = product sign	∏
	8720	2210		∐
sum	8721	2211	n-ary sumation	∑
minus	8722	2212	minus sign	−
	8723	2213		∓
	8724	2214		∔
	8725	2215		∕
	8726	2216		∖
lowast	8727	2217	asterisk operator	∗
	8728	2218		∘
	8729	2219		∙
radic	8730	221a	square root = radical sign	√
	8731	221b		∛
	8732	221c		∜
prop	8733	221d	proportional to	∝
infin	8734	221e	infinity	∞
	8735	221f		∟
ang	8736	2220	angle	∠
	8737	2221		∡
	8738	2222		∢

Entity			Name	Glyph
Alpha	Dec	Hex		
	8739	2223		∣
	8740	2224		∤
	8741	2225		∥
	8742	2226		∦
and	8743	2227	logical and = wedge	∧
or	8744	2228	logical or = vee	∨
cap	8745	2229	intersection = cap	∩
cup	8746	222a	union = cup	∪
int	8747	222b	integral	∫
there4	8756	2234	therefore	∴
	8757	2235		∵
	8759	2237		∷
sim	8764	223c	tilde operator = varies with = similar to	∼
cong	8773	2245	approximately equal to	≅
asymp	8776	2248	almost equal to = asymptotic to	≈
ne	8800	2260	not equal to	≠
equiv	8801	2261	identical to	≡
	8802	2262	not identical to	≢
le	8804	2264	less-than or equal to	≤
ge	8805	2265	greater-than or equal to	≥
sub	8834	2282	subset of	⊂
sup	8835	2283	superset of	⊃
nsup	8836	2284	not a subset of	⊄
	8837	2285	not a superset of	⊅
sube	8838	2286	subset of or equal to	⊆
supe	8839	2287	superset of or equal to	⊇
	8840	2288	not a subset of or equal to	⊈
	8841	2289	not a superset of or equal to	⊉
oplus	8853	2295	circled plus = direct sum	⊕
otimes	8855	2297	circled times = vector product	⊗
perp	8869	22a5	up tack = orthogonal to = perpendicular	⊥
sdot	8901	22c5	dot operator	⋅
lceil	8968	2308	left ceiling = apl upstile	⌈
rceil	8869	2309	right ceiling	⌉
lfloor	8870	230a	left floor = apl downstile	⌊
rfloor	8871	230b	right floor	⌋
lang	9001	2329	left-pointing angle bracket = bra	**(n/a)**
rang	9002	232a	right-pointing angle bracket = ket	**(n/a)**
loz	9674	25ca	lozenge	◊
spades	9824	2660	black spade suit	♠
clubs	9827	2663	black club suit = shamrock	♣
hearts	9829	2665	black heart suit = valentine	♥
diams	9830	2666	black diamond suit	♦

Special Entities

Entity			Name	Glyph
Alpha	Dec	Hex		
quot	34	0022	quotation mark	"
amp	38	0026	ampersand	&
lt	60	003c	left angle-bracket (‘less than’ symbol)	<
gt	62	003e	right angle-bracket (‘greater than’ symbol)	>
OElig	338	0152	latin capital ligature OE	Œ
oelig	339	0153	latin small ligature oe	œ
Scaron	352	0160	latin capital letter S with caron	Š
scaron	353	0161	latin small letter s with caron	š
Yuml	376	0178	latin capital letter Y with diaeresis	Ÿ
circ	710	02c6	modifier letter circumflex accent	ˆ
tilde	732	02dc	small tilde	˜
ensp	8194	2002	en space	
emsp	8195	2003	em space	
thinsp	8201	2009	thin space	
zwnj	8204	200c	zero width non-joiner	
zxj	8205	200d	zero width joiner	
lrm	8206	200e	left-to-right mark	
rlm	8207	200f	right-to-left mark	
	8236	202c	end directionality override	
	8237	202d	force ltr directionality	
	8238	202e	force rtl directionality	
ndash	8211	2013	en dash	-
mdash	8212	2014	em dash	—
lsquo	8216	2018	left single quotation mark	‘
rsquo	8217	2019	right single quotation mark	’
sbquo	8218	201a	single low-9 quotation mark	‚
ldquo	8220	201c	left double quotation mark	“
rdquo;	8221	201d	right double quotation mark	”
bdquo	8222	201e	double low-9 quotation mark	„
dagger	8224	2020	dagger	†
Dagger	8225	2021	double dagger	‡
permil	8240	2030	per mille sign	‰
	8241	2031		‰‰
lsaquo	8249	2039	single left-pointing angle quotation mark	‹
rsaquo	8250	203a	single right-pointing angle quotation mark	›
euro	8364	20ac	euro sign	€

a - Anchor

Syntax: `<a>...</a>`

Description: An anchor is a fundamental part of html documents, as it provides the hypertext links between those documents.

See also: a, bdo, br, img, map, object, q, script, span, sub, sup

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements (not a)
Can contain:	Inline elements (not a)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

core:	- see core-attributes
i18n:	- see language-attributes
events:	- see event attributes

accesskey	accessibility key character
charset	char encoding of linked resource (RFC2045☞)
coords	for use with client-side image maps
href	URI for linked resource (RFC2396☞).
hreflang	2-char language codes☞
name	named link end (see also id within core-attributes).
onblur	(script) the element lost the focus
onfocus	(script) the element got the focus
rel	forward link types
rev	reverse link types
shape	for use with client-side image maps
tabindex	position in tabbing order
type	advisory content type (RFC2045☞)

id:

href:

name:

(within-page links):

Sometimes you want to be able to link to *sections* of your document, but upon the same page. That is done by using `a` with the `name` attribute, which names a *fragment* of your document. The link to it then employs a URI 'fragment' construct. Here is what naming a fragment of a document can look like (see this note on namespaces for why both 'id' & 'name' are used):

```
<a id=”section-A” name=”section-A”>Section A</a>
```

The technical name is a URI '*fragment*'. An anchor on the same document to that fragment would then look like this:

```
<a href="#section-A">Section A</a>
```

...or from another document:

```
<a href="doc.html#section-A">Section A</a>
```

abbr - Abbreviation

Syntax: `<abbr>...</abbr>`

Description: `'abbr'` & `'acronym'` are easily confused; each is an abbreviation of a longer word, but acronym is used when the short form is a pronounceable word. For each, `title` (see core-attributes) can be used to allow the long-form to be shown as a 'tooltip' in visual browsers.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

acronym – Acronym

Syntax: `<acronym>...</acronym>`

Description: `'abbr'` & `'acronym'` are easily confused; each is an abbreviation of a longer word, but acronym is used when the short form is a pronounceable word. For each, `title` (see core-attributes) can be used to allow the long-form to be shown as a 'tooltip' in visual browsers.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

address - Address

Syntax: `<address>...</address>`

Description: `address` is designed to provide contact information for the author of the document.

See also: `address`, `blockquote`, `div`, `dl`, `fieldset`, `form`, `hr`, `noscript`, `p`, `table`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

applet - Java applet

Syntax: `<applet width="100" height="100">...</applet>`

Description: `applet` is used to embed Java applets. It is deprecated in HTML4 in favour of `object`

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level except `pre` + Inline elements
Can contain:	`param` elements followed by block-level elements and/or inline elements
Standard:	HTML 4.01 Loose, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	vertical or horizontal alignment	
`alt`	short description	
`archive`	comma-separated archive list	
`code`	applet class file	
`codebase`	optional base URI for applet	
`height`	initial height	*Required*
`hspace`	horizontal gutter	
`name`	allows applets to find each other	
`object`	serialized applet file	
`vspace`	vertical gutter	
`width`	initial width	*Required*

See also: `param`

area - Image map region

Syntax: `<area alt="functional description">`

Description: `area` defines a client-side image map. HTML4 has extended the `map` element so that it can contain one or more block-level elements—such as `object`—in addition to the `area` element.

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Other
Contained in:	`map`
Can contain:	(empty)
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`accesskey`	accessibility key character	
`alt`	short description	*Required*
`coords`	comma-separated list of lengths	
`href`	URI for linked resource	
`nohref`	inactive region	
`onblur`	(script) the element lost the focus	
`onfocus`	(script) the element got the focus	
`shape`	(rect | circle | poly | default) (default rect) controls interpretation of coords	
`tabindex`	position in tabbing order	

Here is a typical pre- HTML4 usage:

```
<map name="mymap">
<area href="/reference/" alt="html and css reference" coords="5,5,95,195">
<area href="/design/" alt="design guide" coords="105,5,195,195">
<area href="/tools/" alt="tools" coords="205,5,295,195">
</map>
<img src="sitemap.gif" alt="site map" usemap="#mymap" width=300
height=200>
```

b - Bold text

Syntax: `<b>...</b>`

Description: `bold` and `strong` text by default are normally **rendered the same** in visual browsers.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

base - Document base url

Syntax: `<base>`

Description: `base` (one only) defines the *absolute URL* used to resolve document *relative* urls. It can be especially useful for docs sent in an email, etc.

See also: `base`, `isindex`, `head`, `link`, `meta`, `object`, `script`, `style`, `title`

Properties:

Content model:	Other
Contained in:	`head`
Can contain:	(empty)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`i18n`:	- see language-attributes

`href`	URI that acts as base URI	*Required*
`target`	(4.01 Frameset) render links in that frame by default	

href:

(*Required* in HTML4 Strict but optional in Loose) `href` specifies an absolute URI that acts as the base URI for resolving relative URIs. Most of the time this element should not be needed, as the browser will be able to accurately determine what the base url is, or perhaps all `href` attributes throughout the document are already *absolute* URLs.

absolute v's relative URI:

A URI can consist of many parts. The *relative* URI *starts* at the part following the last slash ('/') in the *path*:

URI: scheme://user:pass@host:port/path?query#fragment
eg: http://www.w3.org/TR/REC-html40/struct/links.html#h-12.4.1☞

- scheme: "http"
- user: (optional) username
- pass: (optional) password
- host: "www.w3.org"
- port: (optional) port number (eg 80 for most web documents)
- path: (can be empty) "TR/REC-html40/struct/links.html"
- query: (optional) (not used here)
- fragment: (optional) "h-12.4.1"

Using the previous example, that document could have the following (note that a *fragment* would NOT appear within a `base.href`, and that a ".." says *"start at the next higher directory"*, which here is "REC-html40"):

base.href:
http://www.w3.org/TR/REC-html40/struct/links.html
start dir for relative links:
http://www.w3.org/TR/REC-html40/struct/

example relative link:
```
<a href="../index/list.html">Index</a>
```
...which latter would resolve to the URI:
http://www.w3.org/TR/REC-html40/index/list.html☞

applet & object:

The `object` and `applet` elements define attributes that take precedence over the value set by the BASE element.

basefont - Base font change

Syntax: `<basefont>...</basefont>`

Description: `basefont` is used to change the base font of all content that follows the instruction. It is deprecated in HTML4 in favour of `style-sheets` (and, just like `font`, also considered completely naff).

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level except `pre` + inline elements
Can contain:	(empty)
Standard:	HTML 4.01 Loose, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`size`	[+|-]nn e.g. size="+1", size="4"
`color`	(sRGB colours) text colour
`face`	comma-separated list of font names

bdo - BiDi override

Syntax: `<bdo dir="ltr">...</bdo>`

Description: `bdo` takes a little bit of explanation (below):

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

Unicode and/or UTF8 chars have an inherent property of *directionality*: essentially, alphabets from languages to the *east* (and inclusive) of Israel are written right-to-left, whilst those *west* of Israel are written left-to-right. `bdo`, together with the `dir` attribute (see language-attributes), is used to signal a change to the default direction of those characters (typically when a word from a language with one dir is included within (say) a quotation with a different dir value).

Note: Unicode contains identical signals within the charset:

`&#x202D;` - force ltr directionality
`&#x202E;` - force rtl directionality
`&#x202C;` - end the override

big - Large text

Syntax: `<big>...</big>`

Description: `big` is normally rendered in a visual browser in a larger font size. It is also possible to nest this element, which can lead to unpredictable results, as you cannot be certain of the reader's starting font size.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level except pre + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

blockquote - Block quotation

Syntax: `<blockquote>...</blockquote>`

Description: `blockquote` is normally compared to `q` (short quotation):
`q` is inline & can contain only inline elements
`blockquote` can contain block-level elements

See also: `address`, `blockquote`, `div`, `dl`, `fieldset`, `form`, `hr`, `noscript`, `p`, `table`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	*4.01 Strict:* One or more block-level elements, or `script`. *4.01 Loose*: inline elements or block-level elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

Paradoxically, whilst `blockquote` can *contain* `p` elements it can NOT itself be contained within a paragraph element under 4.01 Strict (to do so would cause the render mode to switch to "Quirks mode").

body - Document body

Syntax: `<body>...</body>`

Description: `body` contains the document *body*. Curiously, it is *Required* within the html container object-model, yet it's start & end tags are *optional* (making it then a kind of *ghost* in the machine).

See also: `body`, `del`, `frameset`, `ins`, `html`, `noframes`, `script`

Properties:

Content model:	Other
Contained in:	4.01 Strict: `html` 4.01 Loose: `html` 4.01 Frameset: `noframes`
Can contain:	4.01 Strict: one or more block-level elements or `del`, `ins`, `script` 4.01 Loose: inline elements, block-level elements, `del`, `ins` 4.01 Frameset: (same as Loose)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`alink`	(COLOR) colour of selected links (deprecated)
`background`	(URI) texture tile for document background (deprecated)
`bgcolor`	(COLOR) document background colour (deprecated)
`link`	(COLOR) colour of links (deprecated)
`onload`	(script) the document has been loaded
`onunload`	(script) the document has been removed
`text`	(COLOR) document text colour (deprecated)
`vlink`	(COLOR) colour of visited links (deprecated)

frameset:

HTML4 Frameset documents do not use `body` at all, but rather use `frameset` as their equivalent element. The one exception to this rule is `noframes`, which has `body` as it's first element (implicitly or explicitly) within the content.

br - Line break

Syntax: `<br>`

Description: `br` forces a break within the flow of text.

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	(empty)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`clear`	(left|all|right|none) control of text flow (deprecated)

clear=(left|all|right|none):

The (deprecated) `clear` attribute is used to decide whether the flow of content continues below adjacent, floating blocks (normally images or tables).
eg
`<br clear="left">:` move next content flow below any content at left (but together with content at right)

button - Button

Syntax: `<button type="submit" name="something">...</button>`

Description: `button` is new in HTML4, and defines a *submit* (the default), *reset* or (push) button. Pre- HTML4 browsers used the `input` control; `button` is designed to allow a greater range of internal labels, including images & emphasis.

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

Properties:

Content model:	Inline
Contained in:	Block-level elements, inline elements except `button`
Can contain:	Inline elements except `a`, `button`, `input`, `iframe`, `label`, `select` or `textarea` Block-level elements except `fieldset`, `form` or `isindex`
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes	
`i18n:`	- see language-attributes	
`events:`	- see event attributes	
`accesskey`	accessibility key character	
`disabled`		
`name`	submit as part of form	*Required*
`onblur`	(script) the element lost the focus	
`onfocus`	(script) the element got the focus	
`tabindex`	position in tabbing order	
`type`	(button|reset|submit) (default: submit) for use as form button	*Required*
`value`	sent to server when submitted	

type="...":

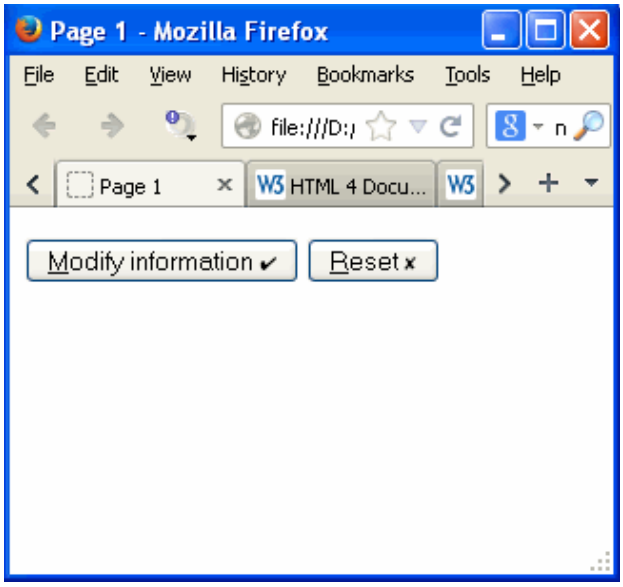
name:

value:

`name` and `value` attributes act in the same way as other controls: their value pairs are sent to server when a *submit* button is pu*sh*ed (if no `type` attribute is specified then the `button` is a *submit* button; good programming practice dictates that you *always* specify name & value atributes for submit & push buttons, even if they are the same as the default values).

Below is a typical usage defining 2 buttons, with the results below. Note that neither image file is available, so thebrowser shows the *alt* values. Note also that that is a *very* poor example for me to set (!), as the web-server error logs would fill up with 404 notices for the missing files:

```
<p>
<button name="submit" value="modify" accesskey="M">
  <span style="text-decoration:underline;">M</span>odify information
  <img src="checkmark.gif" alt="&#10004;">
</button>
<button name="reset" accesskey="R">
  <span style="text-decoration:underline;">R</span>eset
  <img src="x.gif" alt="&#10008;">
</button>
</p>
```



accesskey:

The examples above use 'M' & 'R' as the access-keys, with the point here that each are Capital letters. Any single Unicode/UTF8 letter, or *entity*, can be used, and this flexibility can easily lead into usability issues. How are your users, as one example, going to manage if you specify "`&AElig;`" as the access-character (the *entity* for "Æ"), let alone Unicode/UTF8, where it is perfectly possible to specify a character from the Greek, Cyrillic, Israeli, Arabic, Chinese, Japanese☞ or many other alphabets?

Even the examples above have a usability issue... The classic way to access a control (I was using winXP) is to use the 'Alt' key + the letter (both keys at the same time). So, whilst testing the buttons above I did "Alt+m" & "Alt+r", but neither worked! That was because it needed to be "Alt+M" or "Alt+R" (3 keys at the same time, and not 2). On such small issues will your users curse you, or thank you.

caption - Table caption

Syntax: `<caption>...</caption>`

Description: `caption` optionally defines a caption for the table, and is a useful useability feature for users of all types of browser. `caption` allows a block of (inline) text, whilst the `summary` attribute of `table` allows a mid-length line of text (but mostly for users using browser text-readers) and the `title` attribute of `table` allows a brief tooltip-type text.

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

Properties:

Content model:	Block
Contained in:	`table`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(top|bottom|left|right) caption position relative to table (deprecated)

If used, `caption` must be the first element in the table. See table: the whole thing for most of the elements of `table` assembled together.

center - Centred block

Syntax: `<center>...</center>`

Description: The W3C describes `center` as *"shorthand for DIV align=center"* (that attribute is also deprecated throughout HTML4 Strict).

A note for UK children: we Brits make a point, most generously, of *never*, *ever* pointing out to the Yanks that they cannot spell *"centre"* correctly.

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements, block-level elements
Standard:	HTML 4.01 Loose, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`center` has retained a strong connection with `table` due to poor support from HTML4 & earlier browsers for the `align` attribute + early lack of support for the CSS method of centring tables (setting horizontal margins to 'auto').

cite - Citation

Syntax: `<cite>...</cite>`

Description: `cite` is normally rendered in a visual browser in *italics*.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

code - Computer code

Syntax: `<code>...</code>`

Description: `code` is normally rendered in a visual browser in `monospaced text`.
Note: `code` contained within any container other than `pre` will have any multiple contiguous spaces collapsed to a single space; that is probably not what you want.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

col - Table column

Syntax: `<col>`

Description: `col` optionally defines common attributes.within a column of `table` data-cells (`td` or `th`). `colgroup` is used to group columns structurally, whilst `col` saves typing out the identical attributes for each of the data-cells.

Note: support is poor for `col` in early HTML4 browsers.

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

Properties:

Content model:	Other
Contained in:	`colgroup`, `table`
Can contain:	(empty)
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify|char) horizontal alignment of cells (deprecated)
`char`	(default: '.') char value if `align=char`
`charoff`	(pixels or %) offset in cell to char if `align=char`
`span`	(default: 1) COL attributes affect N columns
`valign`	(top|middle|bottom|baseline) vertical alignment of cells
`width`	(pixels or % or relative) column width

If used, `col` must be *after* the optional `caption` and *before* the optional `thead`. See table: the whole thing for most of the elements of `table` assembled together.

colgroup - Table column group

Syntax: `<colgroup>...<colgroup>`

Description: The optional `colgroup` is intended to "*create structural divisions within a table*".by grouping columns of `table` data-cells (`td` or `th`) together. Fine intentions then subverted by the HTML designers providing the `col` element, which (seems to have) the identical attributes & action of the `colgroup` element (*doh!*). Early browser designers for HTML4 ignored *both* elements.

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

Properties:

Content model:	Other
Contained in:	`table`
Can contain:	Zero or more `col` elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify|char) horizontal alignment of cells (deprecated)
`char`	(default: '.') char value if `align=char`
`charoff`	(pixels or %) offset in cell to char if `align=char`
`span`	(default: 1) (ignored if `colgroup` contains `col` elements) default number of columns in group
`valign`	(top|middle|bottom|baseline) vertical alignment of cells
`width`	(pixels or % or relative) column width

`colgroup` is *very* useful when a `table` contains a very large number of columns with identical layout. `col` elements can then be used if one or two columns differ from the rest (`col` attributes will override `colgroup` attributes, and any span on `colgroup` will be ignored).

If used, `colgroup` must be *after* the optional `caption` and *before* the optional `thead`. See table: the whole thing for most of the elements of `table` assembled together.

dd - Definition description

Syntax: `<dd>...</dd>`

Description: `dd` is part of a useful list structure.

See also: `dd`, `dl`, `dt`

Properties:

Content model:	Block
Contained in:	`dl` elements
Can contain:	Inline elements, block-level elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

See the whole thing for an example of all *definition* elements put together.

del - Deleted text

Syntax: `<del>...</del>`

Description: `del` is normally rendered in a visual browser in ~~strikethrough text~~., and is intended to indicate text *deleted from this version of the document* It is unusual in being capable of being either a block-level *or* an inline element. If used in an inline context—for example, within a paragraph—it cannot then contain any block-level elements.

See also: `ins`, `del`, `strike`

Properties:

Content model:	Inline *or* Block
Contained in:	Block-level elements, inline elements
Can contain:	Block-level elements, inline elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`cite`	(URI) info on reason for change
`datetime`	date and time of change (ISO format: "YYYY-MM-DDThh:mm:ssTZD")

cite:

datetime:

title:

`cite` + `datetime` are optionally available to give a url for readers to obtain further information on the reason for the change (`cite`) and the time & date of that change (`datetime`). W3C states☞ that "*they are primarily intended for private use (e.g. by server-side scripts collecting statistics about a site's edits), not for readers*". A simpler method to give succinct info is via the `title` attribute (see core-attributes).

dfn - Defined term

Syntax: `<dfn>...</dfn>`

Description: `dfn` is normally rendered in a visual browser in *italic text*.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

dir - Directory list

Syntax: `<dir>...</dir>`

Description: `dir` is a.list of items for a directory list (deprecated in HTML 4 Strict in favour of `ol` or `ul`). Note that the contained `li` elements are NOT allowed to contain block-level elements, which prevents a nested listing of (for example) sub-directories.

See also: `dir`, `li`, `menu`, `ol`, `ul`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	One or more `li` elements (which cannot contain block-level elements)
Standard:	HTML 4.01 Loose, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`compact`	display in a compact style (deprecated)

compact:

`compact` is poorly supported by browsers.

div - Generic block-level container

Syntax: `<div>...</div>`

Description: `div` is normally compared to `span` (inline container):
The intention of both containers is essentially to allow style to be applied to a set of block-level (`div`) or inline (`span`) elements.

See also: `address`, `blockquote`, `div`, `dl`, `fieldset`, `form`, `hr`, `noscript`, `p`, `table`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements, block-level elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left | center | right | justify) horizontal alignment (deprecated)

dl - Definition list

Syntax: `<dl>...</dl>`

Description: `dl` is the container for a useful list structure.

See also: `address`, `blockquote`, `div`, `dl`, `fieldset`, `form`, `hr`, `noscript`, `p`, `table`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	One or more `dt` or `dd` elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

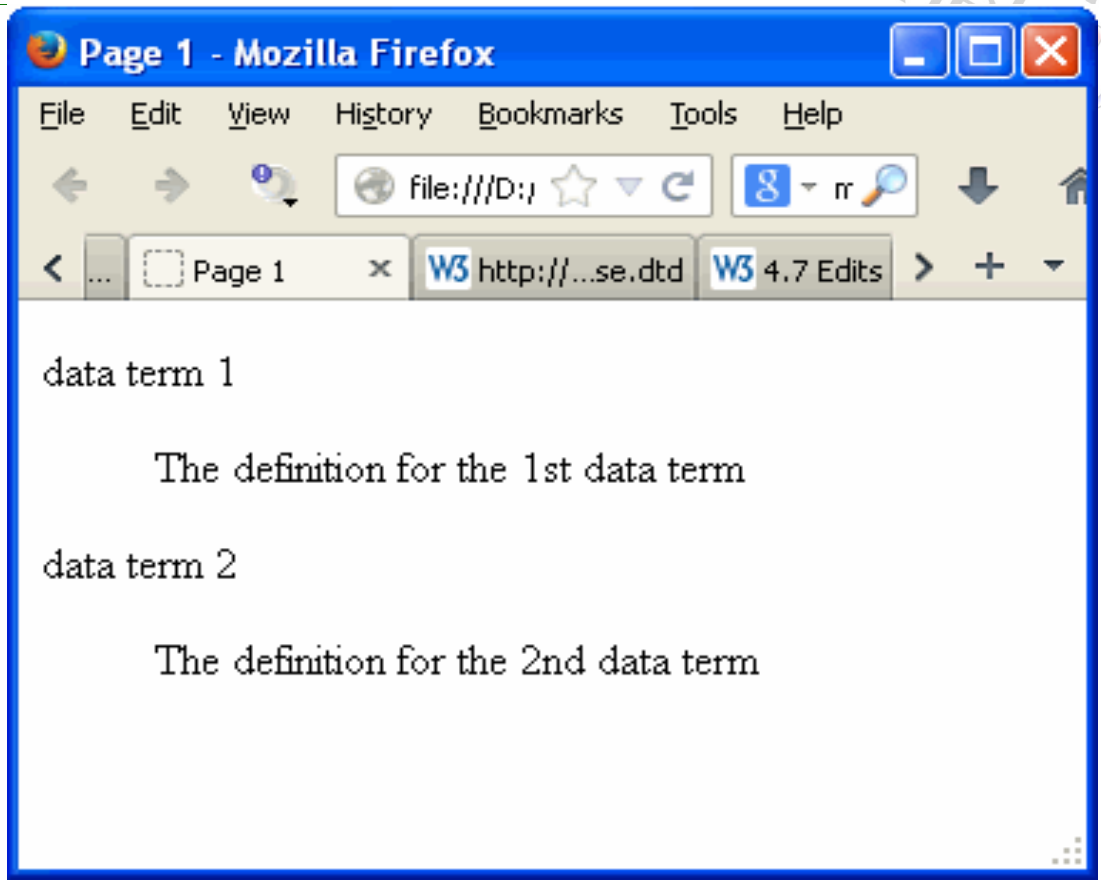
See also: `dd`, `dl`, `dt`

the whole thing:

This is what the whole set can look like:

```
<dl>
   <dt>data term 1</dt>
   <dd>
      <p>
         The definition for the 1st data term
      </p>
   </dd>
   <dt>data term 2</dt>
   <dd>
      <p>
         The definition for the 2nd data term
      </p>
   </dd>
</dl>
```

...and, on the next page, what it looks like in Firefox23 with the default stylesheet:


Page 1 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///D:/
Page 1
http://...se.dtd
4.7 Edits
data term 1
The definition for the 1st data term
data term 2
The definition for the 2nd data term

doctype – Document Preamble

Syntax:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

Description: `doctype` is a "required preamble" and is "required for legacy reasons" (HTML5 Draft☞, Sep 2013) (HTML5 itself does NOT require either *Public* nor *System* identifiers). Before HTML5 it never appears within the syntax, and is mentioned only in passing or within the Document Type Declaration (DTD) but, make no mistake, with HTML4 & later the browser will make sure that you pay for it if you miss it from the document.

Properties:

Content model:	Other
Contained in:	(n/a)
Can contain:	(n/a)
Standard:	HTML5 Draft, HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`Public identifier:`	This is ' PUBLIC ' followed by a FPI for the html version in use. In HTML5, the 'Public identifier + System Identifier' together are called an *"obsolete permitted DOCTYPE string"*, and only the four FPI's + DTDs below are allowed, to help authors transition from HTML4 and XHTML1. *FPIs:* • -//W3C//DTD HTML 4.0//EN • -//W3C//DTD HTML 4.01//EN • -//W3C//DTD XHTML 1.0 Strict//EN • -//W3C//DTD XHTML 1.1//EN
`System identifier:`	This is the URL for the DTD that refers to the html version in use. As a URL it is case-sensitive, whereas all else is case-insensitive. *DTDs:* • http://www.w3.org/TR/REC-html40/strict.dtd • http://www.w3.org/TR/html4/strict.dtd • http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd • http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd
Legacy Doctypes:	• <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> • <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd"> • <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN" • "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd"> • <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> • <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd"> • <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"> • <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

Strict vs Transitional

HTML4 represented a *big* change in working practices compared to earlier versions, and that was a challenge for developers used to working with html3, etc.. *'Transitional'* was therefore intended as a sort of halfway-house for those that found it difficult to completely change overnight, whilst *'Strict'* is how the language was intended to be.

What was the big change? Of the many changes, one of the biggest was support for *Style Sheets* (CSS); this meant moving all presentational aspects from markup to CSS. Ultimately, this makes all websites (but especially very large ones) *very* much easier to maintain. It is, however, a completely different way to work compared to html3 & earlier markup.

PS
Script & *Style* both existed in HTML3.2 Final but only as *placeholders*, ready for the next version.

dt - Definition term

Syntax: `<dt>...</dt>`

Description: `dt` is part of a useful list structure.

See also: `dd`, `dl`, `dt`

Properties:

Content model:	Block
Contained in:	`dl` elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

See the whole thing for an example of all elements put together.

em - Emphasis

Syntax: `<em>...</em>`

Description: `em` and `i` text are normally rendered the *same as each other* in visual browsers.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

fieldset - Form control group

Syntax: `<fieldset>...</fieldset>`

Description: `fieldset` is designed to group together related controls within a form; it *must* be followed by `legend`.

See also: `address`, `blockquote`, `div`, `dl`, `fieldset`, `form`, `hr`, `noscript`, `p`, `table`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	A `legend` element, followed by zero or more block-level elements and inline elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

This is what the whole set can look like:

```
<form method="post" action="/cgi-bin/order.cgi">
   <fieldset>
      <legend accesskey="i">contact information</legend>
         <table>
         ...
         </table>
   </fieldset>
</form>
```

font - Font change

Syntax: `<font>...</font>`

Description: `font` is used to cause a local change to font. It is deprecated in HTML4 in favour of `style-sheets` (and, anyway, also considered completely naff).

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level except `pre` + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Loose, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`size`	[+|-]nn e.g. size="+1", size="4"
`color`	(sRGB colours) text colour
`face`	comma-separated list of font names

form - Interactive form

Syntax: `<form action="(URI)">...</form>`

Description: `form` defines an *interactive* form. The `form` element can contain form controls (`button`, `input`, `option`, `select`, `textarea`) which allow user interaction. The user submits the form data via an `input` or a `button` element with "`type="submit"`".

See also: `address`, `blockquote`, `div`, `dl`, `fieldset`, `form`, `hr`, `noscript`, `p`, `table`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `center`, `dd`, `del`, `div`, `fieldset`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	4.01 Strict: one or more `script` or block-level elements except `form` 4.01 Loose: inline elements or block-level elements except `form`
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`accept`	list of MIME types for file upload (RFC2045)
`accept-charset`	list of supported charsets (RFC2045)
`action`	(URI) server-side form handler *Required*
`enctype`	default: application/x-www-form-urlencoded (RFC2045)
`method`	(get|post) (default: get) HTTP method used to submit the form
`name`	name of form for scripting
`onreset`	(script) the form was reset
`onsubmit`	(script) the form was submitted
`target`	render in this frame (deprecated)

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

method:

(default: “get”) In almost every circumstance this will need to be “post”. If not, the form contents are placed in the URL, and that puts severe limits on the amount of info that can be sent.

enctype:

(default: “application/x-www-form-urlencoded”) This aspect used to be a black art; these days, scripting languages such as PHP make it most simple to handle the forms server-side. This attribute is likely to be changed only if the form receives file uploads.

This is what the whole set can look like:

```
<form method="post" action="/cgi-bin/order.cgi">
 <fieldset>
 <legend accesskey="i">contact information</legend>
 <table>
 ...
 </table>
 </fieldset>
<p>
    <input type=submit value="submit order">
    <input type=reset value="clear order form">
</p>
</form>
```

frame – Frame

Syntax: `<frame>`

Description: `frame` is a rectangular region of the display, which itself is defined by `frameset`, and within which the `frame` *must* be contained. The content of the frame comes from the attribute `src`. Note that *none* of the `frame` attributes are *required*, though many have default values.

See also: `body`, `frame`, `frameset`, `head`, `iframe`, `noframes`

Properties:

Content model:	Other
Contained in:	`frameset`
Can contain:	(empty)
Standard:	HTML 4.01 Frameset

Attributes:

`core:`	- see core-attributes

`longdesc`	link to long description (intended to complement `title`)
`name`	name of frame for targetting
`src`	(URI) source of frame content
`frameborder`	(default: 1 (yes)) request frame borders?
`marginwidth`	margin widths in pixels
`marginheight`	margin height in pixels
`noresize`	(default: noresize) allow users to resize frames?
`scrolling`	(yes|no|auto) (default: auto) scrollbar or none

name:

`name` must begin with a letter (followed by a mixture of letters and/or digits).

See how the W3C frames it☞ for the full picture.

frameset - Frameset

Syntax: `<frameset>...</frameset>`

Description: `frameset` replaces the `body`.element; with this sole exception, the HTML4 Frameset `DTD` is identical to the HTML4 Transitional DTD (do notice the hint that the W3C does not want you to use Frames). Elements normally placed within a `body` element, if *preceding* the `frameset` element, will invalidate the `frameset`.

See also: `body`, `frame`, `frameset`, `head`, `iframe`, `noframes`

Properties:

Content model:	Block
Contained in:	`html`
Can contain:	One or more `frameset` & `frame` elements, as well as an optional `noframes`
Standard:	HTML 4.01 Frameset

Attributes:

`core:`	- see core-attributes

`rows`	(1 or more, comma separated: pixels or % or relative) (default: 100% (1 row)) list of lengths
`cols`	(1 or more, comma separated: pixels or % or relative) (default: 100% (1 row)) list of lengths
`onload`	(script) all the frames have been loaded
`onunload`	(script) all the frames have been removed

Frames are rectangular regions of the display. `frameset` specifies the set of frames that will constitute a full document display.

One of the problematic features of `frameset` & frames is that it allows the URI of each frame to be from different domains. This means, for example, that *your* carefully crafted pages may be shown within a screen with someone else's url in the address bar. Many people call this "theft".

`frameset`.exists for a technical as well as a practical reason; at HTML4 introduction, so many html documents were missing a body tag that they needed to specify a named-element to differentiate 4.0 Frameset docs from 4.0 Loose docs. For the same reason, `frameset` (but not `body`) tags are *Required* within a HTML4 *frameset* document.

how the W3C frames it:

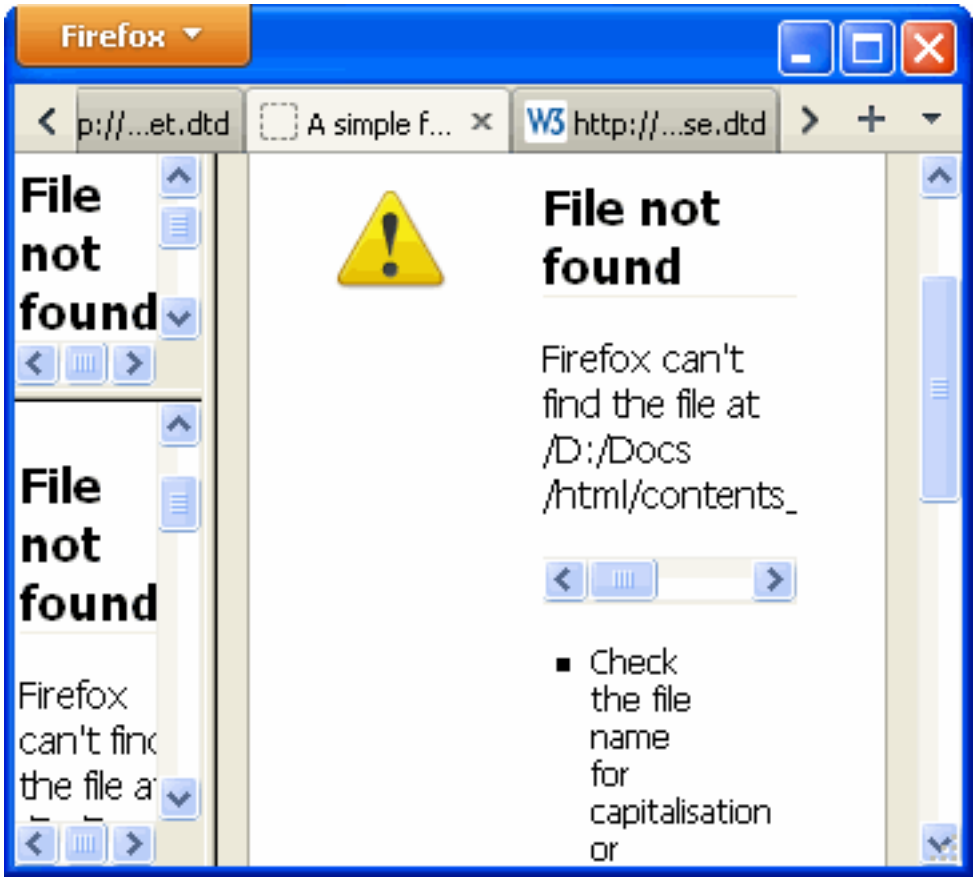
Here is html from the W3C☞:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
 "http://www.w3.org/TR/html4/frameset.dtd">
<HTML>
<HEAD>
<TITLE>A simple frameset document</TITLE>
</HEAD>
<FRAMESET cols="20%, 80%">
 <FRAMESET rows="100, 200">
 <FRAME src="contents_of_frame1.html">
 <FRAME src="contents_of_frame2.gif">
 </FRAMESET>
 <FRAME src="contents_of_frame3.html">
  <NOFRAMES>
      <P>This frameset document contains:
      <UL>
         <LI><A href="contents_of_frame1.html">Some neat contents</A>
         <LI><IMG src="contents_of_frame2.gif" alt="A neat image">
         <LI><A href="contents_of_frame3.html">Some other neat
contents</A>
      </UL>
  </NOFRAMES>
</FRAMESET>
</HTML>
```

...and how it looks in Firefox23 with default style:

Note 1: the 3 pieces of content do not exist, but you can see the separators between each `frame`.

Note 2: although it is not visible in the picture, the separators can be moved via the mouse (see `frame.noresize`) (no support).



h1 - Level-one heading

Syntax: `<h1>...</h1>`

Description: `h1` (etc.) are *headings* for a document, where 'h1' is the most-important heading, and 'h6' is the least-important. As part of learning to create readable documents, you should ensure that each html doc that you write has one (and only one) h1 heading. Other headings are then optional, but can also help make what you write more readable if carefully chosen.

See also: `h1`, `h2`, `h3`, `h4`, `h5`, `h6`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

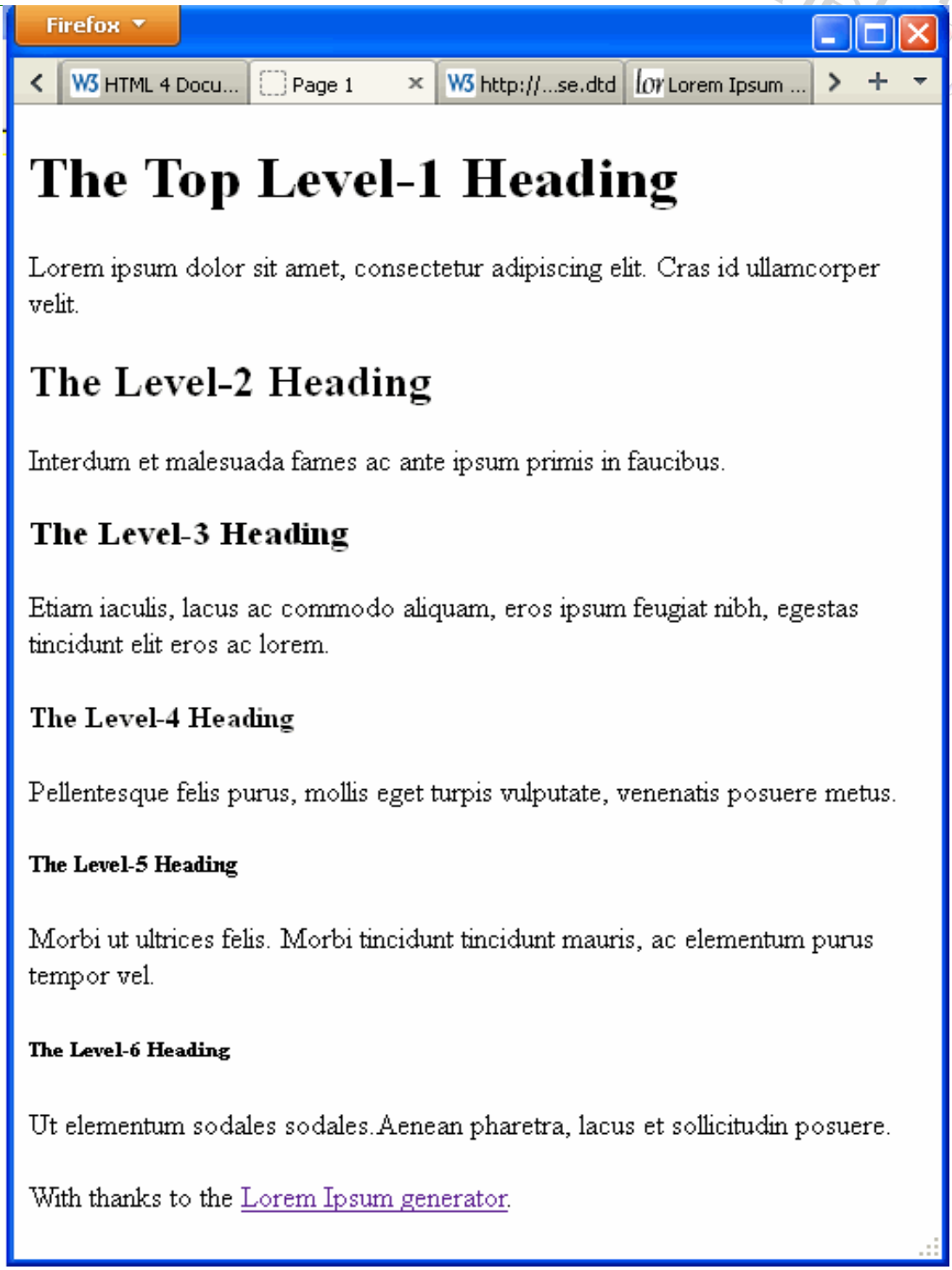
`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify) horizontal alignment (deprecated)

style:

Visual browsers use a *very* simple algorithm in how they display headings; you will almost certainly want to change it. Below is some simple html + then how it looks (default, no added style) in FireFox23.0.1:

```
<h1>The Top Level-1 Heading</h1>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras id
ullamcorper velit.</p>
<h2>The Level-2 Heading</h2>
  <p>Interdum et malesuada fames ac ante ipsum primis in faucibus.</p>
<h3>The Level-3 Heading</h3>
  <p>Etiam iaculis, lacus ac commodo aliquam, eros ipsum feugiat nibh,
egestas tincidunt elit eros ac lorem.</p>
<h4>The Level-4 Heading</h4>
  <p>Pellentesque felis purus, mollis eget turpis vulputate, venenatis
posuere metus.</p>
<h5>The Level-5 Heading</h5>
  <p>Morbi ut ultrices felis. Morbi tincidunt tincidunt mauris, ac
elementum purus tempor vel.</p>
<h6>The Level-6 Heading</h6>
  <p>Ut elementum sodales sodales.Aenean pharetra, lacus et sollicitudin
posuere.<br><br>
    With thanks to the <a href="http://www.lipsum.com/">Lorem Ipsum
generator</a>.</p>
```



You may also be interes*ted* to know that Google (and other SEs) use their investigation of a document's *headings* as part of their decision on the Page-Rank of that document.

h2 - Level-two heading

Syntax: `<h2>...</h2>`

Description: `h2` (etc.) are *headings* for a document, where 'h1' is the most-important heading, and 'h6' is the least-important. As part of learning to create readable documents, you should ensure that each html doc that you write has one (and only one) h1 heading. Other headings are then optional, but can also help make what you write more readable if carefully chosen.

See also: `h1`, `h2`, `h3`, `h4`, `h5`, `h6`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify) horizontal alignment (deprecated)

style:

Look at the bottom of `h1` to see how the 6 headers look in a browser *without* any added style.

h3 - Level-three heading

Syntax: `<h3>...</h3>`

Description: `h3` (etc.) are *headings* for a document, where 'h1' is the most-important heading, and 'h6' is the least-important. As part of learning to create readable documents, you should ensure that each html doc that you write has one (and only one) h1 heading. Other headings are then optional, but can also help make what you write more readable if carefully chosen.

See also: `h1`, `h2`, `h3`, `h4`, `h5`, `h6`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify) horizontal alignment (deprecated)

style:

Look at the bottom of `h1` to see how the 6 headers look in a browser *without* any added style.

h4 - Level-four heading

Syntax: `<h4>...</h4>`

Description: `h4` (etc.) are *headings* for a document, where 'h1' is the most-important heading, and 'h6' is the least-important. As part of learning to create readable documents, you should ensure that each html doc that you write has one (and only one) h1 heading. Other headings are then optional, but can also help make what you write more readable if carefully chosen.

See also: `h1`, `h2`, `h3`, `h4`, `h5`, `h6`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify) horizontal alignment (deprecated)

style:

Look at the bottom of `h1` to see how the 6 headers look in a browser *without* any added style.

h5 - Level-five heading

Syntax: `<h5>...</h5>`

Description: `h5` (etc.) are *headings* for a document, where 'h1' is the most-important heading, and 'h6' is the least-important. As part of learning to create readable documents, you should ensure that each html doc that you write has one (and only one) h1 heading. Other headings are then optional, but can also help make what you write more readable if carefully chosen.

See also: `h1`, `h2`, `h3`, `h4`, `h5`, `h6`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify) horizontal alignment (deprecated)

style:

Look at the bottom of `h1` to see how the 6 headers look in a browser *without* any added style.

h6 - Level-six heading

Syntax: `<h6>...</h6>`

Description: `h6` (etc.) are *headings* for a document, where 'h1' is the most-important heading, and 'h6' is the least-important. As part of learning to create readable documents, you should ensure that each html doc that you write has one (and only one) h1 heading. Other headings are then optional, but can also help make what you write more readable if carefully chosen.

See also: `h1`, `h2`, `h3`, `h4`, `h5`, `h6`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify) horizontal alignment (deprecated)

style:

Look at the bottom of `h1` to see how the 6 headers look in a browser *without* any added style.

head - Document head

Syntax: `<head>...</head>`

Description: `head` is designed to provide *header information* for the HTML document. In general, the `title` (*required*) is the only element that is seen. Curiously, and exactly like `html` and `body`, `head` is *Required* within the html object-model yet it's start & end tags are *optional*.

See also: `body`, `DTD`, `frameset`, `head`, `html`

Properties:

Content model:	Other
Contained in:	`html`
Can contain:	*Required* `title` element (just one); *optional* `base` and `isindex` elements; zero or more `link`, `meta`, `object`, `script` or `style` elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`i18n`:	- see language-attributes
`profile`	(URI) named dictionary of meta info

See also: `base`, `isindex`, `head`, `link`, `meta`, `object`, `script`, `style`, `title`

profile:

`profile` was never seriously adopted, and has been *dropped* in HTML5.

optional tags:

See the section under `html`.

hr - Horizontal rule

Syntax: `<hr>`

Description: `hr` is possibly one of the simplest html elements: it draws a horizontal rule across the page. If you want to use it for purely decorative purposes, then using *style* (eg `border-top`) on a suitable block element will be far better.

See also: `address`, `blockquote`, `div`, `dl`, `fieldset`, `form`, `hr`, `noscript`, `p`, `table`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	(empty)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right) horizontal alignment (deprecated)
`noshade`	(a solid line) (deprecated)
`size`	(pixels) height (deprecated)
`width`	(pixels or %) (deprecated)

html - HTML document

Syntax: `<html>...</html>`

Description: `html` contains the HTML *document*. Curiously, and exactly like `body`, it is *Required* within the html object-model (it is the top-level container), yet it's start & end tags are *optional*.

See also: `body`, `DTD`, `frameset`, `head`

Properties:

Content model:	Other
Contained in:	(`html` is the top-level container)
Can contain:	4.01 Strict: `head` then `body` 4.01 Loose: (same as Strict) 4.01 Frameset: `head` then `frameset`
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`i18n:`	- see language-attributes
`version`	(`DTD`) (deprecated in favour of DOCTYPE)

version:

This is a kind of `DTD` *lite* (and thoroughly deprecated in HTML4). An example is:

"-//W3C//DTD HTML 4.01 Transitional//EN"

(optional tags):

Both opening and end-tags for `html` are *optional* in HTML4 (but not xml), and you will find that to be *very* rare in this PDF.

Immediately before introduction of HTML4, the designers were faced with the issue of an existing base of tens of *millions* of html pages (that number is now *billions*) that—by definition—were not HTML4. A very great number of those documents had neither `html` nor `head` nor `body` tags.

It seems clear that the HTML4 designers opted for optional tags to give the best chance for those earlier documents to validate in HTML4 browsers. In this, as with so much else, they were thwarted by the browser designers...

Always use opening/closing tags for `html`/`head`/`body` in your own html docs. If you do not then the browser will declare your work to be *'Quirky'*, rather than '*Compliant'*, and may not give a reliable display for every component, possibly leading to a case of *screaming hab-nabs* in the author.

i - Italic text

Syntax: `<i>...</i>`

Description: `em` and `i` text are normally rendered the *same as each other* in visual browsers.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

iframe - Inline frame

Syntax: `<iframe>...</iframe>`

Description: `iframe` is used to create an inline subwindow. It is beloved of advertisers, as the space occupied is owned entirely by the domain that provides the `src`, regardless of the URL in the address bar. It is deprecated in HTML4 in favour of `object`.

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Block-level + inline elements
Standard:	HTML 4.01 Loose

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	vertical or horizontal alignment
`frameborder`	(0|1) (default 1) request frame borders?
`height`	frame height
`longdesc`	(URI) link to long description
`marginheight`	margin height in pixels
`marginwidth`	margin widths in pixels
`name`	name of frame for targetting
`scrolling`	(yes|no|auto) (default auto) scrollbar or none
`src`	source of frame content
`width`	frame width

img - Inline image

Syntax: `<img src="pic.jpg" alt="a picture">`

Description: `img` is an embedded image.

See also: a, bdo, br, img, map, object, q, script, span, sub, sup

Properties:

Content model:	Inline
Contained in:	Block-level except pre + inline elements
Can contain:	(empty)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

core:	- see core-attributes
i18n:	- see language-attributes
events:	- see event attributes

align	vertical or horizontal alignment (deprecated)	
alt	short description	*Required*
border	link border width (deprecated)	
height	(pixels or %) override height	
hspace	horizontal gutter (deprecated)	
ismap	use server-side image map	
longdesc	URI link to long description	
name	name of image for scripting	
src	URI of image to embed	*Required*
usemap	use client-side image map	
vspace	vertical gutter (deprecated)	
width	(pixels or %) override width	

alt:

The alt description will be shown whilst the image loads (or instead if it fails to load); consider also that many folks use browsers that are text only.

height:

width:

Using the height and width attributes can assist with perceived page load speeds: the browser will reserve that space for the image immediately if used, whilst it has to wait for the images to arrive to calculate the space required if not.

input - Form input

Syntax: `<input type="text" name="something">`

Description: `input` provides controls to facilitate user input, and is most often used within a `form`, although beware that some pre- HTML4 browsers will not show the control outside of a form.

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

Properties:

Content model:	Inline
Contained in:	Block-level elements, inline elements except `button`
Can contain:	(empty)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`accept`	list of MIME types for file upload (RFC2045☞)	
`accesskey`	accessibility key character	
`align`	(top|middle|bottom|left|right) vertical or horizontal image alignment (deprecated)	
`alt`	short description	
`checked`	for radio buttons and check boxes	
`disabled`		
`ismap`	use server-side image map	
`maxlength`	max chars for text fields	
`name`	submit as part of form	*Required*
`onblur`	(script) the element lost the focus	
`onchange`	(script) the element value was changed	
`onfocus`	(script) the element got the focus	
`onselect`	(script) some text was selected	
`readonly`	for text and passwd	
`size`	specific to each type of field	
`src`	for fields with images	

tabindex	position in tabbing order	
type	(button|checkbox|file|hidden|image|password|radio|reset|submit|text) what kind of widget is needed	*Required*
usemap	(URL) use client-side image map	
value	Specify for radio buttons and checkboxes	

id="...":

name="...":

The *Required* *name* attribute of every input element is sent to the server on form submission as part of a set of name/value pairings; the 'name' part of the pair comes from the input's *name* attribute, whilst the value depends on both the type of input and the user's input to that control.

namespaces:

The html writer needs to bear in mind both that *name* & *id* share the same name-space, and also that each needs to be unique within the document (each form control in the document also needs to have a unique name). The easiest way to do this is to specify both when defining the control.

type="button":

type="reset":

type="submit":

button is a push-button for use with client-side scripting. value gives the button text-label whilst, typically, onclick would be used to define the script action. The two snippets below achieve the same result; the second avoids a non-functioning button for those with JavaScript disabled (the "explode()" function would also need to be defined in an earlier script element):

```
<input type="button" value="Blow up your computer" name="explode"
id="explode" onclick="explode()">


<script type="text/javascript">
<!--
  document.write("<input type=\"button\" value=\"Blow up your computer\""
                 + "name=\"explode\" id=\"explode\""
                 + "onclick=\"explode()\">");
// -->
</script>
```

reset + submit will also create buttons with those default actions; the *value* attribute is optional, and will over-ride the default browser text for the button. As with other controls, the *name* attribute will determine what value is sent to the server when the button is pressed; this allows multiple 'submit' buttons to placed on a form, and for different actions to be taken according to which one is pressed.

type="checkbox":

type="radio":

checked:

value:

`checkbox` and `radio` controls share all characteristics with one crucial difference: only one *`radio`* control can be selected at a time, whilst *any* *`checkbox`* control may be selected. Each is placed in a group by specifying the same *`name`* atribute; note that they will be received at the server in an *array* which has the same base name-element as the *`name`* attribute (in the example below as "`ice_cream[]`"). The value returned is that of the `value` attribute for that specific control (empty for non-*`checked`* controls).

Some older browsers will insist that one of a radio group is selected at all times. It is good practice, in any case, to always define one of a radio group as *`checked`*.

Here is a typical usage:

```
<p>Please indicate your favourite ice-cream:</p>
<p>
<label accesskey="S"><input type="radio" name="ice_cream"
value="strawberry"><span style="text-
decoration:underline;">S</span>trawberry</label><br>
<label accesskey="C"><input type="radio" name="ice_cream"
value="chocolate"><span style="text-
decoration:underline;">C</span>hocolate</label><br>
<label accesskey="V"><input type="radio" name="ice_cream"
value="vanilla"><span style="text-
decoration:underline;">V</span>anilla</label><br>
<label accesskey="A"><input type="radio" name="ice_cream" value="all"
checked="checked"><span style="text-decoration:underline;">A</span>ll of
them!</label>
</p>
```

type="file":

accept:

(one of the dangerous controls – hackers love to discover if the html writer has left holes in their security with controls such as this)

This is intended to create a method of uploading a file to the server. `value` is intended to specify the name of the initial file, yet is ignored by many browsers. `accept` is supposed to be a comma-separated list of acceptable media types, and is also ignored by many browsers. The specific `method` and `enctype` below are *Required* for this type of action:

```
<form method=post action="/dev/null" enctype="multipart/form-data">
<p>Select a document to upload for our urgent attention.</p>
<p><input type="file" name="file" accept="text/plain"></p>
<p><input type="submit" value="hurry!"></p>
</form>
```

type="hidden":

`hidden` controls are not displayed but, in all other respects, are treated as normal controls; this means that their *name+value* is submitted together with all other controls. They are therefore often used to carry information across between several linked forms.

It is important to realise that `hidden` controls *can* be viewed by anyone that looks at the html source.

```
<input type="hidden" name="oliver" value="is very short">
<input type="hidden" name="rebuttal" value="You are ugly, whilst I am
young and will grow">
```

type="image":

alt, usemap + src:

`image` defines a *graphical* submit button; `src` is then *Required* as the URL for the image and `alt` (new in HTML4) as replacement text for those not loading images. Pre- HTML4 browsers often use `name` or `value` as the replacement for `alt`, which means specifying all 3 to remain backwards compatible! Authors will probably want to use `button` instead, which can easily have an image specified for it.

When clicked, the co-ordinates of the click are sent together with the form submission as *`name.x=x-value`* and *`name.y=y-value`*, where *name* is the value of the `name` attribute, *`x-value`* is the click's pixels from the left of the image, and *`y-value`* is the click's pixels from the top of the image. The *`usemap`* attribute combined with this `type` defines a client-side image map that can be used with client-side scripting. The *`usemap`* attribute gives the url of the defining *`map`*.

Phew!

type="password":

type="text":

maxlength + size:

`password` is a specialised form of `text` control in which characters input by the user are masked (often as asterisks) – that is the sole difference. Note, however, that—unless the page is secure (sent under SSL encryption, shown by a "https" prefix)—that all form values are sent in clear text, including `password` controls.

`text` is the default form of `input` control, and specifies a single-line text-input control. Any *`value`* attribute will specify the default text within the control after creation, whilst *`size`* will be a suggestion for the size of the control and `maxlength` the maxinum number of characters that it will be allowed to contain. See also `textarea` for the multi-line text input control.

Important note: it is simplicity itself for a user to evade the restrictions enforced by browsers with `maxlength`. It is therefore important for form-handling script authors to enforce size-limits server-side.

Here is a simple username/password input routine:

```
<p>
<label accesskey="U"><span style="text-decoration:underline;">U</span>ser
name: <input type="text" name="username" size=8 maxlength=8></label><br>
<label accesskey="P"><span style="text-
decoration:underline;">P</span>assword: <input type="password" name="pw"
size=12 maxlength=12></label>
</p>
```

ins - Inserted text

Syntax: `<ins>...</ins>`

Description: `ins` is normally rendered in a visual browser in underlined text, and is intended to indicate text inserted *in this version of the document* It is unusual in being capable of being either a block-level *or* an inline element. If used in an inline context—for example, within a paragraph—it cannot then contain any block-level elements.

See also: `ins`, `del`

Properties:

Content model:	Inline *or* Block
Contained in:	Block-level elements, inline elements
Can contain:	Block-level elements, inline elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`cite`	(URI) info on reason for change
`datetime`	date and time of change (ISO format: "YYYY-MM-DDThh:mm:ssTZD")

cite:

datetime:

title:

`cite` + `datetime` are optionally available to give a url for readers to obtain further information on the reason for the change (`cite`) and the time & date of that change (`datetime`). W3C states☞ that "*they are primarily intended for private use (e.g. by server-side scripts collecting statistics about a site's edits), not for readers*". A simpler method to give succinct info is via the `title` attribute (see core-attributes).

isindex - Input prompt

Syntax: `<isindex>`

Description: `isindex` is a single line *prompt* (an *input* field, equivalent to:
a `form` with a single `input` of "`type="text"`"
a "`method="get"`"
an `action` pointing to the url of the containing document
). `isindex` is deprecated in HTML4 in favour of the `input` element.

See also: `base`, `isindex`, `head`, `link`, `meta`, `object`, `script`, `style`, `title`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `center`, `dd`, `del`, `div`, `fieldset`, `iframe`, `head`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	(empty)
Standard:	HTML 4.01 Loose, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`prompt`	prompt message (effectively, the label)

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

kbd - Text to be input

Syntax: `<kbd>...</kbd>`

Description: `kbd` text is normally rendered in a visual browser as `monospaced text`.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

label - Form field label

Syntax: `<label>...</label>`

Description: `label` is a main usability aspect of `form`. The key feature is that the `label` must NOT contain (or be associated with) more than one field.

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

Properties:

Content model:	Inline
Contained in:	Block-level elements, inline elements except `button`
Can contain:	Inline elements except `label`
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`accesskey`	accessibility key character
`for`	matches field ID value
`onblur`	(script) the element lost the focus
`onfocus`	(script) the element got the focus

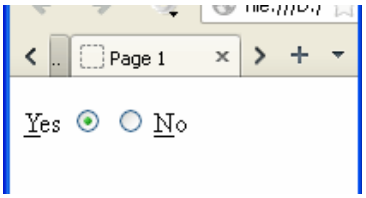
The association between `label` and a control can be implicit (the label directly contains the field) or explicit – this latter is achieved by the `for` attribute (below). The association thus created then assists speech-browser users whilst also allowing visual browser GUI features: eg clicking on the label will select the associated radio/checkbox control.

Implicit association:
Below, each of 2 labels contain a *radio* control, the 1st for "Yes" and the 2nd for "No". Each label is therefore *implicitly* associated with the label that they contain. A snapshot of what this html code produces within FireFox is below the code.

The *accesskey* will allow the radios to be controlled via the keyboard. In addition, hovering the mouse cursor above either text will highlight the associated control, and a click will select it.

Note: these are all *inline* elements. Therefore, even though the opening label element, text, radio control and closing label element are all (deliberately) on different lines, the whole thing is displayed in the browser on the same line:

```
<p>
<label accesskey="Y">
<span style="text-decoration:underline;">Y</span>es
<input type="radio" name="ice_cream" value="1" checked="checked">
</label>
<label accesskey="N">
<input type="radio" name="ice_cream" value="0">
<span style="text-decoration:underline;">N</span>o
</label>
</p>
```

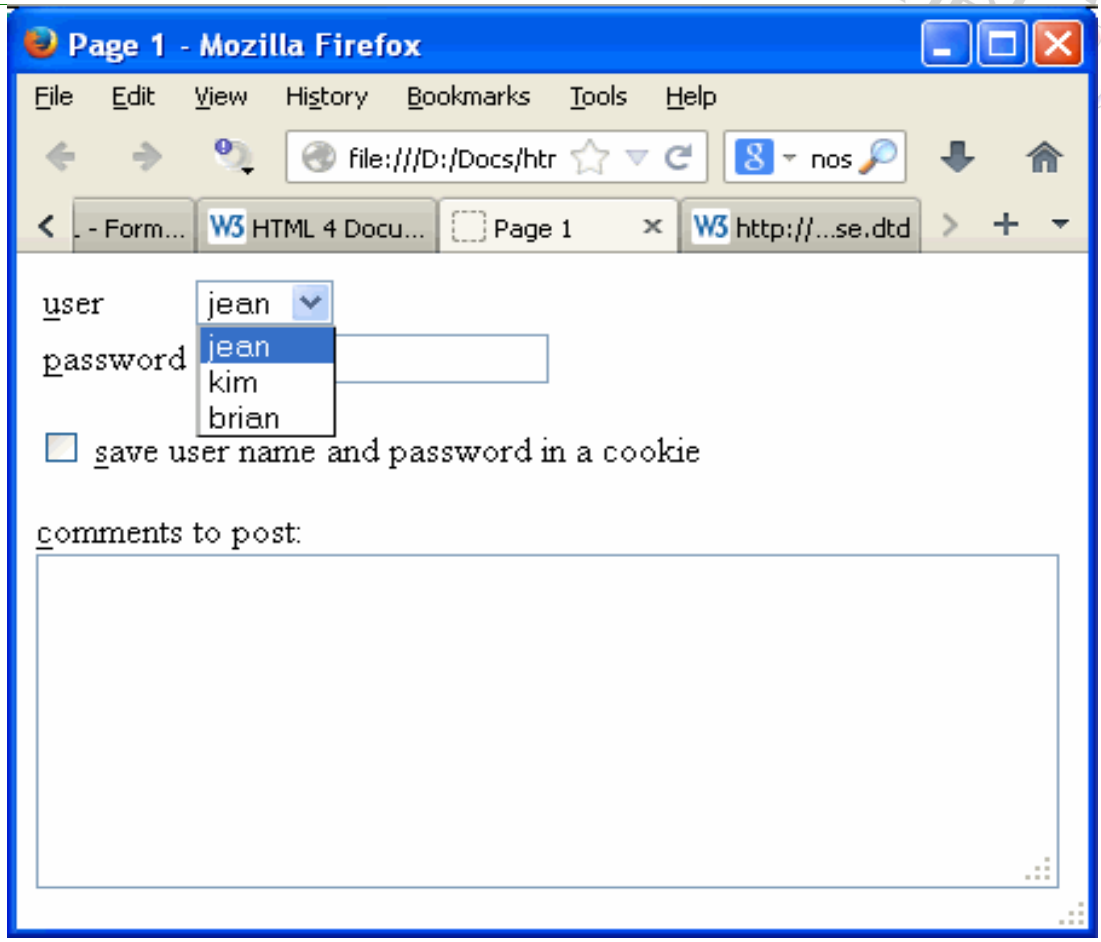


for:

id:

Explicit association:
It is very common for the label to be separated from the control; in that situation, the `for` attribute in the label is used to link with the `id` attribute in the control ('`id`' is one of the core-attributes for every element). Below is a table arrangement, with the results shown below the html:

```
<table><tr>
  <td>
    <label for="user" accesskey="u">
      <span style="text-decoration:underline;">u</span>ser
    </label>
  </td>
  <td>
    <select name="user" id="user">
      <option>jean</option>
      <option>kim</option>
      <option>brian</option>
    </select>
  </td>
</tr><tr>
  <td>
    <label for="passwd" accesskey="p">
      <span style="text-decoration:underline;">p</span>assword
    </label></td>
  <td><input type="password" name="password" id="passwd"></td>
</tr></table>
<p>
  <label accesskey="s">
    <input type="checkbox" name="save" value="yes">
    <span style="text-decoration:underline;">s</span>ave user name and
password in a cookie
  </label>
</p>
<p>
  <label accesskey="c">
    <span style="text-decoration:underline;">c</span>omments to post:
    <textarea name="comments" rows=8 cols=50></textarea>
  </label>
</p>
```


Page 1 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///D:/Docs/htr
nos
- Form...
HTML 4 Docu...
Page 1
http://...se.dtd
user
jean
password
jean
kim
brian
save user name and password in a cookie
comments to post:

legend - Fieldset caption

Syntax: `<legend>...</legend>`

Description: `legend` gives the caption for a group of related controls within a form; it *must* be the first item of a `fieldset`.

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

Properties:

Content model:	Block
Contained in:	`fieldset`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`accesskey`	accessibility key character
`align`	(top|bottom|left|right) relative to fieldset (deprecated)

This is what it can look like:

```
<form method="post" action="/cgi-bin/order.cgi">
   <fieldset>
      <legend accesskey="i">contact information</legend>
         <table>
         ...
         </table>
   </fieldset>
</form>
```

li - List item

Syntax: `<li>...</li>`

Description: `li` defines a *list* item, and is the element that contains the actual content.

See also: `li`, `ol`, `ul`

Properties:

Content model:	Block
Contained in:	`ol`, `ul`
Can contain:	Inline elements, block-level elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`type`	(both OL + UL types) (deprecated)
`value`	(number) start number for the sequence (deprecated)

type:

value:

Both govern the *style* of presentation of the bullet, and thus are deprecated in HTML4.01 Strict. `type` must be one of the `ol.type` if contained in an `ol`, and one of the `ul.type` if contained in a `ul`. Likewise, value only makes sense if the `li` is contained within an `ol`.

See the whole thing for a set of nested `li` elements in a browser.

link - Document relationship

Syntax: `<link>`

Description: `link` (zero, one or many) defines document *relationships*. See bottom for discussion & examples.

See also: `base`, `isindex`, `head`, `link`, `meta`, `object`, `script`, `style`, `title`

Properties:

Content model:	Other
Contained in:	`head`
Can contain:	(empty)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`charset`	char encoding of linked resource (RFC2045)
`href`	(URI) URI for linked resource
`hreflang`	language code (RFC1766)
`media`	(1, or comma-separated list) for rendering on these media
`rel`	(*LinkTypes*) forward link types
`rev`	(*LinkTypes*) reverse link types
`type`	advisory content type (RFC2045)

rel:

rev:

The W3C authors had a *book* type of format in mind when these two attributes were put in place. A handful of examples using some *LinkTypes* originally defined in HTML4 should help explain that better:

- `<link rel="prev" href="page6182.html" title="My Sister's Annoying habits">`
- `<link rel="next" href="page6184.html" title="My Sister's Annoying habits">`
- `<link rel="start" href="page1.html" title="My Sister's Annoying habits">`
- `<link rel="index" href="index.html" title="My Sister - Index">`

(cont. next page)

This above kind of *LinkTypes* (note: case-insensitive) are almost entirely ignored by visual browsers and, in my experience, only serve to uselessly inflate the document size whilst providing lots of urls for hi-speed scrapers; my general advice would be to ignore them. That advice completely changes with regard to specific *LinkTypes* which are required for specific purposes:

rel=apple-touch-icon:

rel=shortcut icon:

These were introduced by Microsoft for MSIE with the *'shortcut icon'* (and break the W3C stipulation that *LinkTypes* should not contain a space) (*doh!*). Internet Explorer was ubiquitous at the time, web-server error_logs were filling up with 404s for missing *'favicon.ico'* files (intended to be placed at the root of the web-space, or in a specified location) and, essentially, everyone jumped on board, so it is now a standard. If the *favicon* was not specied in the `head` browsers would try to find the file—default name: 'favicon.ico'—in *every* wretched sub-directory, so I advise a belt 'n' braces action, with a file called 'favicon.ico' in the root of the web-space & also a `rel` element specifying an *absolute* URI for where & what it is.

Apple have taken lessons from MS in non-compliance and added 'sizes' as an extra attribute for use under iOS (iPhone, iPad, iPod, iBall, etc.) (default: 57x57), but at least their *favicon* is a PNG file (a well-supported graphic format), whilst MS went for ICO (another, but poorly-supported, graphic format – try a favicon generator site☞). To rub salt in the wound, Apple will search for four different filenames☞ (I've just checked & yup, 2 of those are in my error logs; grr). 'apple-touch-icon.png' is the most widely-used default name. Advice is the same as for the *favicon*.

Here are two examples:

```
<link rel="shortcut icon" href="favicon.ico" type="image/x-icon">
<link rel="apple-touch-icon" href="apple-touch-icon.png" type="image/png">
```

Notes: the `href` above is a *relative* URL, which will rely upon a `base` element being used for accurate translation by the browser (see absolute v's relative URIs) (or better, as you probably do not want a *favicon* in every directory, enter an *absolute* URI).

rel=StyleSheet:

rel=alternate stylesheet:

Coupled together with a `meta` element, this is the advised method for including an external *StyleSheet* in to the document. Here is an example:

```
<meta http-equiv="Content-Style-Type" content="text/css">
<link rel="StyleSheet" href="style.css" type="text/css"
media="screen,projection,print">
```

Notes: the `href` above is a *relative* URL, which may rely upon a `base` element being used for accurate translation by the browser (see absolute v's relative URIs) (or better, as you probably do not want a style document in every directory, enter an *absolute* URI). See also attaching style to HTML.

map – Image map

Syntax: `<map name="mapname">...</map>`

Description: `map` is a client-side image map. The *Required* `name` attribute is used as an anchor for the `usemap` attribute of an `img` or `object`.

HTML4 has extended the `map` element so that it can contain one or more block-level elements—such as `object`—in addition to the `area` element.

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	One or more block-level, or one or more `area` elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`name`	for reference by usemap	*Required*

Here is a typical pre- HTML4 usage of `map`:

```
<map name="mymap">
<area href="/reference/" alt="html and css reference" coords="5,5,95,195">
<area href="/design/" alt="design guide" coords="105,5,195,195">
<area href="/tools/" alt="tools" coords="205,5,295,195">
</map>
<img src="sitemap.gif" alt="site map" usemap="#mymap" width=300
height=200>
```

menu - Menu list

Syntax: `<menu>...</menu>`

Description: `menu` is an *unordered*.list of items for a menu (deprecated in HTML 4 Strict in favour of `ul`). Note that the contained `li` elements are NOT allowed to contain block-level elements, which prevents cascading menus.

See also: `dir`, `li`, `menu`, `ol`, `ul`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	One or more `li` elements (which cannot contain block-level elements)
Standard:	HTML 4.01 Loose, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`compact`	display in a compact style (deprecated)

compact:

`compact` is poorly supported by browsers.

meta – Metadata

Syntax: `<meta>`

Description: `meta` (zero, one or many) defines *generic metainformation*. The W3C never defines what this might be.

See also: `base`, `isindex`, `head`, `link`, `meta`, `object`, `script`, `style`, `title`

Properties:

Content model:	Other
Contained in:	`head`
Can contain:	(empty)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`i18n`:	- see language-attributes

`content`	associated information	*Required*
`http-equiv`	HTTP response header name	
`name`	metainformation name	
`scheme`	select form of content (rarely used)	

Certain `meta` elements have become *de rigueur* in every html document:

content:
name:

`name`/`content` are the classic *name/value* pairings found throughout this PDF. `content` can contain text or entities but never HTML tags. The following are typical but never required:

name=author:

Everyone likes their work to be recognised.

name=description:
name=keywords:

SEs ("Search Engines") will often use all/part of the *description* in the SERPs ("Search Engine Results Pages"). Keep it concise.

Keywords (comma-separated values) have become much abused across the years; SEs will penalise the page if you practise keyword-stuffing.

name=robots:

You need a file called '*robots.txt*☞' in the root of your web-space. Please be aware that there are bots that give a little back after scraping your site ("goodbots") & others that are scraping your site for their sole benefit☞ ("badbots"). In particular, using *robots.txt*☞ to detail all the sensitive directories on your site is a gift to the *badbots*.

'*robots.txt*' is a site-wide declaration for general and/or specific bots. If you wish to make a page-specific declaration it can be done via the `name` attribute (note: the page needs then to be accessible to the bots):

- `<meta name="robots" content="...,...">`

'robots' declares that this declaration is for all bots. Use the bot-specific name to direct it at a specific bot (try robotstxt.org☞ for a list).

`content` (comma-separated, case-insensitive values) are the directives. There are zero directives in law; what exist are through convention & usage. The following are generally agreed, and for best in one statement:

- all
  = "index, follow"
  no restrictions; robots consider this to be the default
- noindex
  noshow in SERPs + no 'cached' link
- nofollow
  do not follow the links on this page
- none
  = "noindex, nofollow"

http-equiv:

(The value is case-sensitive)
HTML documents transmitted across the Internet are sent via the HTTP protocol ("*Hypertext Transfer Protocol*"). HTTP packets have a Header+Body format just like the HTML documents that they contain. The document itself is placed within the body of those packets, whilst the header contains important *metadata* about content within the body. A Web Browser makes use of both HTTP headers & body in making decisions on how it will render the HTML document into a web-page.

Ideally, an HTML writer needs to have control over both parts of the HTTP packets; the `meta` element + `http-equiv` attribute offers a possibility of doing this even if—as one example—the HTML document is being sent via email. Some web-servers can be configured to auto-send HTTP headers that are contained within the `meta` element, whilst most browsers will also take account of this element. Even if yours is a dedicated server, this author advises a belt 'n' braces approach (send headers + include `meta` statements) as long as the two do not conflict.

http-equiv=Content-Type:

This is one of the valuable declarations, but can have a downside if the *charset* is different to the browser default, in which case it may re-load the page (one of many where the HTTP headers are essential). Here is an example:

- <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

http-equiv=Content-Script-Type:

This is necessary to include *inline* JavaScript script.

- <meta http-equiv="Content-Script-Type" content="text/javascript">

http-equiv=Content-Style-Type:

This is necessary to include *inline* style. See also link rel=StyleSheet for loading an *external* CSS file:

- <meta http-equiv="Content-Style-Type" content="text/css">

http-equiv=Refresh:

This can be controversial, messes up the browser history (and thus the 'Back' button), has been much misused in the past and can also lead to the SEs penalising the page (particularly for short-duration refreshes).

It will cause the page to be refreshed after the interval marked (in seconds), optionally loading a completely different page (making the first page a kind of "splash screen"). You should note that most modern browsers default to *"no-refresh"*:

- <meta http-equiv="Refresh" content="10; url=/another.html">

noframes - Frames alternate content

Syntax: `<noframes>...</noframes>`

Description: `noframes` is designed for content to be shown when *frames* cannot be shown (in this way it is the *frames* equivalent of `noscript`). The W3C advice is to *always* have a `noframes` section in a Frameset document.

See also: `body`, `frame`, `frameset`, `head`, `iframe`, `noframes`, `noscript`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	4.01 Loose: block-level elements, inline elements 4.01 Frameset: one `body` element, which must NOT contain any `noframes` elements
Standard:	HTML 4.01 Frameset

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

how to hack-off your users:

All the Frameset elements are new in HTML4, which meant at HTML4 introduction that all previous (html3.2 & earlier) browsers did not present the *frames* correctly. Many html authors put a message in `noframes` saying: *"Upgrade your browser"*. Some of the new HTML4 browsers made it possible for their users to switch frames-display *off* meaning that, for these folks, *"Upgrade your browser"* was all that they saw. Now, make a careful note of this; should you ever want to utterly wind up your user-base, that is a perfectly wonderful way to do it.

a guide to content:

A set of links to the other frames (`frame.src` URLs) is probably enough.

noframes in 4 Transitional:

Probably a good idea, if only the browser will support it (it will always show the `noframes` content if not).

See how the W3C frames it for the full picture.

noscript - Alternate script content

Syntax: `<noscript>...</noscript>`

Description: `noscript` is... odd.

See also: `script`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Block-level elements, inline elements
Standard:	HTML 4.01 Strict

The W3C describes `noscript` as an *"alternate content container for non script-based rendering"*. It is customary to place it within the html immediately following any `script` element, but note that there is zero connection within the DTD between the two; effectively, that means that there should only be one pair of `noscript` tags within a document. Today, most html authors appear to use this element to warn the visitor that they will not see anything on the page unless scripting is enabled in their browser.

One issue arises if the browser has scripting *enabled* but does not support the specific language, when browsers may fail to support `noscript` (a browser with client-side scripting *disabled* should be unaffected).

object – Object

Syntax: `<object>...</object>`

Description: `object` is a generic embedded object. It was intended to replace the img + `applet` elements & particularly proprietary elements such as the Netscape *embed* or Microsoft *bgsound*.

See also: `param`
See also: `base`, `isindex`, `head`, `link`, `meta`, `object`, `script`, `style`, `title`
See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	One or more block-level, or one or more `area` elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	vertical or horizontal alignment (deprecated)
`archive`	space-separated list of URIs
`border`	link border width (deprecated)
`classid`	identifies an implementation
`codebase`	base URI for classid, data, archive
`codetype`	content type for code
`data`	reference to object's data
`declare`	declare but don't instantiate flag
`height`	(pixels or %) override height
`hspace`	horizontal gutter (deprecated)
`name`	submit as part of form
`standby`	message to show while loading
`tabindex`	position in tabbing order
`type`	content type for data
`usemap`	use client-side image map
`vspace`	vertical gutter (deprecated)
`width`	(pixels or %) override width

ol - Ordered list

Syntax: `<ol>...</ol>`

Description: `ol` is an *ordered*.list of items. A browser will normally supply the numbering for the content.

See also: `li`, `ol`, `ul`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	One or more `li` elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`compact`	display in a compact style (deprecated)
`start`	(number) (default: 1) start number for the sequence (deprecated)
`type`	(1|a|A|i|I) style for bullet (deprecated)

type:

`type` governs the *style* of presentation of the bullet allotted to the contained `li` elements, and thus is deprecated in HTML4.01 Strict. The value is a single digit, and these are interpreted as follows:

1	arabic numbers	1, 2, 3, ...
a	lower alpha	a, b, c, ...
A	upper alpha	A, B, C, ...
i	lower roman	i, ii, iii, ...
I	upper roman	I, II, III, ...

See the whole thing for a set of nested `li` elements in a browser.

optgroup - Option group

Syntax: `<optgroup label="(text)">...</optgroup>`

Description: `select`, `optgroup` and `option` together are intended to create a cascading set of menus.

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

Properties:

Content model:	Inline
Contained in:	`select`
Can contain:	One or more `option` elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes	
`i18n:`	- see language-attributes	
`events:`	- see event attributes	
`disabled`		
`label`	for use in hierarchical menus	*Required*

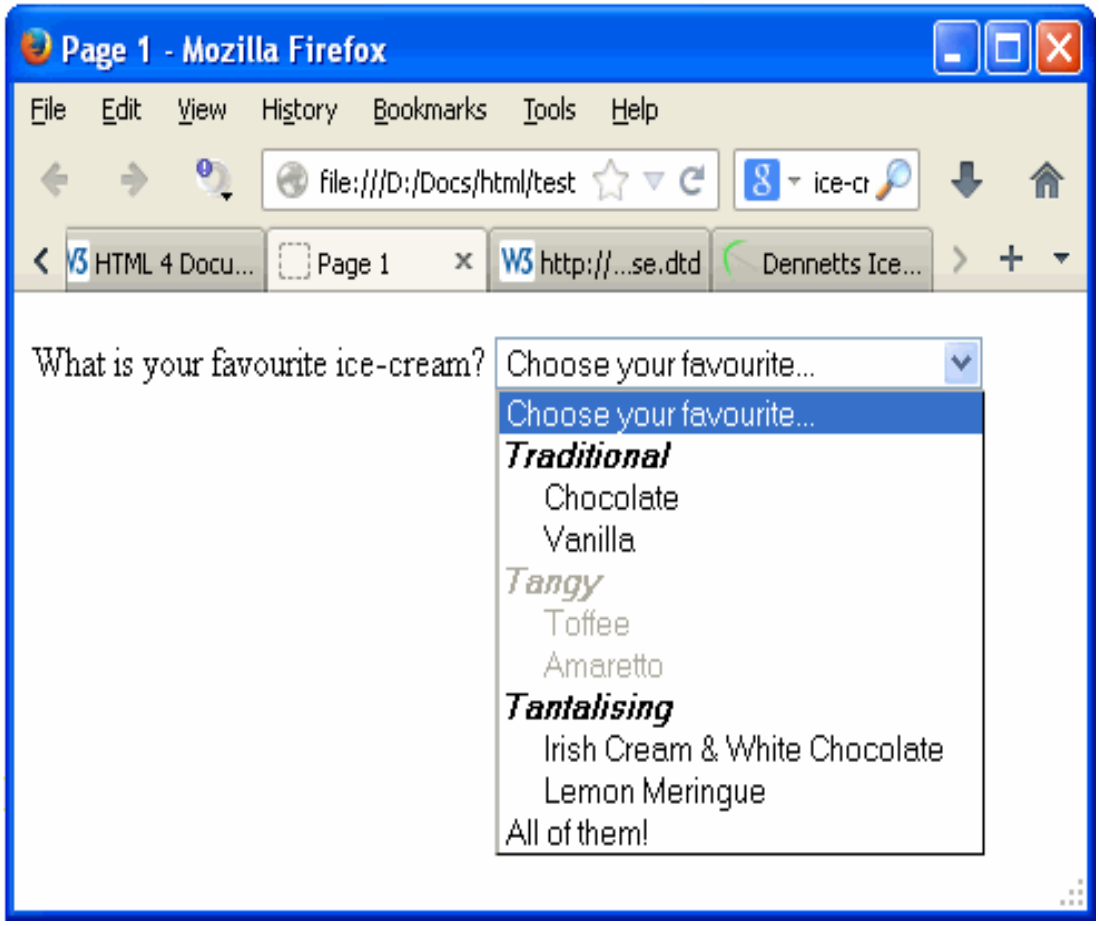
See next page for a demonstration...

label:

`label` is *Required* on `optgroup` but optional on `option`. The easiest way to explain these is to demonstrate it (as below). Note the use of `selected` with an empty value to provide a default heading to the menu group:

```
<p>What is your favourite ice-cream?
  <select name="ice_cream">
    <option selected="selected" value="">Choose your favourite...</option>
    <optgroup label="Traditional">
      <option value="choc">Chocolate</option>
      <option value="vanilla">Vanilla</option>
    </optgroup>
    <optgroup label="Tangy" disabled="disabled">
      <option value="toffee">Toffee</option>
      <option value="amaretto">Amaretto</option>
    </optgroup>
    <optgroup label="Tantalising">
      <option value="irish">Irish Cream &amp; White Chocolate</option>
      <option value="lemon">Lemon Meringue</option>
    </optgroup>
    <option>All of them!</option>
  </select>
</p>
```

...and the result in a browser:



...with many thanks to Dennetts of Spilsby☞ for some delicious suggestions!

option - Menu option

Syntax: `<option>...</option>`

Description: `option` is the menu option(s) available within a menu.

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

Properties:

Content model:	Inline
Contained in:	`select`
Can contain:	One or more `option` elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`disabled`	
`label`	for use in hierarchical menus
`selected`	default selected value
`value`	defaults to element content

selected:

The option marked as `selected` will be pre-selected at menu construction. Sometimes, you do not want any option to be selected (allowing a possible NULL response) and, in addition, may want to give a short hint at the top of the menu. Using "`selected value=""`" will achieve that.

value:

On form submission, the `value` of the selected option will be posted. If `value` is missing, then the (text) content of the option is sent instead.

See the page following `optgroup` for a full demonstration...

p – Paragraph

Syntax: `<p>...</p>`

Description: `p` defines a *paragraph* element. One curiosity is that it can only contain inline elements, and another is that the end-tag is optional with html (although that can then lead to stylesheet bugs with some browsers).

See also: `address`, `blockquote`, `div`, `dl`, `fieldset`, `form`, `hr`, `noscript`, `p`, `table`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify) horizontal alignment of content (deprecated)

param - Object parameter

Syntax: `<param name="(the_name)">`

Description: `param` is a *"named property value"*. It is designed to provide parameter values to the `applet` or `object` element that contains it, and MUST be the first content within that element. `id` / `name` is the name of the parameter, whilst `value` is it's value.

See also: `object`

Properties:

Content model:	Other
Contained in:	`applet`, `object`
Can contain:	(empty)
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`id`	document-wide unique id (also a core-attribute)
`name`	property name
`type`	content type for value when `valuetype="ref"` (RFC2045☞)
`value`	property value
`valuetype`	(data|object|ref) (default: data) How to interpret `value`

id:

name:

The html author needs to bear in mind both that *`name`* & *`id`* share the same name-space, and also that each needs to be unique within the document. The simplest method is *always* to specify both whenever either are used.

pre - Pre-formatted text

Syntax: `<pre>...</pre>`

Description: `pre` contains preformatted text. Importantly, `pre` does NOT collapse whitespace and (unfortunately) will not wordwrap (that latter can be overcome with some style *black magic*) which can lead to side-scanning if the screen width is insufficient.

Under the hood, several inline elements are excluded, preventing any markup that will involve changes in font size or inclusion of images.

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	Inline elements except `applet`, `basefont`, `big`, `font`, `img`, `object`, `small`, sub, `sup`
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes
`width`	expected line length (deprecated) (ignored by browsers)

width:

`width` is ignored by browsers and, in any case, deprecated in HTML4 Strict. If that was not enough, it is a `NUMBER`, which immediately says *"of what?"* (pixels, em, en, inches, ...) (probably characters, but who knows? as it is never specified).

q - Short Quotation

Syntax: `<q>...</q>`

Description: `q` text is intended to be a short, *inline* quotation; use blockquote for longer, *block-level* quotations.

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`cite`	URI of the source of the quotation.

s - Strike-through text

Syntax: `<s>...</s>`

Description: `s` is normally rendered in a visual browser in a ~~strike-through style~~.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + Inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Loose (deprecated in HTML4)

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

See also `strike`

samp - Sample output

Syntax: `<samp>...</samp>`

Description: `samp` text is normally rendered in a visual browser as `monospaced text`.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

script - Client-side script

Syntax: `<script type="text/javascript">...</script>`

Description: `script` is intended to include *client-side script(s)* in the document.

See also: `base`, `isindex`, `head`, `link`, `meta`, `object`, `script`, `style`, `title`

Properties:

Content model:	Inline
Contained in:	`head`, block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`charset`	char encoding of linked resource (RFC2045)	
`defer`	UA may defer execution of script	
`language`	predefined script language name (deprecated)	
`src`	URI for an external script	
`type`	content type of script language (RFC2045)	*Required*

See also: `noscript`
See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Client-side scripts are used (and mis-used) to create greater utility within html documents. They are also the single greatest vector for abuse of multiple kinds. For that reason, the author would strongly advise you to make use of noscript☞ in your browser, and also to bear in mind that the user may well prevent use of Script in *their* browser.

select - Option selector

Syntax: `<select>...</select>`

Description: `select` defines a form control (menu) for the selection of options. Before HTML4 these *had* to be defined within a form.

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

Properties:

Content model:	Inline
Contained in:	Block-level elements, inline elements except `button`
Can contain:	One or more `optgroup` or `option` elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`disabled`	
`multiple`	default is single selection
`name`	field name
`onblur`	(script) the element lost the focus
`onchange`	(script) the element value was changed
`onfocus`	(script) the element got the focus
`size`	# rows visible
`tabindex`	position in tabbing order

multiple:
name:

The default is for just one option allowed to be selected. On submission, name/value pairs are sent: `name` is used for the name, whether as a single or as multiple pairs; see `option` for the value part.

size:

`size` suggests to visual browsers how many options should be on-screen at any one time, with a scroll-bar to get access to any others.

See the page following `optgroup` for a full demonstration...

small - Small text

Syntax: `<small>...</small>`

Description: `small` is normally rendered in a visual browser in a smaller font size. It is also possible to nest this element, which can lead to unpredictable results, as you cannot be certain of the reader's starting font size.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level except pre + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

span - Generic inline container

Syntax: `<span>...</span>`

Description: `span` is typically used to apply *style* to inline elements; see `div` for the identical application for block-level elements.

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

strike - Strike-through text

Syntax: `<strike>...</strike>`

Description: `strike` is normally rendered in a visual browser in a ~~strike-through style~~. See del as a possible HTML4 replacement.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Loose (deprecated in HTML4), HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

strong - Strong emphasis

Syntax: `<strong>...</strong>`

Description: `Bold` and `strong` text are normally **rendered the same** in visual browsers.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

style - Embedded style sheet

Syntax: `<style>...</style>`

Description: `style` (zero, one or many) embeds a StyleSheet in the document.

See also: `base`, `isindex`, `head`, `link`, `meta`, `object`, `script`, `style`, `title`

Properties:

Content model:	Other
Contained in:	`html`
Can contain:	(embedded stylesheet)
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`i18n`:	- see language-attributes (for use with title)

`media`	designed for use with these media	
`title`	advisory title	
`type`	content type of style language (RFC2045☞)	*Required*

media:

(Single or comma-separated list of media descriptors) With minimal original support by browsers, `media` is intended to specify the display device upon which the StyleSheet should be supported.

Note: Navigator4 ignores any other than 'screen' whilst Opera needs 'projection' for full-screen. The following were defined in HTML4 (but not in 4.01):

- *all* for all devices
- *aural* for speech synthesizers
- *braille* for braille tactile feedback devices
- *handheld* for handheld devices (characterized by a small, monochrome display and limited bandwidth) (this was well before the iPhone, etc.)
- *print* for output to a printer
- *projection* for projectors
- *screen* (default) for non-paged computer screens
- *tty* for fixed-pitch character grid displays (eg. Lynx)
- tv for television-type devices; low-res & limited scroll

title:

The intention was to allow a user to selectively enable/disable particular sheets; this is minimally supported amongst browsers. Multiple sheets with the same *title* are considered to be the same StyleSheet.

type:

This is *always* “text/css”.

See also link.rel=StyleSheet for including an external StyleSheet.

sub – Subscript

Syntax: `<sub>...</sub>`

Description: `sub` (and `sup`) can completely ruin the run of normal text-lines, so use with some care.

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

sup – Superscript

Syntax: `<sup>...</sup>`

Description: `sup` (and `sub`) can completely ruin the run of normal text-lines, so use with some care.

See also: `a`, `bdo`, `br`, `img`, `map`, `object`, `q`, `script`, `span`, `sub`, `sup`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

table – Table

Syntax: `<table>...</table>`

Description: `table` defines a *tabular* block of row-and-columnar data. The place where a row & column meet is called a "data cell" (`td` or `th`) and, because any data cell can also contain a `table`, it becomes possible to display some *very* complex, multi-dimensional data.

See also: `address`, `blockquote`, `div`, `dl`, `fieldset`, `form`, `hr`, `noscript`, `p`, `table`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	An optional `caption`, followed by zero or more `col` and `colgroup` elements, followed by an optional `thead` element, an optional `tfoot` element, and then one or more `tbody` elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right) table position relative to window (deprecated)
`bgcolor`	(sRGB colours) background colour for cells (deprecated)
`border`	(pixels) controls frame width around table
`cellpadding`	(pixels or %) spacing within cells
`cellspacing`	(pixels or %) spacing between cells
`frame`	(void|above|below|hsides|lhs|rhs|vsides|box|border) which parts of frame to render
`rules`	(none|groups|rows|cols|all) rulings between rows and cols
`summary`	purpose/structure for speech output
`width`	(pixels or %) table width

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

align:

bgcolor:

These are deprecated in HTML 4.01 Strict, and are mentioned here only because, before HTML4, `table` became notorious for being used as a layout device for entire documents. One of the main design purposes in the introduction of HTML4 was to separate the aspect of SGML *presentation* from the *markup*, and this latter was achieved by the introduction of `style` and stylesheets (CSS). As mentioned elsewhere, HTML4 Loose was designed as a halfway-house (detox) whilst older html authors made the transition to using HTML4. All elements & attributes marked as 'deprecated' in this PDF will validate under a *HTML4 Loose* `DTD` but will fail under *HTML4 Strict*.

So, what is wrong with `table` being used as a layout device? Well, apart from producing pages that look more than a bit *naff*, `table`s have an inherent problem for users that use narrow windows, or need large fonts, or use non-visual browsers. Even for visual browsers, most will not display very much of a table until that entire markup has been downloaded which—and especially on a bad connection—can lead to a noticeable delay. Most new visitors—if the view window does not fill up quickly—these days will just press the back-button & try elsewhere.

There is only one thing that this author has found that a table can do that CSS cannot, and that is *vertically* align content in a space ("`<td valign="middle">`") (although it has to be said that some other layout with CSS is also a bit of a black art).

border:

frame:

These 2 attributes .together govern whether the table's outer border is visible, and which sides, and by what width. As always, CSS will offer far better control. These are the possible values for `frame`:

- void — no border (default unless `border>0`)
- above — top only
- bottom — bottom only
- hsides — left & right only
- vsides — top & bottom only
- lhs — left only
- rhs — right only
- box/border — border on all sides (default if `border>0`)

`<table border>` is a valid, reliable shorthand for `<table frame="border">`

cellpadding:

cellspacing:

These 2 attributes .can help to make the difference between a crowded layout & a more professional display. 'padding' is the space between the cell border & the content within it, whilst 'spacing' is the distance between adjacent cells. *Take care!* some (older) browsers will not support '%' here.

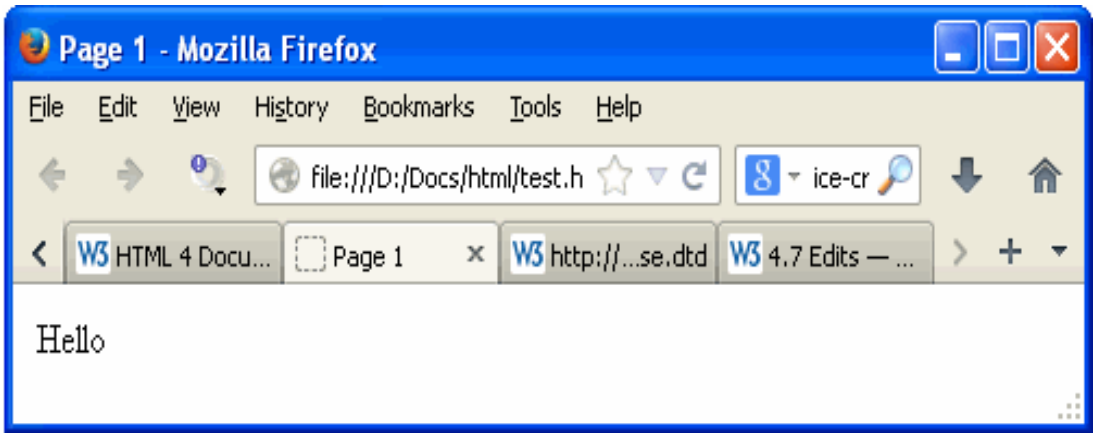
width:

Using pixels for width can be very dangerous in most cases, as you cannot be sure of the width of the user's screen, so best to use '%' if at all.

The whole thing:

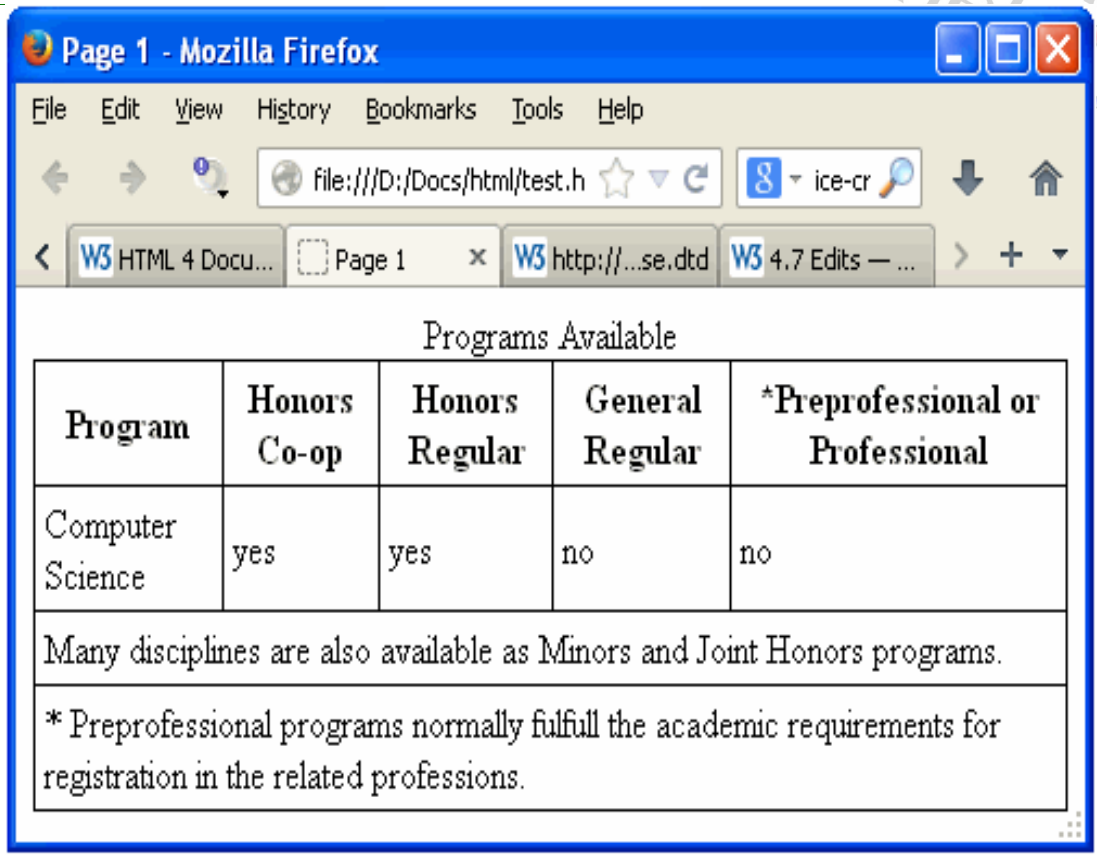
The simplest possible table is *really* simple (and probably not very useful):

```
<table>
  <tr>
    <td>Hello</td>
  </tr>
</table>
```



Here is a much more complex example using most elements, with default browser style (a "`border rules="all"`" attributes have been added so that you can discern all the individual cells) (see below the html code):

```
<table border rules="all" cellpadding=4>
  <caption>Programs Available</caption>
  <colgroup class="program-discipline">
  <colgroup class="program-type" span=5>
  <thead>
    <tr>
      <th scope="col">Program</th>
      <th scope="col">Honors Co-op</th>
      <th scope="col">Honors Regular</th>
      <th scope="col">General Regular</th>
      <th scope="col">*Preprofessional or Professional</th>
    </tr>
  </thead>
  <tfoot class="footnote">
    <tr>
      <td colspan=5>
          Many disciplines are also available as Minors and Joint Honors
programs.
      </td>
    </tr>
    <tr>
      <td colspan=5>
        * Preprofessional programs normally fulfull the academic
requirements for registration in the related professions.
      </td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td scope="row">Computer Science</td>
      <td>yes</td>
      <td>yes</td>
      <td>no</td>
      <td>no</td>
    </tr>
  </tbody>
</table>
```



Programs Available

Program	Honors Co-op	Honors Regular	General Regular	*Preprofessional or Professional
Computer Science	yes	yes	no	no
Many disciplines are also available as Minors and Joint Honors programs.				
* Preprofessional programs normally fulfull the academic requirements for registration in the related professions.				

tbody - Table body

Syntax: `<tbody>...<tbody>`

Description: The optional `thead` , `tfoot` & `tbody` are intended by the HTML4 designers as *structural* elements within tables, grouping common elements together. The vision was that browsers would provide a scrolling region for the body of long tables & fixed sections at head & foot. Instead, much as with `col` & `colgroup`, the early HTML4 browsers decided simply to ignore them.

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

Properties:

Content model:	Block
Contained in:	`table`
Can contain:	One or more `tr` elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify|char) horizontal alignment of cells (deprecated)
`char`	(default: '.') char value if `align=char`
`charoff`	(pixels or %) offset in cell to char if `align=char`
`valign`	(top|middle|bottom|baseline) vertical alignment of cells

The HTML order must be `thead`,`.tfoot` then `tbody` (all are optional) (the intention is to try to avoid rendering delays). See table: the whole thing for most of the elements of `table` assembled together.

Note: the `scope` attribute of `th` and/or `tr` elements offers a simpler alternative to `tbody`, whilst missing the CSS/script opportunities.

td - Table data cell

Syntax: `<td>...<td>`

Description: These are *header* (`th`) or *data* (`td`) cells (if the cell's content is both, then a `td` should be used) (however, the split aids in both table structure & presentation, even though the attributes for each are identical).

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

Properties:

Content model:	Block
Contained in:	`tr`
Can contain:	Inline elements, block-level elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`abbr`	abbreviation for header cell
`align`	(left|center|right|justify|char) horizontal alignment of cells (deprecated)
`axis`	comma-separated list of related headers
`bgcolor`	(sRGB colours) background colour for cell (deprecated)
`char`	(default: '.') char value if `align=char`
`charoff`	(pixels or %) offset in cell to char if `align=char`
`colspan`	(default: 1) number of cols spanned by cell
`headers`	list of id's for header cells
`height`	(pixels or %) height for cell (deprecated)
`nowrap`	suppress word wrap (deprecated)
`rowspan`	(default: 1) number of rows spanned by cell
`scope`	(row|col|rowgroup|colgroup) scope covered by header cells
`valign`	(top|middle|bottom|baseline) vertical alignment of cells
`width`	(pixels or %) width for cell (deprecated)

Note1: a cell may be empty, although visual browsers may then give it a *naff* presentation (as if to tick you off), so always include at least a space.

Note2: a cell may contain another `table`, (or any other block-level element) giving rise to (possibly) *very* complex tables.

abbr:

axis:

headers:

scope:

(all new at HTML4) `headers` + `scope` have a similar purpose, with `scope` being the simpler of the two; all 4 attributes are designed for cells that provide header information (with `axis` also having a broader application).

HTML4 has removed the earlier distinction between `th` & `td` to a *presentation* level (through stylesheets). Both are intended to be *data* cells, and their attributes are thus identical. For common structure/presentation in extremely large tables, HTML4 provides `col` & `colgroup`. For a simple method of structure/presentation in smaller tables it provides `scope` and for full control it gives `td.headers`.

`abbr`: a precis of the cell's contents (to be shown whilst the table loads, etc.).
`scope`: the data cells for which *this* cell provides header information; one of:

- col
- colgroup
- row
- rowgroup

`headers`: the header cell(s) which provide header information for *this* cell (inverse of `scope`). The value is a list of the header-cell `id`'s.
`axis`: the W3c says☞: *"This attribute may be used to place a cell into conceptual categories that can be considered to form axes in an n-dimensional* space" (wtf?).

To the best of my knowledge, browser designers have mostly ignored everything written in this section, and have failed to implement any (or most) of what HTML4 hoped for in that regard. They appear to have decided that, at this point, the HTML4 designers had disappeared up their own axis.

colspan:

rowspan:

(default 1) It is possible to have a value of '0' (zero), which is intended to mean "all cols|rows from here to the end" - beware of the value of `dir` in that. However, it was ignored by most HTML4 browsers.

Some practical advice: it is *very* easy, as you build bigger tables, to make mistakes with these 2 attributes. *Always* test out your layout before entering any data.

See table: the whole thing for most of the elements of `table` assembled together.

textarea - Multi-line text input

Syntax: `<textarea>...</textarea>`

Description: `textarea` defines a multi-line text-input control (see also `input type=text` for the single-line text-input control).

See also: `button`, `fieldset`, `form`, `input`, `label`, `legend`, `optgroup`, `option`, `select`, `textarea`

Properties:

Content model:	Inline
Contained in:	Block-level elements, inline elements except `button`
Can contain:	Plain text (which includes entities)
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`accesskey`	accessibility key character
`cols`	# of columns
`disabled`	
`name`	field name
`onblur`	(script) the element lost the focus
`onchange`	(script) the element value was changed
`onfocus`	(script) the element got the focus
`onselect`	(script) some text was selected
`readonly`	(new in HTML4)
`rows`	# of rows
`tabindex`	position in tabbing order

Any initial content—which must NOT contain any html tags—is treated by browsers as the default content on construction. On submission, the usual name/value pair is submitted; the `name` attribute is the *key* whilst current content is the value. Before HTML4 these *had* to be defined within a form.

cols:

rows:

One important feature to understand is that these two attributes govern only the *visual* appearance of the control, and not the amount of text that can be entered by the user, which *in theory* is unlimited, although some browsers will limit the amount to 32k or 64k. In addition, the mysql DB☞ will typically store an absolute max of 64kB in a TEXT field, which then can drop drastically if UTF8 to 21,844 characters☞.

Ultimately, the author of the server-side routines that process form submissions is responsible for all checks & restrictions on data received from form submissions and, with `textarea`, size needs to be one of those checks.

tfoot - Table foot

Syntax: `<tfoot>...<tfoot>`

Description: The optional `thead` , `tfoot` & `tbody` are intended by the HTML4 designers as *structural* elements within tables, grouping common elements together. The vision was that browsers would provide a scrolling region for the body of long tables & fixed sections at head & foot. Instead, much as with `col` & `colgroup`, the early HTML4 browsers decided simply to ignore them.

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

Properties:

Content model:	Block
Contained in:	`table`
Can contain:	One or more `tr` elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify|char) horizontal alignment of cells (deprecated)
`char`	(default: '.') char value if `align=char`
`charoff`	(pixels or %) offset in cell to char if `align=char`
`valign`	(top|middle|bottom|baseline) vertical alignment of cells

The HTML order must be `thead`,`.tfoot` then `tbody` (all are optional) (the intention is to try to avoid rendering delays). See table: the whole thing for most of the elements of `table` assembled together.

Note: the `scope` attribute of `th` and/or `tr` elements offers a simpler alternative to `tfoot`, whilst missing the CSS/script opportunities.

th - Table header cell

Syntax: `<th>...<th>`

Description: These are *header* (`th`) or *data* (`td`) cells (if the cell's content is both, then a `td` should be used) (however, the split aids in both table structure & presentation, even though the attributes for each are identical).

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

Properties:

Content model:	Block
Contained in:	`tr`
Can contain:	Inline elements, block-level elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`abbr`	abbreviation for header cell
`align`	(left|center|right|justify|char) horizontal alignment of cells (deprecated)
`axis`	comma-separated list of related headers
`bgcolor`	(sRGB colours) background colour for cell (deprecated)
`char`	(default: '.') char value if `align=char`
`charoff`	(pixels or %) offset in cell to char if `align=char`
`colspan`	number of cols spanned by cell
`headers`	list of id's for header cells
`height`	(pixels or %) height for cell (deprecated)
`nowrap`	suppress word wrap (deprecated)
`rowspan`	number of rows spanned by cell
`scope`	(row|col|rowgroup|colgroup) scope covered by header cells
`valign`	(top|middle|bottom|baseline) vertical alignment of cells
`width`	(pixels or %) width for cell (deprecated)

Note1: a cell may be empty, although visual browsers may then give it a *naff* presentation (as if to tick you off), so always include at least a space.

Note2: a cell may contain another `table`, (or any other block-level element) giving rise to (possibly) *very* complex tables.

abbr:

axis:

headers:

scope:

`headers` + `scope` (all new at HTML4) have a similar purpose, with `scope` being the simpler of the two; all 4 attributes are designed for cells that provide header information (with `axis` also having a broader application).

HTML4 has removed the earlier distinction between `th` & `td` to a *presentation* level (through stylesheets). Both are intended to be *data* cells, and their attributes are thus identical. For common structure/presentation in extremely large tables, HTML4 provides `col` & `colgroup`. For a simple method of structure/presentation in smaller tables it provides `scope` and for full control it gives `th.headers`.

`abbr`: a precis of the cell's contents (to be shown whilst the table loads, etc.).
`scope`: the data cells for which *this* cell provides header information; one of:

- col
- colgroup
- row
- rowgroup

`headers`: the header cell(s) which provide header information for *this* cell (inverse of `scope`). The value is a list of the header-cell `id`'s.
`axis`: the W3c says☞: *"This attribute may be used to place a cell into conceptual categories that can be considered to form axes in an n-dimensional* space" (wtf?).

To the best of my knowledge, browser designers have mostly ignored everything written in this section, and have failed to implement any (or most) of what HTML4 hoped for in that regard. They appear to have decided that, at this point, the HTML4 designers had disappeared up their own axis.

colspan:

rowspan:

(default 1) It is possible to have a value of '0' (zero), which is intended to mean "all cols|rows from here to the end" - beware of the value of `dir` in that. However, it was ignored by most HTML4 browsers.

Some practical advice: it is *very* easy, as you build bigger tables, to make mistakes with these 2 attributes. *Always* test out your layout before entering any data.

See table: the whole thing for most of the elements of `table` assembled together.

thead - Table head

Syntax: `<thead>...<thead>`

Description: The optional `thead` , `tfoot` & `tbody` are intended by the HTML4 designers as *structural* elements within tables, grouping common elements together. The vision was that browsers would provide a scrolling region for the body of long tables & fixed sections at head & foot. Instead, much as with `col` & `colgroup`, the early HTML4 browsers decided simply to ignore them.

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

Properties:

Content model:	Block
Contained in:	`table`
Can contain:	One or more `tr` elements
Standard:	HTML 4.01 Strict

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify|char) horizontal alignment of cells (deprecated)
`char`	(default: '.') char value if `align=char`
`charoff`	(pixels or %) offset in cell to char if `align=char`
`valign`	(top|middle|bottom|baseline) vertical alignment of cells

The HTML order must be `thead`,`.tfoot` then `tbody` (all are optional) (the intention is to try to avoid rendering delays). See table: the whole thing for most of the elements of `table` assembled together.

Note: the `scope` attribute of `th` and/or `tr` elements offers a simpler alternative to `thead`, whilst missing the CSS/script opportunities.

title - Document title

Syntax: `<title>...</title>`

Description: `title` is *Required*; there may only be one, composed of text and/or entities (no HTML tags allowed).

See also: `base`, `isindex`, `head`, `link`, `meta`, `object`, `script`, `style`, `title`

Properties:

Content model:	Other
Contained in:	`html`
Can contain:	Plain text, including entities
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`i18n:`	- see language-attributes

Writing a good title is something of an art-form, long practised by Newspaper & Magazine sub-editors; the main advice given here will be: "less is more" (*"multum in parvo"*). Consider the following:

- tabbed browsers put it in the tab (with the full title as a *tooltip*)
- OSs will use it for the taskbar, dock, menu bar, etc.
- SEs will pay strong attention to it in placing the page in the SERPs, in part according to how it matches the content that follows

The Title has become yet another place that has become keyword-stuffed as HTML writers fall prey to the lure of SEO & chase the SERPs. That can easily become a fools' errand. It is humans that read web-pages (bots only scrape them), so write for your human readers first. If they like what they read, they will recommend your words to other readers, and your pages will rise up the SERPs. It's easy, really; the content& how it is presented truly should be King and Queen in all that you do.

tr - Table row

Syntax: `<tr>...<tr>`

Description: `tr` is a table *row*, and groups together a common row of *header* (`th`) or *data* (`td`) cells. `tr` is thus a *structure* & *presentation* element rather than a data element but, unlike all other such table elements, is *Required*.

See also: `caption`, `col`, `colgroup`, `tbody`, `table`, `td`, `tfoot`, `th`, `thead`, `tr`

Properties:

Content model:	Block
Contained in:	`thead`, `tfoot`, `tbody`
Can contain:	One or more `td` or `th` elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`align`	(left|center|right|justify|char) horizontal alignment of cells (deprecated)
`bgcolor`	(sRGB colours) background colour for row (deprecated)
`char`	(default: '.') char value if `align=char`
`charoff`	(pixels or %) offset in cell to char if `align=char`
`valign`	(top|middle|bottom|baseline) vertical alignment of cells

`tr` must sit within a `thead`, `tfoot` or `tbody`. There can be zero instances of the first two and the start/end tags for `tbody` are optional, which means that the simpler structure of a html3.2 table is still fully valid HTML4.0.1.

See table: the whole thing for most of the elements of `table` assembled together.

tt - Teletype text

Syntax: `<tt>...</tt>`

Description: Not everyone reading this will know what `teletype-text` originally looked like. It normally gets rendered in a browser as `monospaced-text`.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

u - Underlined text

Syntax: `<u>...</u>`

Description: `u` is normally rendered in a visual browser in an underlined style.

See also: `abbr`, `acronym`, `b`, `bdo`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `q`, `samp`, `small`, `span`, `strong`, `sub`, `sup`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Loose (deprecated in HTML4), HTML 3.2 Final

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

ul - Unordered list

Syntax: `<ul>...</ul>`

Description: `ul` is an *unordered*.list of items and, through the black magic of stylesheets, is often used for menu items.

See also: `li`, `ol`, `ul`

Properties:

Content model:	Block
Contained in:	`applet`, `blockquote`, `body`, `button`, `center`, `dd`, `del`, `div`, `fieldset`, `form`, `iframe`, `ins`, `li`, `map`, `noframes`, `noscript`, `object`, `td`, `th`
Can contain:	One or more `li` elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

`compact`	display in a compact style (deprecated)
`type`	(disc|square|circle) style for bullet (deprecated)

compact:

`compact` is poorly supported by browsers.

type:

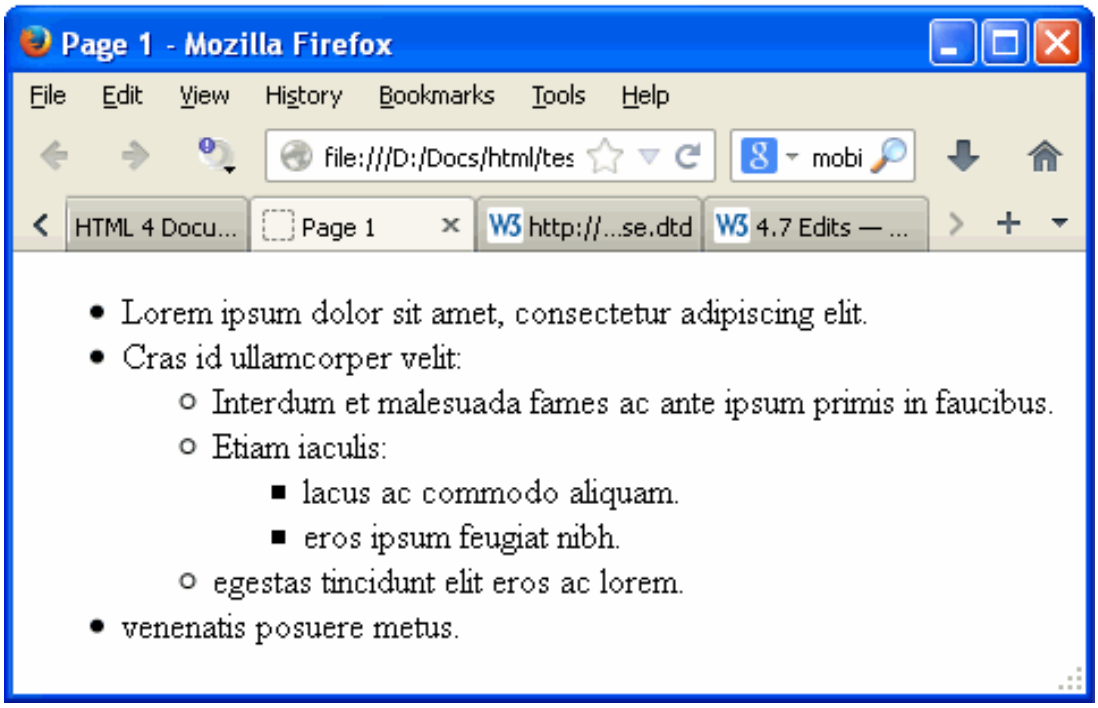
`type` governs the *style* of presentation of the bullet allotted to the contained `li` elements, and thus is deprecated in HTML4.01 Strict. The value is disc|square|circle and, in the absence of any stylesheet information, the browser will decide according to the *level* of the list (`li` elements can be nested, which will give rise to different levels of listing – see next page for an example in a browser).

The whole thing:

Here is html for `ul`, with nested `li`; note that the *white space* in the html should have zero effect on the results in the browser, although some browsers in the past have had bugs in some versions causing differences in presentation according to the presence/absence of white space. For that kind of reason I would always advise to use the closing end-tag, even though—as here—it is often optional:

```
<ul>
  <li>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</li>
  <li>Cras id ullamcorper velit:<ul>
    <li>Interdum et malesuada fames ac ante ipsum primis in faucibus.</li>
    <li>Etiam iaculis:<ul>
      <li>lacus ac commodo aliquam.</li>
      <li>eros ipsum feugiat nibh.</li></ul></li>
    <li>egestas tincidunt elit eros ac lorem.</li></ul></li>
  <li>venenatis posuere metus.</li>
</ul>
```

...and what it looks like with a default stylesheet in Firefox23.0.1:



var – Variable

Syntax: `<var>...</var>`

Description: `var` is normally rendered in a visual browser in *italics*.

See also: `abbr`, `acronym`, `b`, `big`, `cite`, `code`, `dfn`, `em`, `i`, `kbd`, `samp`, `small`, `strong`, `tt`, `var`

Properties:

Content model:	Inline
Contained in:	Block-level + inline elements
Can contain:	Inline elements
Standard:	HTML 4.01 Strict, HTML 3.2 Final, HTML 2.0

Attributes:

`core:`	- see core-attributes
`i18n:`	- see language-attributes
`events:`	- see event attributes

Reference Guide:- CSS

Introduction

HTML: "HyperText Markup Language"
CSS: "Cascading Style Sheets"

HTML is a markup language for documents that may contain more than just plain text, and CSS is a simple method for the author to attach visual (or perhaps spoken) *style* to that document's display.

Introduced together with `script` as *placeholders* in HTML3.2 Final☞, `style` first became an active element in HTML4☞.

Visually, HTML is designed to be displayed on a *canvas*. That 'canvas' may be a mobile, a computer VDU ('Visual Display Unit'), a piece of paper, a projector screen or the side of a house; CSS attempts to provide a formatting & layout model that will give a similar result in all circumstances, yet remain simple to understand & use. It mostly succeeds in that.

The building blocks of CSS–the various CSS Properties—are easy to understand & apply for those that are familiar with HTML. Complexity may arrive with the ways that those properties can interact with the HTML elements & each other on the canvas, and perhaps also with the terminology used. The latter—the terms used—will be familiar for anyone that has used DTP ('Desk-Top Publishing') software.

The next-but-one section in this brief intro is the CSS Layout Model. It will set out how HTML elements are laid out on the canvas, and will also intoduce all the DTP terms that will be used in these pages. It's simple stuff, but vital to know if you want to get to grips with CSS. But first...

"Cascading":

The next section is how to attach *style* & *StyleSheets* to the HTML document, and in particular how 'cascading' works.

It is perfectly normal to have 3 or more sets of *style* acting on a document at once; here is an example:

1. The author's site-wide *StyleSheet*
2. Document-specific inline *style (an* embedded *StyleSheet)*
3. HTML element inline *style*

The first helps to give the same look 'n' feel across an entire site.
The second can give a specific look to a specific page.
The third will (say) give a specific style to a specific sentence.

Further, the first *StyleSheet* may import other *StyleSheets* (and so on), and it is also possible for the reader to setup a personal stylesheet (perhaps they want to increase the font-size on all docs that they read); in the end, it can get *really* complicated as to which style-statement for a HTML element should apply. The methods & resolution of all this is known collectively as the *"Cascading Mechanism"*.

Attaching Style/StyleSheets to a HTML document

Overview:

There are lots of ways to tell the browser what style you want. Things are done like that to both provide flexibility & also to expand the author's options. It can mean, however, that the browser needs to decide which, from several competing style statements, it will apply to a specific HTML element.

This section will first itemise all the many ways to attach style to a doc, list common methods of specifying style (the stylesheet *'syntax'*) and then, at the end, will explain the *cascading order* by which the browser chooses one style statement over other, competing statements.

```
<html>
  <head>
    <title>title</title>
    <meta http-equiv="Content-Style-Type" content="text/css">
    <link rel="stylesheet" type="text/css"
      href="http://style.com/cool" title="cool">
    <link rel="alternate stylesheet" type="text/css"
      href="http://style.com/uncool" title="uncool">
    <style type="text/css"><!--
      @import url(http://style.com/basic);
      h1 { color: blue; }
    --></style>
  </head>
  <body>
    <h1>headline is blue</h1>
    <p style="color: green;">while the paragraph is green.
  </body>
</html>
```

The W3C example☞ above shows 4 methods of how to combine style & HTML. Strictly, this is a HTML topic, and is also discussed in the relevant sections of the HTML Reference.

1. Loading an external StyleSheet
2. Using an Embedded StyleSheet
3. Importing a StyleSheet (@import)
4. Inline Style – class, id or style
5. Stylesheet Syntax – Intro
6. Stylesheet Syntax
   a. Comments
   b. Whitespace
   c. Statements
   d. Grouping selectors
   e. Class & ID selectors
   f. Inheritance
   g. Contextual Selectors
7. Cascading Order
   a. !important

Loading an external StyleSheet

example:

```
<link rel="stylesheet" type="text/css"
   href="http://style.com/cool" title="cool">
<link rel="alternate stylesheet" type="text/css"
   href="http://style.com/uncool" title="uncool">
```

Definition:- a *"StyleSheet"* is a plain-text document full of style statements.

(examples of such 'style statements' are at the bottom of each CSS Property page in this PDF; web-page stylesheets can also be viewed as plain-text in a browser, or by using various browser extensions) (try Web Developer☞)

rel=stylesheet:

The HTML `link` element with 'rel=stylesheet' (must be within the `head`) will cause every style statement in the 'cool' stylesheet to be applied to the document. Multiple such elements can be used, with differing `href` values, and the `title` attribute is important:-

- *no* `title`*:-* is the *persistent* StyleSheet (always applied) (one only)
- *some* `title`*:-* is the *preferred* StyleSheet (auto-applied) (one only)
- *same* `title`*:-* all are concatenated together into one stylesheet

rel="alternate stylesheet":

The HTML `link` element with 'rel="alternate stylesheet"' (must be within the `head`) will cause every style statement in the 'uncool' stylesheet to be available as an *alternative* stylesheet for the document. The `title` attribute is *required* with this value. Multiple such elements can be used, with differing `href` values, to offer multiple options.

(many recent browsers, when there are optional stylesheets, will offer a method—try the menu—to switch between the styles)

This author strongly recommends that you make use of this method. Develop a basic 'look 'n' feel for your site in a single StyleSheet—most small & medium-sized sites will need just one—then get on with adding content to your site.

See also: (HTML) `link rel=stylesheet`

example:

```
<style type="text/css"><!--
   h1 { color: blue; }
--></style>
```

Definition:- an *"Embedded StyleSheet"* are style statements placed between `style` elements.

(must be within the `head`) These are *inline style* statements that apply to the whole document. Essentially, this is a stylesheet embedded within the document; that 'stylesheet' can be as few as one style statement, or many. The same rules as for external stylesheets apply.

HTML comments:

The example at top is shown embedded within SGML *comments* ("`<!-- ... -->`"). These were essential to use shortly after introduction, as pre-HTML4 browsers would ignore the unknown 'style' tag, yet display the style statements as text in the document. Embedding those statements within comment-tags prevented those browsers from doing that, but still allowed conforming browsers to recognise the `style` statements. Few folks use such ancient browsers these days, but the practice does no harm.

External stylesheets do not need to use SGML *comments*.

Importing a StyleSheet

example:

```
<style type="text/css"><!--
   @import url(http://style.com/basic);
--></style>
```

Definition:- an *"Imported StyleSheet"* is an external stylesheet added using the '`@import`' statement.

It must be:

1. Within a stylesheet (whether external or embedded).
2. The first statement in the stylesheet.

More than one `@import` statement can be used.

Style statements within the stylesheet containing the `@import` statement over-ride any conflicting imported statements for the same element (the last one wins). See the Cascading Mechanism for the full set of Cascade rules.

HTML comments:

See the page on embedded stylesheets | html comments for the reason for their inclusion here.

What's the point?

'Modularity' is the main reason for this mechanism. The idea is to try to reduce the amount of typing, and to allow style-rules to be built in *modules* that can be re-used.

examples:

```
<meta http-equiv="Content-Style-Type" content="text/css">
...
<p class="engineer">
<p id="grass">
<p style="color: green;">
```

Definition:- "Inline Style" is style applied to a HTML element via the 'class', 'id' or 'style' core attributes.

HTML4 introduced the 'class', 'id' & 'style' attributes as core attributes for *almost* every HTML element (only basefont, param & script do NOT accept them). The style associated with that attribute is then applied directly to it, and also over-rides any other, competing style statements from other sources.

These 3 attributes effectively allow a stylesheet to be embedded *inline* within the element itself (remember:- a 'stylesheet' can be one, single.style statement, or many). If there are multiple style statements, then each one needs to be separated from the others by a semi-colon (';') (see 'style' example, top). Strictly, the last statement in a stylesheet does NOT need a terminating ';', but this author—from long, bitter experience—advises you to *always* include a terminating ';' for *every* statement, whether it needs it or not (it does no harm).

What are the differences?

- style:
  The attribute value (in the example above, 'color: green;') is the style to be applied to that element. If you find yourself wanting to use 3 or more statements, then you should probably use 'class', or possibly 'id'.

  Style statements within the style attribute will *always* apply, regardless of other competing style (on the basis of 'last one wins') (see also id).
- id:
  This attribute shares a *namespace* with the 'name' attribute, and is thus guaranteed to have a unique value within the document (in the example above, there will be one, and one only, 'grass' id or name (and not both) throughout the document).

  The id attribute is given the highest style *specificity*. It shares this specificity with the style attribute.
- class:
  This attribute is explicitly for style; learning to use it is more of an art than a science.

  The class attribute is given the next-highest style *specificity* (below id). Like id, it can be applied to *any* html element (repetitively, unlike id).

What are the similarities?

- `id, class`:
  Each of these attributes needs to be both declared & defined (given both a name as an attribute + a value) within a *stylesheet* before use. That stylesheet may be external, embedded or imported. Once declared + defined, the attribute can be used as normal within a HTML *tag*, and the style within the attribute value is then applied to that HTML element.

Some artificial examples:

There is every chance that you, the reader, may not yet understand all the differences nor similarities between these 3 attributes. If so, that will most likely be because you have not yet tried out a stylesheet on a document & played with it. This is one of those occasions where most of the learning will come from doing rather than reading (in addition, the stylesheet rules are in a later section). Nevertheless, to try to help:

1. Use a stylesheet to set the style for all the HTML elements that you will use in your document(s).
2. Use *class*(es) within a stylesheet for situation(s) where the style of particular sections will differ from the default. One art to try to pickup is to give the class a name that reflects the *style* that you want rather than a literal name (eg 'engineer' rather than 'dark-green', or 'agreement_conditions' rather than 'really_small') (think: you may at some later time decide to change the font-size).
3. *id* is useful for sections-within-a-section where it differs again on rare occasions. 'agreement_conditions' is perhaps a better example as a name for *id* than it is for *class*.
4. *style* is then useful for the one-off where you need to tweak the style.

default:
class:
id:

```
p           { background: #004400; color: #ccffcc; }
p.engineer  { background: #ccffcc; color: #white; }
p#grass     { background: green;   color: black; }
```

the html:

```
<p>
Some words to show you the default, shifting to <span
class="engineer">the Engineer Class to illustrate that</span>, and
finishing with the defaults.
</p>
<p class="engineer">
But, of course, then you want to give the Class it's full reign and
let it strut it's stuff in a full paragraph.
</p>
<p class="engineer" id="grass">
Now you mix it up a bit,       and discover that the id will always
win in an arm-wrestling match with the     Class.
</p>
<p style="background: white; color: black;" id="grass">
But what happens now? (remember that this style was defined <span
style="font-style: italic;">after</span> id.)
</p>
```

The result:

Some words to show you the default, shifting to the Engineer Class to illustrate that, and finishing with the defaults.

But, of course, then you want to give the Class it's full reign and let it strut it's stuff in a full paragraph.

Now you mix it up a bit, and discover that the id will always win in an arm-wrestling match with the Class.

But what happens now? (remember that this style was defined *after* id.)

example:

```
p color: green;
```

Definition 1:- "Syntax" is the set of words & punctuation that a browser can recognise as valid CSS – the rules of the game.

Definition 2:- "Computer" is a very fast idiot.

Do NOT: quote neither properties nor keywords.
(in the example above, '`color`' is a *property* & '`green`' is a *keyword*, representing the value for '`color`', and in this sentence I've quoted both of them, which is exactly what you should NOT do in your stylesheets!)

Hint: write & save your stylesheets—just like your HTML documents—in UTF-8, (there is more on this below) with no BOM ("Byte-Order Mark"). If you decide to use a different charset (DON'T!), make sure that the same charset is used for both documents & stylesheets.

Stylesheets will accept a *very* limited set of characters. UTF-8 contains approx 32,000 characters; stylesheets will allow you to use less than 127 of those (and only from the US-ASCII charset) (technically, *CDATA* with extra limits):

- A ⇒ Z
- a ⇒ z
- 0 ⇒ 9
- hyphens ('-')
- underscores ('_')

- periods ('.')
- colons (':')
- semi-colons (';')
- curly brackets ('{' + '}')
- hash marks ('#')
- exclamation marks ('!')
- slash-dot, dot-slash C-Style comments ('/* ... */')
- single-quotes ('')
- double-quotes ("")

- whitespace is ignored
  (tabs: 0x09; newlines: 0x0a; form-feeds: 0x0c; carriage-return: 0x0d; space: 0x20)

errata: The above is untrue. I'm deliberately lying to you; but for a good reason!

The above is true for HTML4.01☞ (plus `id` & `class` must begin with a letter, no spaces allowed). HTML5 has completely changed the rules of the game☞, which is a little dangerous: now, `id` & `class` must *contain* at least one letter, and spaces are still not allowed. Apart from that, you can use whatever characters that you like (including all punctuation if you want!).

My advice?: start with HTML4 until you are comfortable with it. HTML5 is not finalised & keeps changing as I type this. If you take the advice given so far, then you can begin to add HTML5 elements as you like with zero changes (all of *HTML 4.01 Strict* is also part of *HTML5*). In any case, the tighter rules for HTML4 will help you as you learn to give meaningful names for `class` and `id`.

Learn to use the HTML elements with the purpose for which they were designed. Some elements are b*lock*, whilst others are i*nline*, and there is a reason for that. CSS allows you to make any element either block *or* inline at will. That is ever so useful if you have a specific instance where you need to change it's character, but you should also ask whether perhaps you are using the wrong element.

Finally:- have fun! If it is hard work, but you are enjoying yourself, that is about right.

charset: This pdf suggests that you use UTF-8 as the encoding for your HTML documents and also for the CSS files. The Primer lessons also explain the steps that you need to take to make sure that your browser can recognise those documents correctly. That will allow you to use *any* language within your document & CSS, if your text-editor is also configured correctly to use UTF-8. Some of all that can be fiddly to get right, but none of it is difficult.

The next section will itemise all of the StyleSheet syntax, then give the *cascading order*.

A *stylesheet* is a plain-text file full of *statements* that link *style* to HTML *elements*. This section will set out the *syntax* & *grammar* of stylesheets, with examples. Here are two useful links to help you validate your HTML docs & your CSS, and another for the formal specification:

- W3C CSS Validator☞
- W3C HTML Validator☞
- CSS1 Grammar☞

Comments:

```
/* ...
 * ...
 */
```

In addition to style *statements* you can also have style *comments* within your stylesheets. These are C-style comments, and can be single- or multi-line. They begin with a *slash-dot* (‘`/*`’) and finish at a *dot-slash* (‘`*/`’). They can NOT be nested. When the browser *parses* (browser-speak for ‘reads’) a stylesheet & finds a matching *slash-dot* & *dot-slash*, it will remove them plus everything between them & replace it all with a single space.

Comments are useful for others that want to understand your stylesheet. They are also mighty useful for *you* if you later decide to edit the stylesheet.

Hint: learn to write useful comments.

Whitespace:

Whitespace is ignored, so you can place as many spaces, tabs and/or blank-lines in the stylesheet as you like.

Statements:

```
<selector> { <declaration> }
p { color: blue; }
```

This is a simple example of a style statement (and, all by itself, is sufficient to be a stylesheet); ‘`p`’ is the example of a “selector”, whilst ‘`color: blue`’ is the example of a “declaration”, which in this example would be the style that would be attached to all paragraph elements within that web-page. So:-

- `selector` == HTML element

The declaration above is itself a combination of style *property* (‘`color`’) and *value* (‘`blue`’) and, in this case, only sets a single property. Many properties may be set at the same time (“grouped”); each property/value pair should be separated by a semi-colon (‘;’). Here is an example:

```
p {
   color:      blue;
   background: white;
}
```

Grouping selectors:

```
h1, h2, h3 { color: blue; }
```

Grouping the selectors saves typing. It is also possible to group both selectors & declarations at the same time:

```
h1, h2, h3 {
   color:      blue;
   background: white;
}
```

Class & ID selectors:

```
code.html { color: blue; }
code#css  { color: red;  }
```

A class is set with a dot (‘.’), whilst an id is set with a hash (‘#’). Above they are associated closely with the `code` element (the declarations could also be grouped), but they can also be declared independently of the element (and then also with grouped declarations if you wish):

```
.html { color: blue; }
#css  { color: red;  }
```

This ‘bare’ declaration is perhaps more useful for Classes than for IDs: a class can be used in the html doc as often as you wish; the id can be used just once.

Hint: learn to give class/id names that are *meaningful,* and also that describe their *function* rather than their *appearance*. That will help you both remember what the name is as you use them, and also what it is supposed to do. Imagine if you call it a “*greeney*” class, then later decide that it looks better as ‘brown’, when what you wanted was a “*grounded*” class of style.

Inheritance:

We are now getting close to setting out the *‘cascading order’* of stylesheets but first, consider inheritance:

Each one of the css Properties sets out whether it is inherited, or not. The browser constructs an object tree from your HTML (called a ‘DOM’: “Document Object Model”) which starts with the `html` object itself as the top-level member; next is ‘`body`’, then so on, depending on how each is declared in your document. The language used is of ‘parent’ & ‘child’, so the ‘body’ object is a child of the top-level HTML object (and so on throughout the branches of the tree).

Most of the CSS Properties are inherited from parent to child (but some are not). An obvious example is font, which will be inherited—if set on `body`–as the default for all text throughout the object tree, and thus for all text in your webpage. Imagine then if you have a paragraph element & change the font in it. That paragraph will initially inherit a value for font from it’s parent, but your style declaration will change that. However, any children declared in the paragraph—perhaps an `em` (‘emphasised’ text) element—will inherit the *new*

value for font & not the old.

Contextual Selectors:

```
p em { background: yellow; }
```

The final syntax to consider with CSS1 are a type of selector that are *higher* in the *cascading order* than simple selectors. In the example above, setting `em` ('emphasis') on a word in a `p` ('paragraph') would change the background to yellow (whilst the default was white), yet would have zero impact on emphasised words within any other element. You can make the list of nested selectors as large as you like (*remember*: the order is important).

Cascading order:

It is the browser's job to decide which, out of (possibly) a multitude of different style statements, from (possibly) multiple stylesheets, will be chosen as the one(s) to apply to a specific HTML element at any point within that document. The browser is therefore given a set of rules to go through to decide which ones to choose. First, however, there is one, last, little wrinkle in the syntax which CSS offers to the writer to shift their favourite statement one notch higher up the preference order:

!important:

```
p { color: blue ! important; }
```

In general, the rule is *"last one wins"*. As an example, it was mentioned in earlier sections that one or many '@import' statements can be placed at the very top of the stylesheet, to import other stylesheets into the current stylesheet. Imagine that both the current and the imported stylesheet *both* contain `color` statements for `p`. Which one will apply? The answer is the *current* stylesheet, and the reason is that the current stylesheet statements are made *later* than the imported stylesheet, and—if everything else is equal—the *"last one wins"*.

Now, consider the reader's stylesheets (because the person using the browser can have their own, personal stylesheets). The general import order is:

1. Browser defaults
2. Reader's stylesheets
3. Author stylesheets

...so if the reader wants all paragraph text in a blue font, but the author wants red text, the author's choice will prevail as it was the last. However, if the reader uses the syntax of *"!important"* as at top, then the reader's choice will prevail... unless the author is naughty, and also uses *"!important"* for their red text, because then the weights will be identical and, in that situation, as always, *"last one wins"*.

So, the *cascading order* is next. In this, remember: the criteria is "weight". Which style statement carries the greatest weight? Which is the 500lb gorilla in the room? This is how it is decided:

Style properties are applied to HTML elements and, by design, there may be a large number of—and possibly conflicting—such properties for each element. The browser runs through a set of rules to discover which style statement has the greatest weight for a particular element at the place that it occurs within that html document. This is the algorithm used:

*Note: t*he 'sorting' below continues only whilst the browser has more than one rule with equal weight. As soon as it finds itself with just one rule at the top of the order, that one is chosen, and the sorting stops.

1. Find a stylesheet selector that matches the html element
   (go to next if one or more matches)
   a) If none apply, use the inherited value, or
   b) If no inherited value (eg html itself, or non-inherited properties) use the initial value (and we are done)
2. Sort by weight ('!important' raises the weight)
   a) author rules are chosen if equal weight
3. Sort by origin (most to least):
   a) Author's stylesheets
   b) Reader's stylesheets
   c) Browser defaults
4. Sort by specificity of selector (most to least):
   a) ID attributes in the selector
   b) Class attributes in the selector
   c) Contextual Selectors (how many tags)

   This one is complicated, so here is an example:

```
li            {...}  /* a=0 b=0 c=1 -> specificity =   1 */
ul li         {...}  /* a=0 b=0 c=2 -> specificity =   2 */
ul ol li      {...}  /* a=0 b=0 c=3 -> specificity =   3 */
li.red        {...}  /* a=0 b=1 c=1 -> specificity =  11 */
ul ol li.red  {...}  /* a=0 b=1 c=3 -> specificity =  13 */
#x34y         {...}  /* a=1 b=0 c=0 -> specificity = 100 */
```

   Pseudo-elements count as normal elements
   Pseudo-classes count as normal classes

5. Sort by order specified.
   a) the last one specified is the one chosen.

Phew!

Next is the final section in this introduction: the css Layout Model.

This is one of those vital css subjects. Here are the sections:

A Brief Note Before:

The organisational structure & governance of the Internet, including those bodies that control the protocols that allow the whole thing to work & those that police what everybody does with it is completely fascinating, because... well... because there arn't any & nobody does.

The W3C☞ *("World Wide Web Consortium")* is a good case in point. It was established by Sir Tim Berners-Lee in October 1994; he was the Brit that, in 1989, put together the (already existing) ideas of *Hypertext*, *TCP/IP*, *DNS* and (what was later called) the Internet. Crucially, he designed the HTTP protocol, wrote the first web-server and browser software, then gave this trillion-dollar package away to the world for nothing (*"this is for everyone"*). He was instrumental in creating the world's first website (at CERN, based in both Switzerland & France). This is what the W3C say about themselves:

"
The World Wide Web Consortium (W3C) is an international community that develops open standards to ensure the long-term growth of the Web.

The W3C acts as the central clearing-house for all efforts to develop HTML & CSS. Yet, they cannot *force* anyone to accept their proposals; in particular, the browser providers have to translate such proposals into actual working products, and their history is of, initially at least, eagerly accepting parts of some of the proposals & blithely ignoring others.

There is another, somewhat controversial aspect. In my view the W3C has made some very dumb decisions; I will pick out two in particular:

1. The CSS Box Model places the HTML element at the heart of the model, and it is not possible to directly specify the width nor height of the enclosing box. I have zero problems with the first part, but the second struck me as nuts when I first met it. Microsoft tried, using the dominance of MSIE at that time, to enforce a non-conforming fix with Internet Explorer 5 on the PC☞, but no-one else would accept that, and now we are all stuck with the original W3C proposal.
2. CSS Line-Box formatting is most ignorant, and makes it almost impossible to mix differing fonts on the page and obtain a professional result (in brief, CSS has abandoned the font *baseline* as the common feature for all font-display).

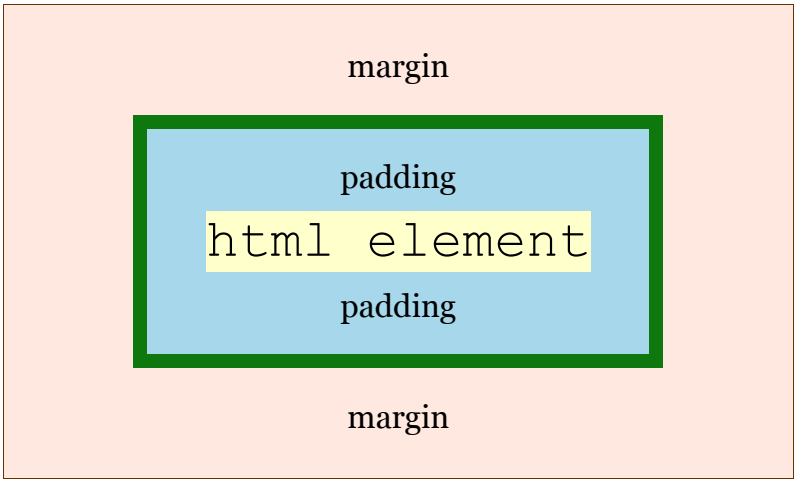
All the above, together with the browser wars & other features, has probably made the web-designer's work much more difficult than it needed to be. But then, nobody has ever tried to do this before, and on a world-wide basis. It is brand new, and the road for brand-new efforts is often rocky.

Well, it all makes for an interesting life and, as the ancient Chinese insult goes, *"may you live in interesting times"*.

Overview: An HTML element is *much* bigger than you think.

Warning!: This starts nice & simple, and steadily gets more complicated.

The job of the browser is to layout your precious html document on to a canvas, so that others can read (or *listen*) to it, and that whole process is called *"formatting"* (also: *"rendering"*) the document. The formatting *model* that the W3C has chosen☞ is a box, with the content at it's heart; specifically, the HTML element is the content, and that content sits within a box:



So:

1. html element:
   Strictly, '`height`' & '`width`' are almost *never* applied to an element (shown as yellow above) (see 'box' below); it fills the space left by the...
2. padding:
   This is padding *inside* the box, and surrounds the element. Padding has no colour. Put another way, it's colour is that of the HTML element(s) that lay underneath it, which will be their `background color` (shown as blue above).
3. box:
   (shown as green above) It has a 'border' (thickness), 'width', 'height' & many other properties.

   !! watch out !!: the 'width' & 'height' properties for the box are actually the width & height of the *content* of the box, and not of the box itself. If you want to know the height of the box, then you will need to add together the height of the contained element(s) + both sets of vertical padding + both sets of vertical border (and similar for the width).
4. margin:
   The is the margin *outside* the box, and gives a visual separation from adjacent boxes. Margin has no colour. Put another way, it's colour will be that of it's *parent* HTML element, which will be it's parent's `background color` (shown as pink above).

next: In HTML there are 3 *types* of element:

1. Block-Level
   Remember: a block-level HTML element has an *intrinsic* `display` value of 'block' or 'list-item' (as applicable)

   In CSS, the following are block-level:
   - 'display: block'
   - 'display: list-item'
   - any value for `float` other than 'none'

   Key feature: a block-level element opens on a new line
2. Inline
   Remember: an inline HTML element has an *intrinsic* `display` value of 'inline' (mostly, this is text)

   In CSS, the following are inline:
   - 'display: inline'
   - 'float: none'

   Key feature: an inline element opens at the next display position within the display flow
3. Other
   Remember: 'other' HTML elements have an *intrinsic* `display` value of 'none' (most of these must appear within the `head`)

   In CSS, the following are 'other':
   - 'display: none'

   Key feature: an 'other' element is not displayed on the canvas within the ordinary flow of HTML elements; at all; ever

Block-Level Elements:

There is a problem with a display based on boxes. You see, it can make the final page look a bit... well, *boxy*. I'll try to demonstrate that using text (which is *inline* rather than *block-level*, but no matter). Look at these 2 lines of text:

This is some text using a fixed-width font (Courier)
This is some text using a proportional ('typeset') font (Georgia)

Most folks will decide that the 'typeset' font is both more attractive and also, curiously, easier to read, even though there are far more letters on the line.

In both fonts, every letter occupies a little box. With the mono-spaced Courier font, every box is the same size, and no letter will ever touch nor overlap any other. With the proportional font Georgia, the vertical *height* of each box is identical to those in Courier, but the horizontal *width* is varied according to the content (hence the name 'proportional'). But there is more!

The human eye & brain together are fantastically good at picking out patterns, plus we have an innate quality-control mechanism that says *"I like that pattern better than those other patterns"*. The interesting feature in this is that we pay more attention to those patterns that we like than those that we do not. So, early computers started with fixed-width fonts like Courier, but as soon as computers like the Macintosh brought out typeset displays...
...everybody went *"I like those better"*!

The bottom line is: you do not want your web pages to look too boxy. Browsers try to cope with this by *"collapsing vertical margins"*; unfortunately, the treatment of horizontal margins is even more complicated.

Collapsing Vertical Margins:

```
<ul>
  <li>1st element of list
  <li>2nd element of list
</ul>
```

This is *far* easier to show than to explain...

The above is an attempt to demonstrate how a browser would lay out the boxes for the html shown above them (both the `ul` & the `li` elements have been given padding + margin in the css; the `ul` has a pink background whilst the `li` has a blue background; the `ul` padding is not marked, but surrounds the `li` margins outside the pink box that you can see).

You can see the *"collapsed vertical margins"* in the middle. If not collapsed, the gap between the two list elements would be twice what it currently is; that would look wrong. So, the browser looks at list1 bottom-margin, and at list2 top-margin, and chooses whichever is biggest for the gap; that looks much better.

PS: ul == 'unordered list' & li == 'list item' (lines of words with bullet points)

Here are the precise words☞ that define 'collapsing vertical margins':

"The width of the margin on non-floating block-level elements specifies the minimum distance to the edges of surrounding boxes. Two or more adjoining vertical margins (i.e., with no border, padding or content between them) are collapsed to use the maximum of the margin values"

CSS life becomes even more interesting when one of the elements has a *negative* margin (it gets deducted from the positive one). If both are negative, then *"the absolute maximum of the negative adjoining margins is deducted from zero"*.

Horizontal formatting:

Horizontal margins are *not* collapsed.

remember 1: the width of a box is actually the width of the *contents*
remember 2: the default value for width is 'auto'
remember 3: the sum of the 7 below == the width of the *parent* element

You need 7 CSS properties to discover the full, formatted width of a non-floating, block-level HTML element:

1. margin (left & right)
   (can be set to 'auto')
2. border (left & right)
3. padding (left & right)
4. width
   (can be set to 'auto')
   (the browser is also allowed to change it to 'auto' if it wants)

remember 4: a *'replaced element'* is an HTML element such as `img`, where the size of the element is not known until the browser acquires it & only then discovers what it's intrinsic size is.

How the Browser Handles '`width: auto`':

- *replaced element:*
  The width is replaced by the element's intrinsic width
- *all other elements:*
  Width calculated using the 7-Property sum as above

remember 5: 'width' has a non-negative, browser-defined, minimum value. This value can vary from element-to-element, and can even depend upon the value of other properties.

If 'width' goes below the minimum limit, because either:

- the author set too low a figure, or
- the author set it to 'auto' + the rules (below) would make it go below the minimum, then...

...the value for 'width' is replaced by the browser with it's minimum value.

The Rules, or, "Who Messed with my css?"!

- If none of the 7 properties are 'auto', then '`margin-right: auto`' is assigned
- If just one of 'margin' (left or right) or 'width' is 'auto', then that property is changed to a value which allows the browser to satisy the 7-Property calculation.
- If 'width' is 'auto' and either/both of 'margin' (left or right) are also 'auto', then both margins are set to zero, and width is changed to satisy the 7-Property calculation.
- Otherwise, if both margins are 'auto', they are set to equal values (which centres the element inside it's parent).

(Inline/Floating Elements)
Any of the 7 properties set to 'auto' with such an element is treated as if the value were actually zero.

Phew! Well, I did warn you.

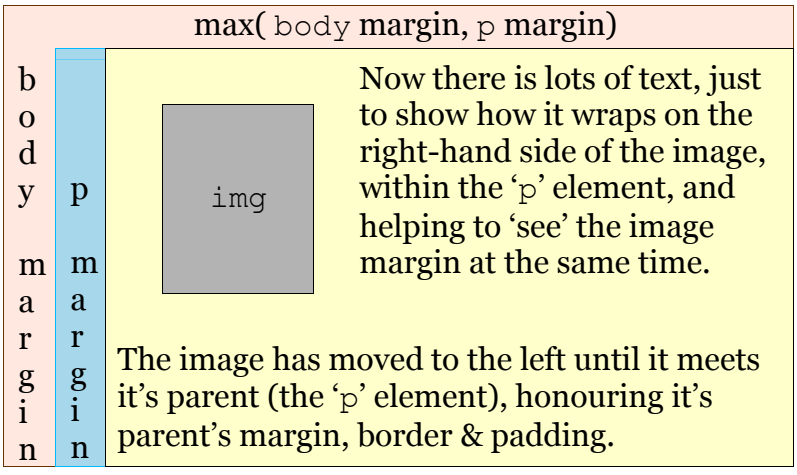
List-Item Elements

These are also formatted as block-level elements, with the added extra that they are preceded by a list-item marker. Perhaps the main item to note with these elements is that the directionality of the text decides as to whether the marker is placed on the left (*ltr* languages) or the right (*rtl* languages) (see `bdo` for extra on this).

Floating (Block-Level) Elements

Using the '`float`' property with any value other than 'none', any HTML element can be declared to be outside the normal flow of elements, and is then formatted as a block-level element. This is commonly done with images ('`img`'), so let's consider one of those to see what happens in the formatting:

```
img { float: left; }
body, p, img { margin: 1em; }
```



Did you notice the *"collapsed vertical margins"* at the top (but *not* with the `img`) (and also how margins are *not* collapsed horizontally)?

There are 2 situations in which floating, block-level elements can overlap with the margin, border and padding areas of other elements:

1. when the floating element has a negative margin
   Negative margins on floating elements are honoured, just as they are on other block-level elements
2. when the floating element is wider or higher than it's parent element

As always, far easier to show than to explain. Nevertheless, we have 2 sides to our brains, so here are the rules (and, as always, remember that when you see the word 'inner', we have a *content*-centric view, so 'inner' means inside the element margin, border *and* padding, whereas 'outer' means outside all those three):

A floating, block-level element is positioned subject to the following rules:

1. The left outer edge of a left-floating element may not be to the left of the left inner edge of its parent element (analogously for right-floating elements)
2. The left outer edge of a left floating element must be to the right of the right outer edge of every earlier (in the HTML source) left-floating element or, the top of the former must be lower than the bottom of the latter (analogously for right-floating elements)
3. The right outer edge of a left-floating element may not be to the right of the left outer edge of any right-floating element that is to the right of it (analogously for right-floating elements)
4. A floating element's top may not be higher than the inner top of its parent
5. A floating element's top may not be higher than the top of any earlier floating or block-level element
6. A floating element's top may not be higher than the top of any line-box with content that precedes the floating element in the HTML source
7. A floating element must be placed as high as possible
8. A left-floating element must be put as far to the left as possible (right-floating element as far to the right as possible). A higher position is preferred over one that is further to the left (right)

And once again: phew!

Inline Elements:

The W3C spends an awful lot of space in talking about block-level element css formatting, and relatively very little☞ on inline-elements. This PDF will try to explain the latter a little more thoroughly.

W3C definition: Elements that are not formatted as block-level elements are inline elements.

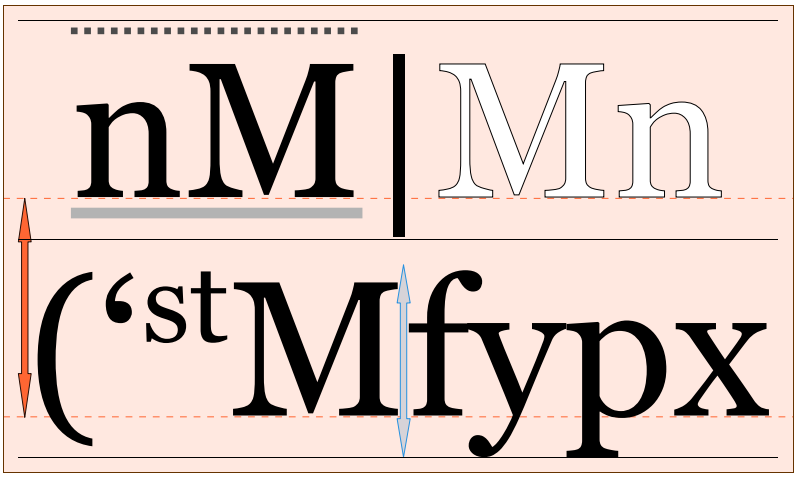
Most of the HTML elements in all the web-documents on the Internet are inline elements. That becomes obvious, just as soon as you realise that the commonest inline element is *text* (words). In fact, each & every *letter* in a word is essentially an inline element (unless it has had it's `display` property changed to 'block'). It is simply that, within the HTML elements, whilst you will find many elements that affect the *appearance* of text, and also many elements that can *contain* text, there is not a single element that *is* text. Nevertheless, as inline element formatting is entirely based on the established principles of text-formatting (derived from DTP ('Desk-Top Publishing'), which itself was derived from magazine & newspaper typography, which themselves drew upon principles established by the earliest printers, going back thousands of years and cast (literally) in die by the earliest mechanical printing presses of Gutenberg & others in the 14th Century), it seems to me that anyone who wants to attempt to use CSS within HTML documents should have a chance to understand the basic principles, and the terms used, and what they mean. So, let's start with a brief intro:

Above is some centred 72-point Georgia text, set single-space with default-leading, and with 0.5pt hairlines placed above & below each paragraph-line so that you can see the line-height. The first 2 letters ('nM') have a solid underline plus a dotted overline, whilst the last 2 letters on that line ('Mn') are shown as outline characters (which lets you see their mathematical shapes, because this font is drawn in curves rather than bit-mapped). The character in the middle is an 'upright bar' (0x7c), which goes from just above the bottom descender to just below the top ascender.

The second line shows some lower-case letters plus punctuation. Notice how the letters with *descenders* ('y' & 'p') go all the way to the bottom, and that the 'f' actually overhangs the left-most part of the 'y' (called 'pair kerning') and rises *above* the capital 'M'. Finally, the 'st' are in *superscript* text.

The Baseline

Now that you have been introduced to each other, let's start to define some important names & ideas. The first is the notion of a *"Baseline"*, and we will immediately find ourselves mired in controversy as someone, somewhere, has made a dumb decision which, once again, we all have to live with.



Here is the identical box & text as before, but with a *Baseline* (orange dashes) added for both lines of text (plus an orange vertical arrow, showing the *typographic* measure of line-height—which is upwards, from baseline-to-baseline—and also a blue vertical arrow—72 points high, stretching from just below the bottom of the lowest descender to just above the tallest ascender—to help you visualise what a font-height actually means).

So, the *Baseline* is the imaginary, horizontal line that most characters (certainly in Latin text) sit upon. So, where is the controversy?

Professional DTP, just like the publishing industry, bases it's text layout upon the *Baseline*. CSS bases it's text layout upon the *"line-box"*, which is essentially the text line-height with the text itself embedded centrally within it (more later). That latter is most foolish; the baseline is *the* consistent feature within all fonts, and basing layout upon the baseline allows different fonts & families & sizes to be placed within the same line of text, yet look OK (which is why printers did that). On the other hand, the CSS designer's choice allows you to easily make a mixed-font page look like a dog's dinner.

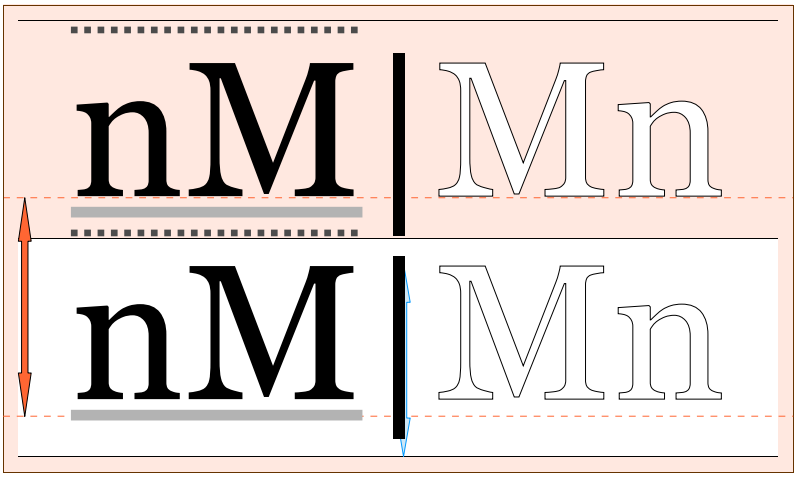
It is easy to understand: some of the things that make different font families look different is the ratio of the height of lowercase letters to uppercase letters (eg 'n' compared to 'M'), plus how far down the descenders go ('yp'), or how far up the ascenders go ('f'), and so on. It all changes the ratio of the distances from baseline to font bottom, and so on, and can be dramatically different between different fonts. To simply slap the font centrally within the line-height—a most ignorant decision—means that the baselines may not match up between different fonts, which then just looks wrong.

We need to look at CSS *line-box* formatting next, then will return to these text terms.

How the line-box is formatted

The notion of a *"line-box"* is based on the way that the browser will embed text within the centre of the line-height allocated to that text.

The W3C states: "*All elements have a 'line-height' property that, in principle, gives the total height of a line of text. Space is added above and below the text of the line to arrive at that line height ... After formatting, each line will form a rectangular line-box*".



In order to show you this visually I've had to resort to a little print trickery, so this is a simulation rather than an exact show. The box above is identical to the previous one, except that the lower line of characters have been removed & replaced with the identical characters from the top line. Then (using the trickery) I've raised all the text in the lower paragraph so that it sits in the middle, between the 2 hairlines (if you concentrate upon the upright bar, which is *almost* the same height as the font, that becomes clearer). In the end, the top line (pink background) shows you how the print industry does it, and the bottom line (white background) shows you how HTML + CSS does it.

Now, in order to explain this better we need to introduce another important term from the print industry:

Leading

"Leading" is the space between the lines of text and, for hundreds of years, was just that:- a thin strip of lead placed between the blocks that held the lines of text, with the whole thing—paragraphs of text + lead separators—held together with wooden (later metal) vices.

Earlier, it was said that the text in all those pink boxes is '72pt Georgia' (the words in *these* paragraphs are 12pt Georgia). More accurately, the text is '72/84pt Georgia'; the '72 point' is the `font-size`, whilst the '84 point' is the `line-height`. 'Leading' is the difference between the two, so the (virtual) lead placed between each line of 72pt Georgia text is 12 points thick. Notice that in print the lead is placed *above* each line of text, but in HTML + CSS *half* (6 points) is placed above & half is placed below. Also, via the trickery of computers, in HTML the lead can be *negative* (good trick), allowing the text to either be very close (imagine it is ALL CAPITALS) or even overlap.

Pt, Points

"Point" is the standard (and smallest) measure for fonts in the print industry. These days, it is defined as:

72 points == 72pt
== 1 inch
== 2.54cm

em

"em" is the same as the current `font-size`. So, our 72-point Georgia font has an em-width of 72pt (it originally is supposed to come from the width of the letter "M" in a font, although that letter is actually very rarely fully as wide as the font-size).

Kerning

"Kerning" got a brief mention with the first pink box, where it was pointed out that, in the 2 letters "fy", the top of the 'f' extended *over* the top of the left-part of the 'y'. The text in the pink boxes has *"pair kerning"* turned on (as does all this text), and it is *kerning* that allows that to happen. css1 does not have any properties for kerning (though css3 does☞, so let's look at it).

Every computer font that is not `monospaced` has pair-kerning tables as part of the font (each font contains far more than just the characters). These tables apply to *pairs* of characters; one of the most extreme examples is "AV" (look how much *those* two characters overlap!) and, if you think about it just a little, it is obvious that some character-pairs are likely to be able to be kerned, whilst others never will be (how about "OO"?). The information within the tables is unique to each font, and placed there by the font designer.

Why kern? The most obvious answer is that it allows more characters on the line, but another is that it reads easier when kerned and also, somehow, looks more *professional*.

Here is the W3C definition from css3☞: *"Kerning is the contextual adjustment of inter-glyph spacing"*.

Justification

"Justification" has some connections with kerning but is actually different (both involve changing the 'normal' horizontal placing of a text character within the line-box). Kerning involves reducing 'normal' inter-character spacing, whilst *justification* involves changing 'normal' inter-word spacing (although in print it may sometimes also involve the characters *inside* a word, particularly in narrow columns when there is just one word on the line, when that word can be set to be expanded to fit the width). Note that the *last* line in a justified paragraph is not normally justified.

Now we can look at inline-element formatting on the canvas:

Inline-Element Formatting:

Key feature 1: css Properties are not applied directly to text, but instead are applied to HTML elements that can *contain* text.
Key feature 2: all of the main Properties that affect text (colour, font & text) are inherited, with the sole exception of text-decoration & vertical-align.
Key feature 3: different browsers can have different defaults for text (and also for the Box Properties); those defaults can also vary by OS (*"Operating System"*).

The above means that you are strongly advised to place a whole set of css Properties upon `body` within your base stylesheet. `body` is the first HTML element within the HTML object tree that displays upon the canvas, and it is guaranteed to always be present (even if it does not appear within the HTML markup). Doing that will both ensure that your whole site has a similar look 'n' feel, and also that it keeps that look 'n' feel no matter what browser nor OS your readers use. That will help to give a professional style to what you write, and *that* will help to give your readers the confidence that you know what you are doing (and therefore saying).

Inline-elements form part of the flow of HTML elements within a document. Any element with a `display` property of 'block' or 'list-item' (which includes floating elements) is removed from that flow & placed independantly upon the canvas, according to rules given previously within the Block-Level Elements section. For the rest, they are formatted within a line-box, and that line-box flows across the display space and around any block-level elements. The nature of that latter flow is decided by the values of `float` & `clear` for the block-level element(s) (see also Floating Elements).

whitespace:

Having created the line-box of inline-elements, the browser will render it upon the canvas. That will normally create many lines of text (just like these paragraphs; it is called text *'wrapping'*), meaning that the line-box is split into many smaller boxes, each of which are stacked on top of each other, with zero regard for any internal margin, border or padding. The position of those breaks—and whether there are any— is decided by the `white-space` property, since the line-box is wrapped *between* words, and 'whitespace' is defined as those characters that can *not* appear within words.

br:

All of the words in the previous paragraph are now thrown into disarray by the `br` ("line break") element, which forces the flow to be wrapped upon a fresh line at the point where it appears. Somehow, `br` got overlooked by CSS, and none of the CSS1 properties nor values can describe it's behaviour. The W3C says *"CSS1-based formatters must treat 'BR' specially"*.

So far, we have spoken about the line-box as if all that it contains is text. Not only that: there has been an inherent assumption that all the text will be the same font, size & line-height throughout, and that is rarely the case, since mixing up the fonts & sizes in a structured manner can greatly improve the readability of your text. Further, there are non-text inline-elements to take into account. The final sub-sections within this section will try to take all of these variations into account.

Replaced Elements:

Replaced elements can be either Block-Level or inline; we are only interested in inline elements within this section, so that essentially reduces it to either `img` or `object` elements. In practice, you are almost certain to set either of these elements (or any other inline replaced elements) to become floating elements, which then means that they then fall under the block-level element rules. Nevertheless, let's maintain the fiction that they are non-floating, inline replaced elements simply so that you can understand what the heck they are, and how they are formatted within the line-box (which will also assist with understanding what happens within the other sub-sections lower down).

Definition: A replaced element is an element which is replaced by content pointed to from the element.

Let's imagine that, for some crazy reason, you are going to place a 300x300px (pixel) image within your 12px line-height text...

Now, recall from the section on how the line-box is formatted that the text itself is placed character-by-character in the centre of the allotted `line-height`, and that any shortfall between font-height & line-height is filled with space evenly at top & bottom. So, your 10px text—in the `color` of the containing element-- is formatted at plumb centre and 1px leading—in the `background-color` of the containing element—is added at top & bottom. When the browser reaches the image within the HTML flow, it knows where it starts (the next character position within the text-flow), but it cannot know how wide or tall it is unless you tell it.

Hint: always state the (formatted) dimensions for replaced elements within the html, else the browser may have to re-format the entire document below the element once it gets it, which is most unprofessional.

If the html tells the browser what the formatted dimension should be it can continue formatting the line-box. However, it still cannot place the formatted image in the allotted space until it gets it, and that may take many seconds. In addition, it does not know the *intrinsic* size of the image until it arrives (it could be a thumbnail, or the size of a bus, or anything inbetween).

This is how the browser handles the image `width` setting (and similar for `height`):

- `width: auto`
  The intrinsic width is used as the width of the element
- (any other value)
  The value is used and the replaced element is resized accordingly

The final item to consider for both width & height considerations is any replaced-element margin, border or padding properties, which will be added to width & height (that only happens with replaced elements and NOT with non-replaced inline elements, for whom the line-height is the line-height is the height of the line-box).

At this point the browser has the final, final height of the replaced element, and will format it within the centre of the line-box.

We now have an important question:

"What happens if the top of the replaced element is above the top of the text (or the bottom below the text bottom)?"

The answer is that the line-box height (for that line of inline elements) is increased to accommodate the element.

As the final item, all the rectangular line-boxes that make up the various lines of text are stacked immediately below each other, following the flow of HTML elements.

Effects of Padding, Border and/or Margin:

If it is a non-replaced element, then the answer is simple: none!

Imagine that you have some emphasised text, and you want to create the emphasis with a dotted border all around the text, so you format an `em` with some `padding` & a border. Then, any text within the `em` will be formatted in the line-box with a border, separated from the text by the padding. Very nice.

If the font remains the same then the text baseline will be unaffected.
If the `line-height` remains the same then the line-box height will be unaffected, and adjacent line-boxes will be stacked as normal.

If the leading applied is too small & the border extends beyond it... well, the border will extend above & below into neighbouring line-boxes.

Effects of vertical-align:

`vertical-align` allows the affected text to be raised or lowered from it's default position within the line-box. That can dramatically affect the line-box height, and thus the appearance of the text. It is doubly difficult because different browsers have different intrinsic values for the different keywords. Fortunately, there is also a percentage value available, which can be negative...

The html `sub` & `sup` elements (also available as vertical-align keywords) will, by default, noticeably increase the line-height. That is because they not only lower (raise) the text substantially, and—in Firefox24 at least—below (above) the text-top, with vertical-align they also keep the font-size the same (which is one clue as to how to fix it).

They key feature for this section is that, if the top of the text rises into the top leading (and similar at the bottom), then the line-box height will be increased to accommodate the affected section, in exactly the same way as for replaced elements. Please note also that this paragraph is describing the real-world of how browsers do things, rather than how the W3C *says* that it should be done (which is essentially: 'the line-height is the final arbitrator').

Box Properties

Properties:

- `border` (shorthand property)
- `border-width` (shorthand property)
- `clear`
- `float`
- `height`
- `margin` (shorthand property)
- `padding` (shorthand property)
- `width`

border properties:

- `border-bottom` (shorthand property)
- `border-color`
- `border-left` (shorthand property)
- `border-right` (shorthand property)
- `border-style`
- `border-top` (shorthand property)

border-width properties:

- `border-bottom-width`
- `border-left-width`
- `border-right-width`
- `border-top-width`

margin properties:

- `margin-bottom`
- `margin-left`
- `margin-right`
- `margin-top`

padding properties:

- `padding-bottom`
- `padding-left`
- `padding-right`
- `padding-top`

See also: Getting boxy to understand the W3C box-model.

Classification Properties

Properties:

- display
- list-style (shorthand property)
- list-style-image
- list-style-position
- list-style-type
- white-space

Colour & Background Properties

Properties:

- `background` (shorthand property)
- `background-attachment`
- `background-color`
- `background-image`
- `background-position`
- `background-repeat`
- `color`

Notes:

The following are *both* the colour of an element:

- `background` background colour
- `color` foreground colour

One of the important features to note is that the properties are inherited. `color` is inherited directly, but—because the background colour will show through (by virtue of the default 'transparent' value of `background-color`) —it also normally inherits.

Different browsers have different `background` & `color` defaults, so you are strongly advised to set these for `body` as to have a consistent look for the entire document, whatever it is viewed in.

Font Properties

Properties:

- font (shorthand property)
- font-family
- font-size
- font-style
- font-variant
- font-weight

Pseudo-classes & Pseudo-elements

- 1st-line pseudo-element
- 1st-letter pseudo-element
- Anchor pseudo-classes

General Usage

Selectors:

Contextual selector: allowed at the end of the selector:

```
body p:first-letter { color: purple; }
```

Combined with classes:

- one pseudo-element allowed per selector
- normal class-names precede pseudo-class/element name

```
a.external:visited { color: blue; }

p.initial:first-letter { color: red; }
<p class="initial">First paragraph</p>
```

Combination:

Pseudo elements can be combined:

```
p { color: red; font-size: 12pt; }
p:first-letter { color: green; font-size: 200%; }
p:first-line { color: blue; }

<p>Some text that ends up on two lines</p>
```

Pseudo-Class:

Anchor pseudo-classes have no effect on elements other than 'a'. Therefore, the element type can be omitted from the selector. The following 2 statements select the same element in CSS1:

```
a:link { color: red }
 :link { color: red }
```

Text Properties

Properties:

- letter-spacing
- line-height
- text-align
- text-decoration
- text-indent
- text-transform
- vertical-align
- word-spacing

see also:

- font (shorthand property)

Units

Color Units:

This is either a keyword, or a numerical RGB specification.

Keywords: (case-insensitive):
White, Silver. Gray (sic), Black, Red, Maroon, Yellow, Olive, Teal, Aqua, Green, Lime, Blue, Navy, Fuschia & Purple.

These are originally derived from the Windows' VGA Palette. The W3C deliberately does not define those colours in the CSS1 specification (which means that browsers are free to do so). Below are the original VGA Palette settings together with their equivalent CSS numeric units:

white
#fff
#ffffff
rgb(255,255,255)
rgb(100%,100%,100%)
	silver
	#ccc
	#c0c0c0
	rgb(192,192,192)
	rgb(75%,75%,75%)
		gray
		#888
		#808080
		rgb(128,128,128)
		rgb(50%,50%,50%)
			black
			#000
			#000000
			rgb(0,0,0)
			rgb(0%,0%,0%)

red
#f00
#ff0000
rgb(255,0,0)
rgb(100%,0%,0%)
	maroon
	#800
	#800000
	rgb(128,0,0)
	rgb(50%,0%,0%)
		yellow
		#ff0
		#ffff00
		rgb(255,255,0)
		rgb(100%,100%,0%)
			olive
			#880
			#808000
			rgb(128,128,0)
			rgb(50%,50%,0%)

teal #088 #008080 rgb(0,128,128) rgb(0%,50%,50%)	aqua #0ff #00ffff rgb(0,255,255) rgb(0%,100%,100%)	green #080 #008000 rgb(0,128,0) rgb(0%,50%,0%)	lime #0f0 #00ff00 rgb(0,255,0) rgb(0%,100%,0%)

blue
#00f
#0000ff
rgb(0,0,255)
rgb(0%,0%,100%)
	navy
	#008
	#000080
	rgb(0,0,128)
	rgb(0%,0%,50%)
		fuschia
		#f0f
		#ff00ff
		rgb(255,0,255)
		rgb(100%,0%,100%)
			purple
			#808
			#800080
			rgb(128,255,128)
			rgb(50%,100%,50%)

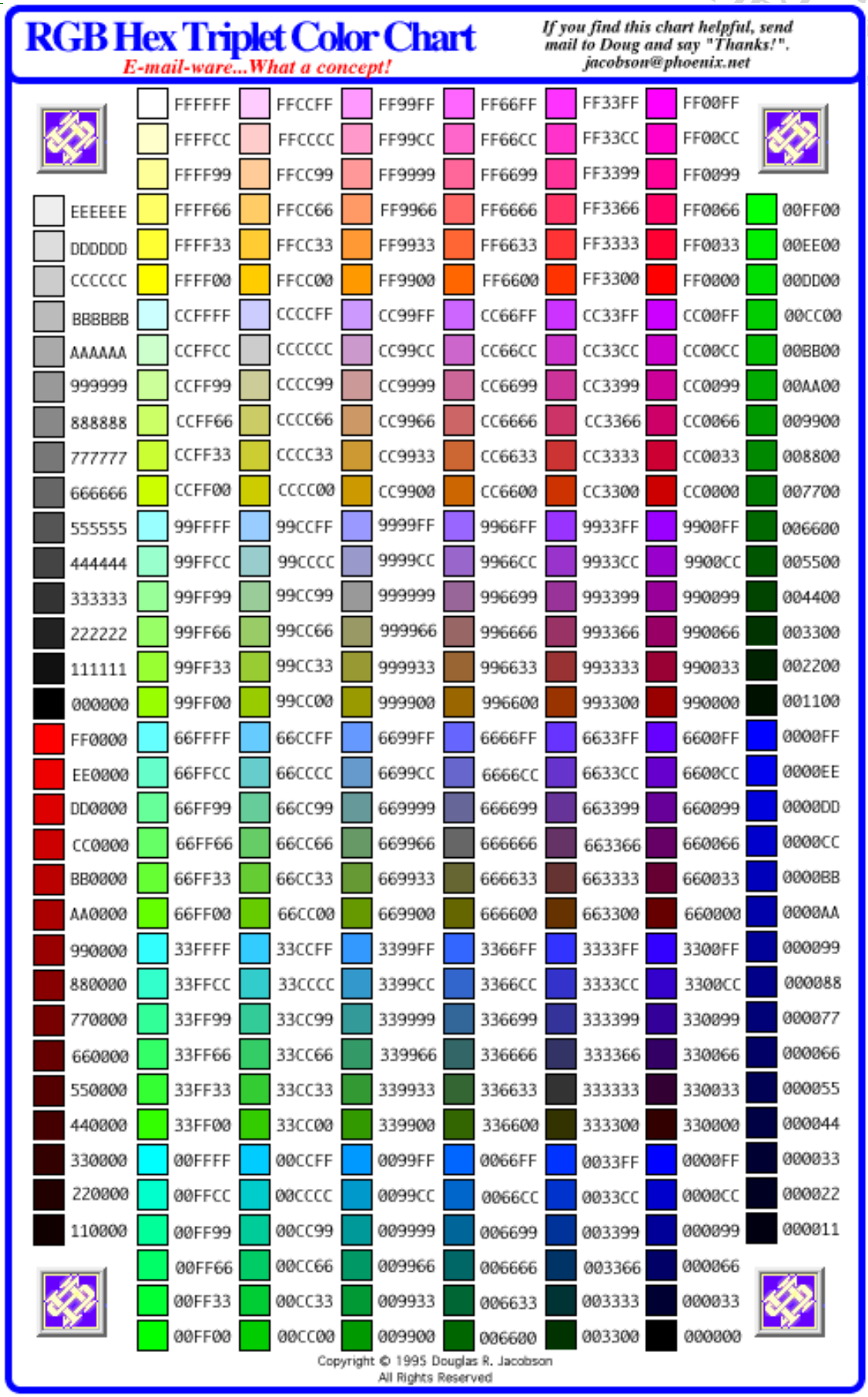
Numeric:

There are 4 formats that may be used; all are specified in the 'sRGB color-space☞' (see also above for examples). All numeric values are clipped by the browser if out of scope:

- #rgb (hex triplets; 12-bit palette; max 4,096 colours)
- #rrggbb (hex triplets; 24-bit palette; max 16,777,216 colours)
- rgb(x,x,x) ('x' = 0 min, 255 max) (integer)
- rgb(y%,y%,y%) ('y' = 0.0% min, 100.0% max)

Extras: See overpage for a very useful hex-triplet colour-chart.
A 24-bit palette contains the max number of colours that a human can see.

Note for UK children:
The Americans cannot spell 'colour' nor 'grey'!



The above chart comes via WDG☞ and is ©1995 Douglas R Jacobson.

Length Units:

Format (in order, no spaces):

1. + / -
   default: '+'; negative units not always allowed (can also hit implementation limits if so)
2. number
   decimal point is optional
3. 2-letter abbeviation
   optional for zero

example:

-0.1em

There are 2 kinds of length units:

1. Relative
   These scale well between different media. Possibly the best example is if you print a webpage, when both a monitor & a printer page should look the same, yet at very different resolutions.
   supported relative units:
   - ***em***
     'em' == font-size = unit of font width/height
     Traditionally this is the height of a capital M in a font, and equal to both it's height & width. The definition has been retained for digital fonts, even though some (non-English) fonts do not contain that character.
   - ***ex***
     'ex' == height of a lowercase 'x' in a font
     Possibly the least-used unit of length. Like 'em', it is defined whether the font contains that letter or not.
   - ***px***
     'px' = 'pixel' (and thus are relative to the canvas)
     This unit would be fine if we knew what the size of the canvas, or even the resolution of the canvas, was, but we cannot, which can make this a most dangerous unit to use.

2. Absolute
   These are dependant on the output medium. That is fine if you know what that medium will be, but can then restrict your audience.
   supported absolute units:
   - ***cm***
     'cm' = 'centimetre'; 1cm == 0.39in
   - ***in***
     'in' = 'inch'; 1in == 2.54cm
   - ***mm***
     'mm' = 'millimetre'; 10mm == 1cm
   - ***pc***
     'pc' = 'pica'; 1pc == 12pt; 6pc == 1in
   - ***pt***
     'pt' = 'point'; 72pt == 1in

Notes:

'*em*' and '*ex*': these are relative to the font size of the element itself, except for '`font-size`', where they refer to the font size of the *parent* element.

Child elements inherit the computed value, not the relative value.

Browsers are allowed to make implementation-specific changes to lengths. An example is font-size, where the font gets mapped to the nearest whole pixel. For all CSS1 properties, further computations and inheritance are then based on the approximated value & not the original value.

Percentage Units:

Format: (in order, no spaces):

1. + / -
   default: '+'; negative units not always allowed (may hit implementation limits if so)
2. number
   decimal point is optional
3. '%'

example: *-120.8%*

Notes: Percentage values are always relative to another value; each property that allows percentage units also defines what value the percentage value refers to. Most often this is the font size of the element itself.

In all inherited CSS1 properties, if the value is specified as a percentage, child elements inherit the resultant value, not the percentage value.

URLs:

Format: (in order):

- 'url('
- white-space
  (optional)
- single quote (') or double quote (") character
  (optional) (must be balanced if included)
- The Uniform Resource Locator (URL) itself
  as defined in RFC1738☞
- white-space
  (optional)
- ')'

example: url(http://www.bg.com/pinkish.gif)

Notes: Parentheses, commas, whitespace characters, single quotes (') and double quotes (") appearing in a URL must be escaped with a backslash:
'\(', '\)', '\,'. '\ ', '\", '\'"

Partial URLs are interpreted relative to the source of the style sheet, not relative to the document.

Hint:

Make life easy on yourself:

1. Do not use partial URLs
2. URL-encode the URL & avoid having to quote (or back-slash) it at all:
   eg
   `"http://a.tld/with two spaces/"`
   ...becomes...
   `http://a.tld/with%20two%20spaces/`

Property Value Syntax

Each CSS *Property* value in this Guide is given in a syntax that may faze you if you are not a seasoned coder. It's not too difficult, and at the bottom of each Property page is an example to try to help.

examples: Some of the syntax that you might expect to see:

- Value: N | NW | NE
- Value: [<length> | thick | thin]{1,4}
- Value: [<family-name> ,]* <family-name>
- Value: <url>? <color> [/ <color>]?
- Value: <url> || <color>

angle brackets:

<length>, <family-name>, <color>, <url>
Words enclosed by *angle brackets* ('<>') are specific *types* of value. If the word is a blue link, then that value *type* may apply to more than one CSS Property, and you will need to click on the link to see what values can be entered. If not, then the type will be described further down the page at "*Extras*".

literals:

N, NW, NE, thick, thin
If *not* enclosed by *angle brackets* then it is a literal *keyword*, and to be used must be entered exactly as shown, without quotes. Any slash ('/') or comma (',') must also appear literally.

a b c
(not exampled at top) More than one thing juxtaposed together means:-
"*all must be entered, and in the order shown*".

upright bar:

<length> | thick | thin
The bar ('|') means "*either/or*", so this example means:-
"<length> *or thick or thin*" (just one)

double-bar:

<url> || <color>
The double-bar ('||') means "*either/both*", so this example means:-
"<*url*> *or* <*color*>*,or both*" (in any order)

square brackets:

[<length> | thick | thin]
Square brackets ('[]') are a *grouping* mechanism. This makes more sense with the modifiers (bottom).

a b | c || d e
[a b] | [c || [d e]]
(not exampled at top–the two examples are equivalent to each other)

- Juxtaposition is stronger than the double bar
- The double bar is stronger than the bar

modifiers:

*	the preceding type, word or group is repeated **zero or more** times
+	the preceding type, word or group is repeated **one or more** times
?	the preceding type, word or group is **optional**
{1,4}	the preceding is repeated at least **1**, and at most, **4** times

CSS Properties A – Z

1st-line pseudo-element

Syntax: `:first-line`

Description: The 'first-line' pseudo-element is used to apply special styles to the first line as formatted on the canvas. Derived from extensive use in newspapers & magazines, it is little-used in web documents. There are a limited set of CSS properties that will be accepted with this element (see 'Extras' below).

See also: `Anchor pseudo-classes`, `1st-line pseudo-element`, `1st-letter pseudo-element`

Properties:

Group:	Pseudo-classes and Pseudo-elements
Values:	(n/a)
Initial values:	(not defined for pseudo-elements)
% values:	(n/a)
Applies to:	Block-level elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

Accepted CSS Properties:	• `clear` • Colour & Background Properties • Font Properties • `letter-spacing` • `line-height` • `text-decoration` • `text-transform` • `vertical-align` • `word-spacing`
Extra:	CSS1 Core conformance☞ allows all browsers to ignore all rules with ':first-line' or ':first-letter' in the selector or, alternatively, only support a subset of the properties on these pseudo-elements.

Examples:

```
p:first-line { font-variant: small-caps; }
```

THE ABOVE SPECIFIES, FOR THE FIRST LINE OF TEXT IN ALL PARAGRAPH ELEMENTS, that small capitals should be used.

1st-letter pseudo-element

Syntax: `:first-letter`

Description: The 'first-letter' pseudo-element is used to apply special styles to the first letter as formatted on the canvas. Derived from extensive use in newspapers & magazines, it is little-used in web documents. There are a limited set of CSS properties that will be accepted with this element (see 'Extras' below).

See also: `Anchor pseudo-classes`, `1st-line pseudo-element`, `1st-letter pseudo-element`

Properties:

Group:	Pseudo-classes and Pseudo-elements
Values:	(n/a)
Initial values:	(not defined for pseudo-elements)
% values:	(n/a)
Applies to:	Block-level elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

Accepted CSS Properties:	• border properties • `clear` • Colour & Background Properties • `float` • Font Properties • `line-height` • margin properties • padding properties • `text-decoration` • `text-transform` • `vertical-align` (only if 'float' is 'none') • `word-spacing`
Extra:	CSS1 Core conformance☞ allows all browsers to ignore all rules with ':first-line' or ':first-letter' in the selector or, alternatively, only support a subset of the properties on these pseudo-elements.

Examples:

```
p              { font-size: 12pt; line-height: 12pt; }
p:first-letter { font-size: 200%; float: left; }
span           { text-transform: uppercase; }
```

The above specifies, for text in all paragraph elements, that a dropcap initial letter spanning two lines should be used.

Anchor pseudo-classes

Syntax: `a:<pseudo-class-values>`

Description: A browser will typically use different styles for new links & recently-visited links, and yet another style when the mouse hovers over a link; here's how. This pseudo-class acts only on `a` elements with an 'href' attribute; each one is then moved into one (only) of 3 groups. Note: 'active' == 'hover'.

See also: `Anchor pseudo-classes`, `font-variant`, `font-weight`

Properties:

Group:	Pseudo-classes and Pseudo-elements
Values:	active | link | visited
Initial values:	(not defined for pseudo-class properties)
% values:	(n/a)
Applies to:	`a` elements with 'href' attribute
Inherited?	Yes
Standard:	CSS1☞

Extras:

misc:

- Pseudo-class & normal class selectors will not match
- Case-insensitive
- target anchors are unaffected
- Can be used *contexually*:
```
a:link img { border: solid blue; }
```
- Can be combined with normal classes:
```
a.ext:visited { color: blue; }
<a class="ext" href="http://out.side/">outside</a>
```
(normal class names precede pseudo-class names in the selector)
- Reformatting is the browser's choice
The 3 pseudo-classes may format on the canvas with different line-heights, widths, etc. Therefore a pseudo-class-transition—particularly between 'active' & either of the other two—may normally require a page re-format.

Examples:

```
a:link { color: red; }        /* unvisited link */
a:visited { color: blue; }     /* visited link */
a:active { color: lime; }       /* active link */
```

The above specifies, for all a elements with a 'href' attribute, the text colours used within their differing states.

background

Syntax: `background: <values>`

Description: `background` is a shorthand property for setting any/all of the individual background properties.

See also: `background`, `background-attachment`, `background-color`, `background-image`, `background-position`, `background-repeat`, `color`

Properties:

Group:	Colour & Background Properties
Values:	<background-color> || <background-image> || <background-repeat> || <background-attachment> || <background-position>
Initial values:	(not defined for shorthand properties)
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

Values:	See the individual properties for all the values that refer.
Note:	`background` always sets all the individual background properties. Any missing properties are set to their initial value.

Examples:

```
p { background: url(chess.png) gray 50% repeat fixed }
```

The above specifies, as a default for paragraphs, ***all the possible background properties***.

background-attachment

Syntax: `background-attachment: <values>`

Description: `background-attachment` is effective when a `background-image` has been specified, and sets whether it is fixed with regard to the canvas, or if it scrolls along with the content.

See also: `background`, `background-attachment`, `background-color`, `background-image`, `background-position`, `background-repeat`, `color`

Properties:

Group:	Colour & Background Properties
Values:	scroll | fixed
Initial values:	scroll
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

Extra:	CSS1 Core conformance☞ allows all browsers to treat 'fixed' as 'scroll'. (though the W3C hopes that 'fixed' will be supported on at least the `html` & `body` elements).

Examples:

```
body { background-attachment: scroll; }
```

The above specifies, (as is the default) for the entire document, that ***a background image will scroll with the content.***

background-color

Syntax: `background-color: <values>`

Description: `background-color` sets the *background* colour of an element.

See also: `background`, `background-attachment`, `background-color`, `background-image`, `background-position`, `background-repeat`, `color`

Properties:

Group:	Colour & Background Properties
Values:	<color> | transparent
Initial values:	transparent
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

A good habit:	Get into the habit—if you specify either `color` or `background-color`—of always specifying *both*. It will save you much future grief.
Inherited:	In spite of not being inherited, the default of 'transparent' means that the colour of parent elements will always show through. In addition, different browsers on different platforms may have different values for the `body` element, so you are well advised to set both the base `color` & `background-color` for all of your own docs.
Accessibility:	Folks are all different. Some have 20:20 vision, others need to expand the text to have a chance to read it. Some can see all 16 million colours in a 24-bit palette, yet red-green is the commonest colour-blindness☞ around (7% of all men). *It is simple:* you want it to look good, but give a little thought as you design your pages, so that all strokes of folks have a good chance of not thinking it to be completely *naff*.

Examples:

```
body { color: yellow; background-color: aqua; }
```

The above specifies, as a default for the entire document, a *splendid combo of yellow text on an aqua background*.

background-image

Syntax: `background-image: <values>`

Description: `background-image` sets the background image of an element. When available, the image will overlay the background colour.

See also: `background`, `background-attachment`, `background-color`, `background-image`, `background-position`, `background-repeat`, `color`

Properties:

Group:	Colour & Background Properties
Values:	<url> | none
Initial values:	none
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

A good habit:	Get into the habit of *also* setting `background-color` for when the image is inaccessible.

Examples:

```
body { background-image: url(marble.gif); color: white; }
```

The above specifies, as a default for the entire document, a ***marbled background, with a default of 'white'.***

background-position

Syntax: `background-position: <values>`

Description: `background-position` is effective when a `background-image` has been specified, and sets it's initial postion. See '*Extras*' for values.

See also: `background`, `background-attachment`, `background-color`, `background-image`, `background-position`, `background-repeat`, `color`

Properties:

Group:	Colour & Background Properties
Values:	[<percentage> | <length>]{1,2} | [top | center | bottom] || [left | center | right]
Initial values:	0% 0%
% values:	refer to the size of the element itself
Applies to:	Block-level and replaced elements
Inherited?	No
Standard:	CSS1

Extras:

Values:	*Important!*: 'top left' in all below refers to the top-left point of the element *content*, and not to the boxes that surround the padding, border or margin. Combinations of 'length' and '%' values are allowed (eg '50% 2cm'), as are negative values. Keywords are single or pairs, but can NOT be combined with '%' nor 'length' values. `background-attachment`: If the background image is fixed with regard to the canvas, the image is placed relative to the canvas rather than the element.
'%' values:	'0% 0%' == 'top left'. '100% 100%' == the bottom-right of image is at element bottom-right. '14% 84%' == the point 14% across and 84% down the image is to be placed at the point 14% across and 84% down the element.
'length' values:	'2cm 2cm': the upper left corner of the image is placed 2cm to the right and 2cm below the upper left corner of the element. (cont next page)

Two '%' or 'length' values:	The 1st value will be the horizontal (x) position; the 2nd value will be the vertical (y) position.
One '%' or 'length' value:	The value will be the x-position, whilst the y-position will be 50%.
Keywords:	'top left' == 'left top' == '0% 0%' 'top' == 'top center' == 'center top' == '50% 0' 'top right' == 'right top' == '100% 0%' 'left' == 'left center' == 'center left' == '0% 50%' 'center' == 'center center' == '50% 50%' 'right' == 'right center' == 'center right' == '100% 50%' 'bottom left' == 'left bottom' == '0% 100%' 'bottom' == 'bottom center' == 'center bottom' == '50% 100%' 'bottom right' == 'right bottom' == '100% 100%'

Examples:

```
body { background: url(banner.jpeg) right top; }
```

The above specifies, as a default for the entire document, a ***banner placed at the top right of the entire content.***

background-repeat

Syntax: `background-repeat: <values>`

Description: `background-repeat` specifies if—and only if—a `background-image` is also specified, how (and whether) that image will be repeated. See '*Extras*' for the meaning of the *Values*.

See also: `background`, `background-attachment`, `background-color`, `background-image`, `background-position`, `background-repeat`, `color`

Properties:

Group:	Colour & Background Properties
Values:	repeat | repeat-x | repeat-y | no-repeat
Initial values:	repeat
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

Values:	• repeat: both horizontally and vertically • repeat-x: just horizontally • repeat-y: just vertically • no-repeat: don't bother with either

Examples:

```
body { background-repeat: repeat; }
```

The above specifies, as a default for the entire document, to ***repeat that annoying gif endlessly across the entire document.***

border

Syntax: `border: <values>`

Description: `border` is a shorthand property for setting the same width, colour and style on all four borders of an element at once. Unlike the shorthand '`margin`' and '`padding`' properties, '`border`' cannot set different values on the four borders. To do so, one or more of the other border properties must be used.

See also: `border`, `border-top`, `border-right`, `border-bottom`, `border-left`

Properties:

Group:	Box Properties
Values:	<border-width> || <border-style> || <color>
Initial values:	(not defined for shorthand properties)
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

One value only for each:	One value only may be given for each of `border-width`, `border-style` & color (a total of one, two or three values).
Order+ specificity is important:	Due to overlapping functionality, the order in which the rules are (or not) specified can become important: `blockquote { border-color: red; border-left: double; color: black; }` The lack of a 'color' specification in `border-left` causes the left border to be black, whilst others are red.

Examples:

```
p { border: thin solid red; }
```

The above specifies, for all paragraphs, a ***thin solid red border.***

border-bottom

Syntax: `border-bottom: <values>`

Description: `border-bottom` is a shorthand property for setting the same width, colour and style on an element's bottom border. Any omitted values will be set to their initial values (browser determined if not inherited).

See also: `border`, `border-top`, `border-right`, `border-bottom`, `border-left`

Properties:

Group:	Border Properties
Values:	`<border-bottom-width>` || `<border-style>` || <color>
Initial values:	(not defined for shorthand properties)
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

One value only for each:	One value only may be given for each of `border-bottom-width`, `border-style` & color (a total of one, two or three values).

Examples:

```
p { border-bottom: thin solid blue; }
```

The above specifies, for all paragraphs, a ***thin solid blue bottom-border.***

border-bottom-width

Syntax: `border-bottom-width: <values>`

Description: `border-bottom-width` sets the width of an element's bottom border, using a keyword or <length>.

See also: `border`, `border-top-width`, `border-right-width`, `border-bottom-width`, `border-left-width`, `border-width`

Properties:

Group:	Border Width Properties
Values:	thin | medium | thick | <length>
Initial values:	medium
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Extra:	○ Border widths cannot be negative. ○ Keyword values are browser-dependent, but the following will always hold true: ● 'thin' <= 'medium' <= 'thick' ● Keyword widths will remain constant throughout a document, regardless of font-width.

Examples:

```
h5 { border-bottom-width: thick; }
```

The above specifies, for `h2` headers, to use a ***thick bottom-border.***

border-color

Syntax: `border-color: <values>`

Description: `border-color` is a property for setting the colour on all 4 borders at once; colours may be all the same (1 color value required) or all different (4 color values required). If 2 or 3 values are given, then missing values are taken from the opposite side. See '*Extras*' for the ordering of values.

See also: `border`, `border-color`, `border-style`

Properties:

Group:	Border Properties
Values:	<color>{1,4}
Initial values:	Value of the 'color' property
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Order & Values:	One value only may be given to set all 4 borders to the same colour. Any other number of color values affects the borders in a specific order: 1. top 2. right 3. bottom 4. left 2 values: 1. top & bottom 2. right & left 3 values: 1. top 2. right & left 3. bottom Zero color values: the value of the 'color' property of the element itself will take its place

Examples:

```
p { border-color: blue navy fuschia purple; }
```

The above specifies, for all paragraphs, a ***blue top-, navy right-, fuschia bottom- & purple left-border.***

border-left

Syntax: `border-left: <values>`

Description: `border-left` is a shorthand property for setting the same width, colour and style on an element's left border. Any omitted values will be set to their initial values (browser determined if not inherited).

See also: `border`, `border-top`, `border-right`, `border-bottom`, `border-left`

Properties:

Group:	Border Properties
Values:	`<border-left-width>` || `<border-style>` || <color>
Initial values:	(not defined for shorthand properties)
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

One value only for each:	One value only may be given for each of `border-left-width`, `border-style` & color (a total of one, two or three values).

Examples:

```
p { border-left: thick solid green; }
```

The above specifies, for all paragraphs, a ***thick solid green left-border.***

border-left-width

Syntax: `border-left-width: <values>`

Description: `border-left-width` sets the width of an element's left border, using a keyword or <length>.

See also: `border`, `border-top-width`, `border-right-width`, `border-bottom-width`, `border-left-width`, `border-width`

Properties:

Group:	Border Width Properties
Values:	thin | medium | thick | <length>
Initial values:	medium
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Extra:	○ Border widths cannot be negative. ○ Keyword values are browser-dependent, but the following will always hold true: ● 'thin' <= 'medium' <= 'thick' ● Keyword widths will remain constant throughout a document, regardless of font-width.

Examples:

```
h5 { border-left-width: thick; }
```

The above specifies, for `h3` headers, to use a ***thick left-border.***

border-right

Syntax: `border-right: <values>`

Description: `border-right` is a shorthand property for setting the same width, colour and style on an element's right border. Any omitted values will be set to their initial values (browser determined if not inherited).

See also: `border`, `border-top`, `border-right`, `border-bottom`, `border-left`

Properties:

Group:	Border Properties
Values:	`<border-right-width>` || `<border-style>` || <color>
Initial values:	(not defined for shorthand properties)
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

One value only for each:	One value only may be given for each of `border-right-width`, `border-style` & color (a total of one, two or three values).

Examples:

```
p { border-right: thick solid olive; }
```

The above specifies, for all paragraphs, a ***thick solid olive right-border.***

border-right-width

Syntax: `border-right-width: <values>`

Description: `border-right-width` sets the width of an element's right border, using a keyword or <length>.

See also: `border`, `border-top-width`, `border-right-width`, `border-bottom-width`, `border-left-width`, `border-width`

Properties:

Group:	Border Width Properties
Values:	thin | medium | thick | <length>
Initial values:	medium
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Extra:	○ Border widths cannot be negative. ○ Keyword values are browser-dependent, but the following will always hold true: ● 'thin' <= 'medium' <= 'thick' ● Keyword widths will remain constant throughout a document, regardless of font-width.

Examples:

```
h5 { border-right-width: thick; }
```

The above specifies, for `h4` headers, to use a ***thick right-border.***

border-style

Syntax: `border-style: <values>`

Description: `border-style` is a property for setting the style on all 4 borders at once; styles may be all the same (1 value required) or all different (4 values required). If 2 or 3 values are given, then missing values are taken from the opposite side. See '`border-color`' '*Extras*' for the ordering of values.

See also: `border`, `border-color`, `border-style`

Properties:

Group:	Border Properties
Values:	none | dotted | dashed | solid | double | groove | ridge | inset | outset
Initial values:	none
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

Value meanings:	• none no border (regardless of the '`<border-width>`' value) • dotted a dotted line drawn on top of the background of the element • dashed a dashed line drawn on top of the background of the element • solid the border is a solid line • double a double line drawn on top of the background of the element. The sum of the two single lines and the space between equals the '`<border-width>`' value • groove a 3D groove is drawn in colours based on the <color> value • ridge a 3D ridge is drawn in colours based on the <color> value. • inset a 3D inset is drawn in colours based on the <color> value • outset a 3D outset is drawn in colours based on the <color> value
CSS1 Core☞	Browsers may interpret all of 'dotted', 'dashed', 'double', 'groove', 'ridge', 'inset' and 'outset' as 'solid'

Examples:

```
#xy34 { border-style: solid double; }
```

The above specifies, for id #xy34, ***solid & double (up) borders.***

border-top

Syntax: `border-top: <values>`

Description: `border-top` is a shorthand property for setting the same width, colour and style on an element's top border. Any omitted values will be set to their initial values (browser determined if not inherited).

See also: `border`, `border-top`, `border-right`, `border-bottom`, `border-left`

Properties:

Group:	Border Properties
Values:	`<border-top-width>` || `<border-style>` || <color>
Initial values:	(not defined for shorthand properties)
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

One value only for each:	One value only may be given for each of `border-top-width`, `border-style` & color (a total of one, two or three values).

Examples:

```
p { border-top: medium solid teal; }
```

The above specifies, for all paragraphs, a ***medium double teal top-border.***

border-top-width

Syntax: `border-top-width: <values>`

Description: `border-top-width` sets the width of an element's top border, using a keyword or <length>.

See also: `border`, `border-top-width`, `border-right-width`, `border-bottom-width`, `border-left-width`, `border-width`

Properties:

Group:	Border Width Properties
Values:	thin | medium | thick | <length>
Initial values:	medium
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Extra:	○ Border widths cannot be negative. ○ Keyword values are browser-dependent, but the following will always hold true: ● 'thin' <= 'medium' <= 'thick' ● Keyword widths will remain constant throughout a document, regardless of font-width.

Examples:

```
h5 { border-top-width: thick; }
```

The above specifies, for `h5` headers, to use a ***thick top-border.***

border-width

Syntax: `border-width: <values>`

Description: `border-width` is a shorthand property (keyword or <length>) for setting the width on all 4 borders at once; widths may be all the same (1 value), all different (4 values) or a mixture (2 or 3 values—missing values are taken from the opposite side). See '`border-color`' '*Extras*' for ordering of values.

See also: `border`, `border-top-width`, `border-right-width`, `border-bottom-width`, `border-left-width`, `border-width`

Properties:

Group:	Box Properties
Values:	[thin | medium | thick | <length>]{1,4}
Initial values:	(not defined for shorthand properties)
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

Extra:	○ Border widths cannot be negative. ○ Keyword values are browser-dependent, but the following will always hold true: ● 'thin' <= 'medium' <= 'thick' ● Keyword widths will remain constant throughout a document, regardless of font-width.

Examples:

```
h6 { border-width: thin; }
```

The above specifies, for `h6` headers, to use a ***thin border.***

clear

Syntax: `clear: <values>`

Description: `clear` specifies if an element allows floating elements on its sides. More specifically, the value of this property lists the sides where floating elements are not accepted. With `clear` set to 'left', an element will be moved below any floating element on the left side. With `clear` set to 'none', floating elements are allowed on all sides.

See also: `border`, `clear`, `float`, `height`, `margin`, `padding`, `width`

Properties:

Group:	Box Properties
Values:	none | left | right | both
Initial values:	none
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

Extra:	The obvious connection is with the 'clear' attribute of `br` (introduced in HTML3.2). CSS allows a similar function to be applied to *any* element.

Examples:

```
table { clear: left; }
```

The above specifies, for all table elements, that they will be displayed *below* all floating elements at their left.

color

Syntax: `color: <values>`

Description: `color` sets the *foreground* text colour of an element.

See also: `background`, `background-attachment`, `background-color`, `background-image`, `background-position`, `background-repeat`, `color`

Properties:

Group:	Colour & Background Properties
Values:	<color>
Initial values:	(browser specific)
% values:	(n/a)
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

A good habit:	Get into the habit—if you specify either `color` or `background-color`—of always specifying *both*. It will save you much future grief.

Examples:

```
body { color: maroon; background-color: fuschia; }
```

The above specifies, as a default for the entire document, a ***delightful combo of maroon text on a fuschia background***.

display

Syntax: `display: <values>`

Description: `display` is an exciting property that governs how—and indeed whether—an element will be displayed on the canvas.

See also: `display`, `list-style`, `list-style-image`, `list-style-position`, `list-style-type`, `white-space`

Properties:

Group:	Classification Properties
Values:	block | inline | list-item | none
Initial values:	block (though also refer to HTML element default)
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Extra:	The HTML Content model is broadly split between block-level & inline elements ('list-items' are also block-level, with the extra that they also have a list-item marker). `display` allows the *author* to decide which of the two models the element should adhere to: ○ *block*: opens a new block-box, positioned relative to adjacent boxes according to the CSS formatting model. ○ *inline*: opens a new inline-box on the same line as the previous content; dimensions are according to the formatted size of the content. `margin`, `border` and `padding` properties apply to inline elements, but do not have any effect at line breaks. ○ *list-item*: it does not make any sense to apply this to any elements other than `li`, `ol` & `ul`. ○ *none*: turns off display of the element, including child elements and the surrounding box.
CSS1 Core:	Browsers may ignore `display` and use only defaults.

Examples:

```
em { display: inline; }
```

The above specifies, for `em` text, to ***display inline*** (which is also the default for `em` text)

float

Syntax: `float: <values>`

Description: `float` can invalidate the `display` property. With a value of 'none' the element will be displayed where it appears within the text flow; all other values will cause it to be treated as a block-level element (invalidating `display`). 'left' will cause it to be moved left until the nearest block-level element is reached & text will wrap to the right (all vice-versa for 'right').

See also: `border`, `clear`, `float`, `height`, `margin`, `padding`, `width`

Properties:

Group:	Box Properties
Values:	left | right | none
Initial values:	none
% values:	(n/a)
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

General:	`float` governs how text wraps around an element. The obvious connection is with the 'align' attribute of `img` (introduced in HTML3.2). CSS allows a similar function to be applied to *any* element.
Specifics:	By setting `float` to 'left': The element is moved to the left until the margin, padding or border of another block-level element is reached (most often it's parent, containing element). The normal flow will wrap around on the right side. The margins, borders and padding of the element itself will be honoured, and the margins never collapse with the margins of adjacent elements.

Examples:

```
img { float: right; }
```

The above specifies, for all `img` elements, that they will be displayed at the side of the nearest block-level element to their right, and that surrounding text—and any other inline elements—will wrap to their left-hand side.

font

Syntax: `font: <values>`

Description: `font` is a shorthand for *all* the `font` properties + `line-height` at once (a 'font' is the typeface used in a visual browser to show text). The W3C says☞: *"The syntax of this property is based on a traditional typographical shorthand notation to set multiple properties related to fonts".*

See also: `font`, `font-family`, `font-size`, `font-style`, `font-variant`, `font-weight`

Properties:

Group:	Font Properties
Values:	[<font-style> || <font-variant> || <font-weight>]? <font-size> [/ <line-height>]? <font-family>
Initial values:	(not defined for shorthand properties)
% values:	allowed on <font-size> and <line-height>
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

Font-Matching: per-property matching rules:	1. '`font-style`' is tried first. 'italic' will be satisfied if there is either a face in the UA's font database labeled with the CSS keyword 'italic' (preferred) or 'oblique'. Otherwise the values must be matched exactly or font-style will fail. 2. '`font-variant`' is tried next. 'normal' matches a font not labeled as 'small-caps'. (A small-caps font can be synthesized electronically) 3. '`font-weight`' is matched next, it will never fail. 4. '`font-size`' must be matched within a UA-dependent margin of tolerance. (Typically, sizes for scalable fonts are rounded to the nearest whole pixel, while the tolerance for bitmapped fonts could be as large as 20%.) Further computations, e.g. by 'em' values in other properties, are based on the 'font-size' value that is used, not the one that is specified.

Examples:

```
p { font: italic bold 12pt/14pt Times, serif; }
```

The above specifies, for all paragraphs, a ***12-point bold, italic Times font*** with a 14-point line height; if the (specific name) "Times" font is not available to the browser then it will use a generic serif font instead.

font-family

Syntax: `font-family: <values>`

Description: `font-family` is a prioritised list of font family names and/or generic family names; unlike most other CSS1 properties, values are separated by a comma to indicate that they are alternatives (fonts come in families of same-style typefaces). Names containing whitespace must be quoted.

See also: `font`, `font-family`, `font-size`, `font-style`, `font-variant`, `font-weight`

Properties:

Group:	Font Properties
Values:	[[<family-name> | <generic-family>],]* [<family-name> | <generic-family>]
Initial values:	Determined by browser
% values:	(n/a)
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

family-name:	The name of a specific Family. Can be dangerous, as it is dependant on the browser being upon a device that has that specific Family of fonts installed.
generic-family:	(below is the generic, followed by a family example): • serif (e.g., Times) • sans-serif (e.g., Arial or Helvetica) • *cursive* (e.g., *Zapf-Chancery*) • *fantasy* (e.g., Western) • `monospace` (e.g., `Courier`) Firefox contains it's own fonts for all the generics above, so I cannot reproduce the browser view here; I've chosen fonts above to try to approximate ('cursive', 'fantasy' are the least accurate).

Examples:

```
body { font-family: "New Century Schoolbook", serif; }
```

The above specifies, as the default font for the document, a 'New Century Schoolbook' font (use either single- or double-quotes if there is whitespace in the name) with a generic serif font if the first is not available.

font-size

Syntax: `font-size: <values>`

Description: `font-size` is, at first sight, a straightforward property: the size of the font. The key feature to keep in mind is that you cannot guess at the size of the view-window that your user has, which makes using *absolute* sizes (length) most dangerous.

See also: `font`, `font-family`, `font-size`, `font-style`, `font-variant`, `font-weight`

Properties:

Group:	Font Properties
Values:	<absolute-size> | <relative-size> | <length> | <percentage>
Initial values:	medium
% values:	Relative to the parent element's font size
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1

Extras:

absolute-size:	Possible values (note: this text is 10pt; 'px' transposed to 'pt' below): xx-small | x-small | small | medium | large | x-large | xx-large In Firefox (parent:16px) these rendered as: 9px | 10px | 13px | 16px | 18px | 24px | 32px
relative-size:	These are intended to be relative to the parent element: larger | smaller In Firefox (parent:16px) these rendered as: 18px | 13px

Examples:

```
em { font-size: 150%; }
```

The above specifies, for all emphasised text, a *larger font*; the W3C suggest to browser manufacturers that they use a 1.5 scaling factor.

font-style

Syntax: `font-style: <values>`

Description: `font-style` selects between normal (sometimes referred to as "roman" or "upright"), *italic* and *oblique* (slanted) faces within a font family. The font that is labeled 'oblique' in the browser's font database may actually have been generated by electronically slanting a normal font.

See also: `font`, `font-family`, `font-size`, `font-style`, `font-variant`, `font-weight`

Properties:

Group:	Font Properties
Values:	normal | italic | oblique
Initial values:	normal
% values:	(n/a)
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

italic v oblique:	Most font families have one font that is 'italic'. Few have one that is 'oblique'. In general, the oblique option is *more* slanted than the italic option. *Historical notes:* 'italic' gained that name in England to differentiate it from the *"English chancery hand"*. 'Chancery-hand' in 16th Century England was derived from Gothic script (similar to *blackletter*) & was angular; The 'Italian-hand' was developed in Italy at that period; the pen was held slanted at a forty-five-degree angle, which allowed greater speed & also produced slightly slanted writing. All official documents at the time were hand-written & it was a legal requirement that they be produced in *Chancery-hand* (itself also scrupilously regulated). The development of the printing-press & typesetting eventually put many thousands of trained clerks out of business (and provided possibly less well-paid employment for many thousands more).

Examples:

```
h1, h2, h3 { font-style: italic; }
```

The above specifies, for h1, h2 & h3 headings, an *italic font*. We could then, perhaps, say to use normal text within h1 *emphasised* headings:

```
h1 em { font-style: normal; }
```

font-variant

Syntax: `font-variant: <values>`

Description: `font-variant` selects between a normal font or a SMALL-CAPS font.

See also: `font`, `font-family`, `font-size`, `font-style`, `font-variant`, `font-weight`

Properties:

Group:	Font Properties
Values:	normal | small-caps
Initial values:	normal
% values:	(n/a)
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1

Extras:

What it is:	The browser *should* always get this right. Some font-families have a small-caps font; many do not. It *should* be easy to synthesize it... If you look at this type you will notice that all the lowercase letters are on the same level. A *small-caps* font is supposed to have all-capitals that are at (slightly more than) the same height as the lowercase letters. Let's see if it works with this font: CAPITALSSMALL-CAPSlowercase ...and the answer is *yes* (it looks OK).

Examples:

```
span { font-variant: small-caps; }
```

The above specifies, for all text within the `span` element, a SMALL-CAPS FONT.

font-weight

Syntax: `font-weight: <values>`

Description: `font-weight` is a method of specifing the *weight* of a font. Child elements inherit the *resultant* weight of the font, and not the *keyword*. Few font families have sufficient weights to populate all 9 levels, so choices must be made by the browser.

See also: `font`, `font-family`, `font-size`, `font-style`, `font-variant`, `font-weight`

Properties:

Group:	Font Properties
Values:	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900
Initial values:	(not defined for shorthand properties)
% values:	(n/a)
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

Help:	The weight of a font is different from it's size; whatever the weight, a 12pt/14pt font should always be 12 points high with a 14 point line-height. That can possibly be most easily understood with **bold text** (which most often has "`font-weight: 700`" - it depends on both the font, the browser & the host device). It is *very* seldom that a font has even 3 weights in the family, which essentially then reduces the values to "normal | bold". This is what those values look like on Firefox23 using the browser-supplied, generic serif font: normal | **bold** | **bolder** | lighter 100 | 200 | 300 | 400 | 500 | **600** | **700** | **800** | **900** ' bolder | lighter' refer to inherited values from the parent.

Examples:

```
strong { font: 900; }
```

The above specifies, for all `strong` text, the **most strongly weighted font** should be used.

height

Syntax: `height: <values>`

Description: `height` is suggested to be used principally with replaced elements such as `img`. The element will be scaled if necessary (see *Extras* below).

See also: `border`, `clear`, `float`, `height`, `margin`, `padding`, `width`

Properties:

Group:	Box Properties
Values:	<length> | auto
Initial values:	auto
% values:	(n/a)
Applies to:	Block-level and replaced elements
Inherited?	No
Standard:	CSS1☞

Extras:

General:	Both `height` & `width` are considered together to find a replaced element's eventual dimensions: • `height` = 'auto'; `width` = 'auto': set to intrinsic dimensions of element • `height` = 'auto'; `width` = <length>: `width` enforced by scaling, aspect ratio preserved • `height` = <length>; `width` = 'auto': `height` enforced by scaling, aspect ratio preserved • `height` = <length>; `width` = <length>: enforced by scaling, aspect ratio ignored
CSS1 Core☞:	Browsers may ignore `height` (treat it as 'auto') if the element is not a replaced element.

Examples:

```
img.icon { height: 100px }
```

The above specifies, for all `img` elements declared with a class of 'icon', that they will be displayed at a height of 100 pixels (note how that could be an issue on a very high-density display device).

letter-spacing

Syntax: `letter-spacing: <values>`

Description: `letter-spacing` normalises / expands / contracts the *spacing* between letters. Negative values *are* allowed. Any value other than 'normal' will switch justification OFF. The browser is the ultimate resultant arbitrator.

See also: `font`, `letter-spacing`, `line-height`, `text-align`, `text-decoration`, `text-indent`, `text-transform`, `vertical-align`, `word-spacing`

Properties:

Group:	Text Properties
Values:	normal | <length>
Initial values:	normal
% values:	(n/a)
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

length:	<length> here is an addition/subtraction from the 'normal' intra-character spacing. Too much extra will make the text look strange. Too much removed may hit implementation limits.
normal:	CSS1 Core conformance☞ allows the browser to interpret *any* value of `letter-spacing` as 'normal'.
Extra:	`letter-spacing` is an internal function of the font first & the browser second. Each font contains information not just on the *glyph* (the font's picture of the character, whether that picture is a bitmap or a mathematical outline) but also on the 'normal' spacing between glyphs. As an implementor, the browser is then free to modify (expand/contract) those spacings, which is where this property has it's effect. The actual spacing used, however, will be *highly* context-specific, dependant on the display resolution & size, text size, etc..

Examples:

```
blockquote { letter-spacing: 0.1em; }
```

The above specifies, for all `blockquote` elements, a *0.1em increase in the letter spacing* of all text within the element.

line-height

Syntax: `line-height: <values>`

Description: `line-height` sets the distance between two adjacent text-lines' *baselines*. Negative values are *not* allowed.

See also: `font`, `letter-spacing`, `line-height`, `text-align`, `text-decoration`, `text-indent`, `text-transform`, `vertical-align`, `word-spacing`

Properties:

Group:	Text Properties
Values:	normal | <number> | <length> | <percentage>
Initial values:	normal
% values:	Relative to the `font-size` of the element itself
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1

Extras:

Help:	`line-height` is normally a little larger than the font-size, to allow for the character *risers* & *descenders* (eg 'l' & 'g'). Each line of text is considered to have a *baseline*; the vertical distance between 2 adjacent *baselines* is the 'line-height'. *Values' Brief:* • normal: a browser-determined 'reasonable' value for the element's font (W3C: 1.0 to 1.2 times the size) • number: element font-size multiplied by the number; child elements inherit the factor, not the resultant • %-value: as number, except children inherit the resultant (a browser-computed end-value)

Examples:

```
div { line-height: 1.2; font-size: 10pt; }
```

The above specifies, for all `div` elements, a 10 point font with 12 point baseline-to-baseline line-height for all text.contained in the element.

list-style

Syntax: `list-style: <values>`

Description: `list-style` is a shorthand notation for setting the three properties 'list-style-type', 'list-style-image' and 'list-style-position' at the same place in the style sheet.

See also: `display`, `list-style`, `list-style-image`, `list-style-position`, `list-style-type`, `white-space`

Properties:

Group:	Classification Properties
Values:	[disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none] || [inside | outside] || [<url> | none]
Initial values:	(not defined for shorthand properties)
% values:	(n/a)
Applies to:	Elements with `display` value 'list-item'
Inherited?	Yes
Standard:	CSS1☞

Extras:

Advice:	Set *list-style* rules on `ol` & `ul` elements.only. Inheritance will pass the rules down to the `li` elements quite naturally, whilst a combo of cascading order & inheritance can lead to unexpected results that will have you tearing your hair out.

Examples:

```
ul { list-style: url(http://png.com/ellipse.png) disc; }
```

The above specifies, for `ul` elements, to ***display a '●' as a list-item marker if the image is unavailable.***

list-style-image

Syntax: `list-style-image: <values>`

Description: `list-style-image` sets the image that will be used as the list-item marker. When the image is available it will replace the marker set with the `list-style-type` marker.

See also: `display`, `list-style`, `list-style-image`, `list-style-position`, `list-style-type`, `white-space`

Properties:

Group:	Classification Properties
Values:	<url> | none
Initial values:	none
% values:	(n/a)
Applies to:	Elements with `display` value 'list-item'
Inherited?	Yes
Standard:	CSS1☞

Extras:

Extra:	See `list-style-type` for a possible behaviour when the image cannot be loaded.

Examples:

```
ul { list-style-image: url(http://png.com/ellipse.png); }
```

The above specifies, for `ul` elements, to ***display an image as a list-item marker*** for contained `li` elements.

list-style-postion

Syntax: `list-style-postion: <values>`

Description: `list-style-postion` determines how the list-item marker is drawn with regard to the content. See *Extras* below to see what on earth that means.

See also: `display`, `list-style`, `list-style-image`, `list-style-postion`, `list-style-type`, `white-space`

Properties:

Group:	Classification Properties
Values:	inside | outside
Initial values:	outside
% values:	(n/a)
Applies to:	Elements with `display` value 'list-item'
Inherited?	Yes
Standard:	CSS1☞

Extras:

Values:	• *outside:* this is simply lots of words, intended to wrap and then show how this marker is *outside* the text. • *inside:* this is simply lots of words, intended to wrap and then show how this marker is *inside* the text.

Examples:

```
ul { list-style: outside; }
```

The above specifies, for `ul` elements, to ***display the list-item marker outside the text*** (which is also the default).

list-style-type

Syntax: `list-style-type: <values>`

Description: `list-style-type` decides on the appearance of the list-item marker, if `list-style-image` is 'none', or if the image pointed to by the URL cannot be displayed.

See also: `display`, `list-style`, `list-style-image`, `list-style-position`, `list-style-type`, `white-space`

Properties:

Group:	Classification Properties
Values:	disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none
Initial values:	disc
% values:	(n/a)
Applies to:	Elements with `display` value 'list-item'
Inherited?	Yes
Standard:	CSS1

Extras:

Values:	This an attempt to simulate the list-item markers in this PDF: ○ circle ● disc ■ square 1. decimal a. lower-alpha i. lower-roman A. upper-alpha I. upper-roman none

Examples:

```
ol { list-style-type: decimal; }
```

The above specifies, for `ol` elements, to ***display decimal list-item markers*** (1,2,3, etc.) (also the default) for contained `li` elements.

margin

Syntax: `margin: <values>`

Description: `margin` is a shorthand property for setting the margin on all 4 sides of an element at once; margins may be all the same (1 value required) or all different (4 values required). If 2 or 3 values are given, then missing values are taken from the opposite side. See '`border-color`' '*Extras*' for the ordering of values.

See also: `border`, `clear`, `float`, `height`, `margin`, `padding`, `width`

Properties:

Group:	Box Properties
Values:	[<length> | <percentage> | auto]{1,4}
Initial values:	(not defined for shorthand properties)
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

Negative values:	Allowed, but there may be implementation-specific limits.
Extra:	The original docs (1996) on this property spoke of each of the 4 possible values being either a length, a percentage or 'auto'. The latest CSS1 docs☞ (2008) speak only about length, but do not explictly exclude the other two, nor speak on the advisability nor otherwise of mixing value types in a single declaration.
Remember:	The `margin` is transparent.

See also: `margin-top`, `margin-right`, `margin-bottom`, `margin-left`

Examples:

```
body { margin: 2em; }
```

The above specifies, for the `body` element, that it should have a margin of 2em (twice the height of the font-height setting) on all 4 sides.

The `body` element is the outermost visible container for the content. You may therefore have a philosophical discussion on who the body.margin affects... ...and the answer is the `html` element, which is the top-level container.

margin-bottom

Syntax: `margin-bottom: <values>`

Description: `margin-bottom` sets the margin at the bottom side of an element.

See also: `border`, `margin`, `margin-top`, `margin-right`, `margin-bottom`, `margin-left`

Properties:

Group:	Margin Properties
Values:	<length> | <percentage> | auto
Initial values:	0
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Negative values:	Allowed, but there may be implementation-specific limits.
Remember:	The `margin` is transparent.

Examples:

```
dt { margin-bottom: 0.5em; }
```

The above specifies, for all `dt` elements, that they will have 0.5em margin outside the bottom border (1em == same as the font-height).

margin-left

Syntax: `margin-left: <values>`

Description: `margin-left` sets the margin at the left side of an element.

See also: `border`, `margin`, `margin-top`, `margin-right`, `margin-bottom`, `margin-left`

Properties:

Group:	Margin Properties
Values:	<length> | <percentage> | auto
Initial values:	0
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Negative values:	Allowed, but there may be implementation-specific limits.
Remember:	The `margin` is transparent.

Examples:

```
dt { margin-left: 0.5em; }
```

The above specifies, for all `dt` elements, that they will have 0.5em margin outside the left border (1em == same as the font-height).

margin-right

Syntax: `margin-right: <values>`

Description: `margin-right` sets the margin at the bottom side of an element.

See also: `border`, `margin`, `margin-top`, `margin-right`, `margin-bottom`, `margin-left`

Properties:

Group:	Margin Properties
Values:	<length> | <percentage> | auto
Initial values:	0
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Negative values:	Allowed, but there may be implementation-specific limits.
Remember:	The `margin` is transparent.

Examples:

```
dt { margin-right: 0.5em; }
```

The above specifies, for all `dt` elements, that they will have 0.5em margin outside the right border (1em == same as the font-height).

margin-top

Syntax: `margin-top: <values>`

Description: `margin-top` sets the margin at the bottom side of an element.

See also: `border`, `margin`, `margin-top`, `margin-right`, `margin-bottom`, `margin-left`

Properties:

Group:	Margin Properties
Values:	<length> | <percentage> | auto
Initial values:	0
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Negative values:	Allowed, but there may be implementation-specific limits.
Remember:	The `margin` is transparent.

Examples:

```
dt { margin-top: 0.5em; }
```

The above specifies, for all `dt` elements, that they will have 0.5em margin outside the top border (1em == same as the font-height).

padding

Syntax: `padding: <values>`

Description: `padding` is a shorthand property for setting the padding on all 4 sides of an element at once; padding may be all the same (1 value required) or all different (4 values required). If 2 or 3 values are given, then missing values are taken from the opposite side. See '`border-color`' '*Extras*' for the ordering of values.

See also: `border`, `clear`, `float`, `height`, `margin`, `padding`, `width`

Properties:

Group:	Box Properties
Values:	[<length> | <percentage>]{1,4}
Initial values:	(not defined for shorthand properties)
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1☞

Extras:

Negative values:	Cannot be negative.
Extra:	In contrast to `margin`, the latest CSS1 docs☞ (2008) speak about *values*, rather than one of the specific value types, when talking about using 1, 2, 3 or 4 values in the declaration
Remember:	Unlike `margin`, `padding` is visible. It's color, etc. properties are set with the `background` property.

See also: `padding-top`, `padding-right`, `padding-bottom`, `padding-left`

Examples:

```
h4 { padding: 1em 2em; }
```

The above specifies, for all `h4` elements, that they will have 1em padding at top & bottom and 2em padding at left & right (1em == same as the font-height).

padding-bottom

Syntax: `padding-bottom: <values>`

Description: `padding-bottom` sets the padding at the bottom side of an element.

See also: `border`, `padding`, `padding-top`, `padding-right`, `padding-bottom`, `padding-left`

Properties:

Group:	Padding Properties
Values:	<length> | <percentage>
Initial values:	0
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Negative values:	Cannot be negative.
Remember:	Unlike `margin`, `padding` is visible. It's color, etc. properties are set with the `background` property.

Examples:

```
dt { padding-bottom: 0.5em; }
```

The above specifies, for all `dt` elements, that they will have 0.5em padding inside the bottom border (1em == same as the font-height).

padding-left

Syntax: `padding-left: <values>`

Description: `padding-left` sets the padding at the left side of an element.

See also: `border`, `padding`, `padding-top`, `padding-right`, `padding-bottom`, `padding-left`

Properties:

Group:	Padding Properties
Values:	<length> | <percentage>
Initial values:	0
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Negative values:	Cannot be negative.
Remember:	Unlike `margin`, `padding` is visible. It's color, etc. properties are set with the `background` property.

Examples:

```
dt { padding-left: 0.5em; }
```

The above specifies, for all `dt` elements, that they will have 0.5em padding inside the left border (1em == same as the font-height).

padding-right

Syntax: `padding-right: <values>`

Description: `padding-right` sets the padding at the right side of an element.

See also: `border`, `padding`, `padding-top`, `padding-right`, `padding-bottom`, `padding-left`

Properties:

Group:	Padding Properties
Values:	<length> | <percentage>
Initial values:	0
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Negative values:	Cannot be negative.
Remember:	Unlike `margin`, `padding` is visible. It's color, etc. properties are set with the `background` property.

Examples:

```
dt { padding-right: 0.5em; }
```

The above specifies, for all `dt` elements, that they will have 0.5em padding inside the right border (1em == same as the font-height).

padding-top

Syntax: `padding-top: <values>`

Description: `padding-top` sets the padding at the top side of an element.

See also: `border`, `padding`, `padding-top`, `padding-right`, `padding-bottom`, `padding-left`

Properties:

Group:	Padding Properties
Values:	<length> | <percentage>
Initial values:	0
% values:	Refer to width of closest block-level ancestor
Applies to:	All elements
Inherited?	No
Standard:	CSS1

Extras:

Negative values:	Cannot be negative.
Remember:	Unlike `margin`, `padding` is visible. It's color, etc. properties are set with the `background` property.

Examples:

```
dt { padding-top: 0.5em; }
```

The above specifies, for all `dt` elements, that they will have 0.5em padding inside the top border (1em == same as the font-height).

text-align

Syntax: `text-align: <values>`

Description: `text-align` determines how text will be aligned within the element. *'justify':* the actual algorithm used is browser and human language dependent (some are left-to-right whilst others are right-to-left).

See also: `font`, `letter-spacing`, `line-height`, `text-align`, `text-decoration`, `text-indent`, `text-transform`, `vertical-align`, `word-spacing`

Properties:

Group:	Text Properties
Values:	left | right | center | justify
Initial values:	Browser specific
% values:	(n/a)
Applies to:	Block-level elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

justify:	CSS1 Core conformance☞ allows the browser to treat 'justify' as 'left' or 'right' (in that case, depending on whether the element's default writing direction is left-to-right or right-to-left, respectively.)
Extra:	Justification is an external function of the browser which depends upon the internal function of the font. See *"`letter-spacing` | Extras | Extra"* for more information on this.
center	(for UK children): Americans cannot spell "centre".

Examples:

```
div.center { text-align: center; }
```

As 'text-align' inherits, the above specifies that all block-level elements inside the `div` element with "class=center" will be centred. Alignments will be relative to the width of the element and not the canvas.

text-decoration

Syntax: `text-decoration: <values>`

Description: `text-decoration` describes decorations that are added to the text of (any) element. If the element has no text (e.g. the 'img' element) or is empty (e.g. ''), it has no effect. A value of 'blink' causes the text to blink.

See also: `font`, `letter-spacing`, `line-height`, `text-align`, `text-decoration`, `text-indent`, `text-transform`, `vertical-align`, `word-spacing`

Properties:

Group:	Text Properties
Values:	none | [underline || overline || line-through || blink]
Initial values:	none
% values:	(n/a)
Applies to:	All elements
Inherited?	No, but see *Extras* below
Standard:	CSS1

Extras:

Help + inheritance:	The colour(s) required for the text decoration should be derived from the 'color' property value. Although not inherited, elements should match their parent. Example: if an element is underlined, the line should span the child elements. The colour of the underlining will remain the same even if descendant elements have different 'color' values.
blink:	You cannot imagine how much the Netscape-inspired 'blink' attribute came to be hated. The W3C edict is: *"browsers must recognize the keyword 'blink', but are not required to support the blink effect"* (rather British).
Examples:	underline, overline, ~~line-through~~, blink, none (default).

Examples:

```
a:link, a:visited, a:active { text-decoration: underline;}
```

The above specifies, for all `a` elements with a 'href' attribute, that they should be underlined.

text-indent

Syntax: `text-indent: <values>`

Description: `text-indent` sets the indentation of the first formatted line of text within a block-level element. Negative values are allowed but there may be implementation-specific limits.

See also: `font`, `letter-spacing`, `line-height`, `text-align`, `text-decoration`, `text-indent`, `text-transform`, `vertical-align`, `word-spacing`

Properties:

Group:	Text Properties
Values:	<length> | <percentage>
Initial values:	0
% values:	Refers to parent element's width
Applies to:	Block-level elements
Inherited?	Yes
Standard:	CSS1

Extras:

Breaks:	An indentation is NOT inserted in the middle of an element that was broken by another (such as `br`)

Examples:

```
p { text-indent: 3em; }
```

The above specifies, for all p elements, that the first line of text should mmmbe indented by 3em.

text-transform

Syntax: `text-transform: <values>`

Description: `text-transform` applies a classic set of character-transformations to text. The precise character-to-character swaps are language-dependent; Latin-1 & utf-8 should always work, but other encodings may not (browser dependent).

See also: `font`, `letter-spacing`, `line-height`, `text-align`, `text-decoration`, `text-indent`, `text-transform`, `vertical-align`, `word-spacing`

Properties:

Group:	Text Properties
Values:	capitalize | uppercase | lowercase | none
Initial values:	none
% values:	(n/a)
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

Help:	These are the transforms: Capitalize | UPPERCASE | lowercase | none
might not work:	CSS1 Core conformance☞ allows the browser to treat all values as 'none' for certain charsets: 1. if not Latin-1 and 2. if *"languages for which the transformation is* different *from that specified by the case-conversion tables of Unicode"*.

Examples:

```
dt { text-transform: uppercase; }
```

The above specifies, for all `dt` elements, that all text should be UPPER-CASED.

vertical-align

Syntax: `vertical-align: <values>`

Description: `vertical-align` affects the vertical positioning of the element. 'top' & 'bottom' are relative to the formatted line that the element is part of; other keywords are relative to the parent element.

See also: `font`, `letter-spacing`, `line-height`, `text-align`, `text-decoration`, `text-indent`, `text-transform`, `vertical-align`, `word-spacing`

Properties:

Group:	Text Properties
Values:	baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage>
Initial values:	baseline
% values:	Relative to the `line-height` of the element itself (can be <0)
Applies to:	Inline elements
Inherited?	No
Standard:	CSS1

Extras:

Help:	Keyword effects (browsers can differ markedly): baseline | sub | super | top | text-top | middle | bottom | text-bottom
baseline:	align the baseline of the element (or the bottom, if no element baseline) with the baseline of the parent
sub:	subscript the element
super:	superscript the element
top:	align element top with the tallest element on the line
text-top:	align element top with the top of the parent element's font
middle:	align element vertical-midpoint with baseline + ½ the parent x-height.
bottom:	align element bottom with the lowest element on the line
text-bottom:	align element bottom with the bottom of the parent element's font

Examples:

```
img { vertical-align: 50%; }
```

The above specifies, for all `img` elements, a mid-point vertical alignment.

width

Syntax: `width: <values>`

Description: `width` is suggested to be used principally with replaced elements such as `img`. The element will be scaled if necessary (see *Extras* below).

See also: `border`, `clear`, `float`, `height`, `margin`, `padding`, `width`

Properties:

Group:	Box Properties
Values:	<length> | auto
Initial values:	auto
% values:	(n/a)
Applies to:	Block-level and replaced elements
Inherited?	No
Standard:	CSS1☞

Extras:

General:	Both `height` & `width` are considered together to find a replaced element's eventual dimensions: • `height` = 'auto'; `width` = 'auto': set to intrinsic dimensions of element • `height` = 'auto'; `width` = <length>: `width` enforced by scaling, aspect ratio preserved • `height` = <length>; `width` = 'auto': `height` enforced by scaling, aspect ratio preserved • `height` = <length>; `width` = <length>: enforced by scaling, aspect ratio ignored
CSS1 Core☞:	Browsers may ignore `width` (treat it as 'auto') if the element is not a replaced element.

Examples:

```
img.icon { width: 100px }
```

The above specifies, for all `img` elements declared with a class of 'icon', that they will be displayed at a width of 100 pixels (note how that could be an issue on a very high-density display device).

white-space

Syntax: `white-space: <values>`

Description: `white-space` decides how whitespace inside the element will be handled: as per 'normal' (whitespace is collapsed); as per 'pre' (like the HTML `pre` element) or as per 'nowrap' (wrapping occurs only via HTML `br` elements).

See also: `display`, `list-style`, `list-style-image`, `list-style-position`, `list-style-type`, `white-space`

Properties:

Group:	Classification Properties
Values:	normal | pre | nowrap
Initial values:	normal (though also refer to HTML element default)
% values:	(n/a)
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

whitespace:	The following are all called 'whitespace' by HTML4: • *space ('sp')*: ascii: 0x0020; decimal: 32; entity: . • *tab ('tb')*: ascii: 0x0009; decimal: 9; entity: . • *form-feed ('ff')*: ascii: 0x000c; decimal: 12; entity: . • *zero width space*: ascii: 0x200b; decimal: 8,203; entity: . • *line-feed ('lf')*: ascii: 0x000a; decimal: 10; entity:
. • *carriage return ('cr')*: ascii: 0x000d; decimal: 13; entity: &#x0013;. *Note:* other languages & encodings contain other whitespace characters; the above are the only ones recognised by HTML4.
'normal':	Consecutive display-whitespace collapsed to a single space.
'pre':	Fixed-pitch font; no whitespace collapse; respect tb, ff, lf + cr.
'nowrap':	Turns OFF (normal) wrap-at-whitespace; also response to ' '
CSS1 Core☞:	Browsers may ignore `white-space` and use only defaults.

Examples:

```
pre { white-space: pre; }
```

The above specifies, for `pre` elements, to ***display preformatted***. (which is also the default for text within `pre` elements).

word-spacing

Syntax: `word-spacing: <values>`

Description: `word-spacing` sets an addition to the default space between words. Negative values are allowed but can hit implementation-limits. Do not use this & justification together.

See also: `font`, `letter-spacing`, `line-height`, `text-align`, `text-decoration`, `text-indent`, `text-transform`, `vertical-align`, `word-spacing`

Properties:

Group:	Text Properties
Values:	normal | <length>
Initial values:	normal
% values:	(n/a)
Applies to:	All elements
Inherited?	Yes
Standard:	CSS1☞

Extras:

Extra:	Word-spacing is an external function of the browser which depends upon the internal function of the font. See *"`letter-spacing` | Extras | Extra"* for more information on this.
normal:	CSS1 Core conformance☞ allows the browser to interpret any value of 'word-spacing' as 'normal'.

Examples:

```
h1 { word-spacing: 1em; }
```

The above specifies, for all `h1` elements, that all contained text should have the word-spacing increased by 1em (the horizontal-space occupied by the letter 'm') (w h i c h i s a *l o t* !)

Miscellany

Ever since the day, as a young adolescent that loved cartoons, and I read a small cartoon in my father's *Daily Mirror*, which showed a chap in an office who had just opened one of the lower drawers in a filing cabinet and was looking down at it in shock, with the caption:

"Good heavens, it's Miss Cellaneous"

...I've loved the word *"Miscellany"*.

What follows now is a medley (*dictionary:* "a heterogenous mixture") of different css items, though strongly biased towards text display, simply because I also love typography & fonts.

Quirks Mode

Most of the time you are going to want to avoid this within your webpages. To do so is fairly simple:

To avoid: Include a DOCTYPE declaration above every HTML document.

And yes: it really is as simple as that (although also read the following links). As to how & why 'Quirks mode' originated, and what it means in practice, that is a *very* long & tortuous tale which I'm not going to try to reproduce in full here – it will just be the headlines.

Read More:

- http://www.quirksmode.org/css/quirksmode.html☞ (scroll halfway down to find some useful test pages)
- http://www.cs.tut.fi/~jkorpela/quirks-mode.html☞

What it causes:

In a sentence, it causes MSIE (Internet Explorer) to behave the same in it's page-layout as Internet Explorer 5.5, and most other browsers to behave the same as Netscape 4 (released in 1997), each of which had horribly broken CSS support; moreover, broken in different ways to each other).

Why?:

At the point of release of their v4 web-browsers in 1997, Netscape & Microsoft were the providers of the two main methods used by the world to access the Internet: it was either *"Netscape 4"* or *"Internet Explorer 4"*. Each company was also engaged in a war to gain control of the Desktop and, in particular, of the Browser. Their focus was upon each other rather than upon their users. Their implementation of the newly-available CSS was each buggy in the extreme (neither conformed 100% to the W3C standard). Web authors had to choose to design for one browser or the other.

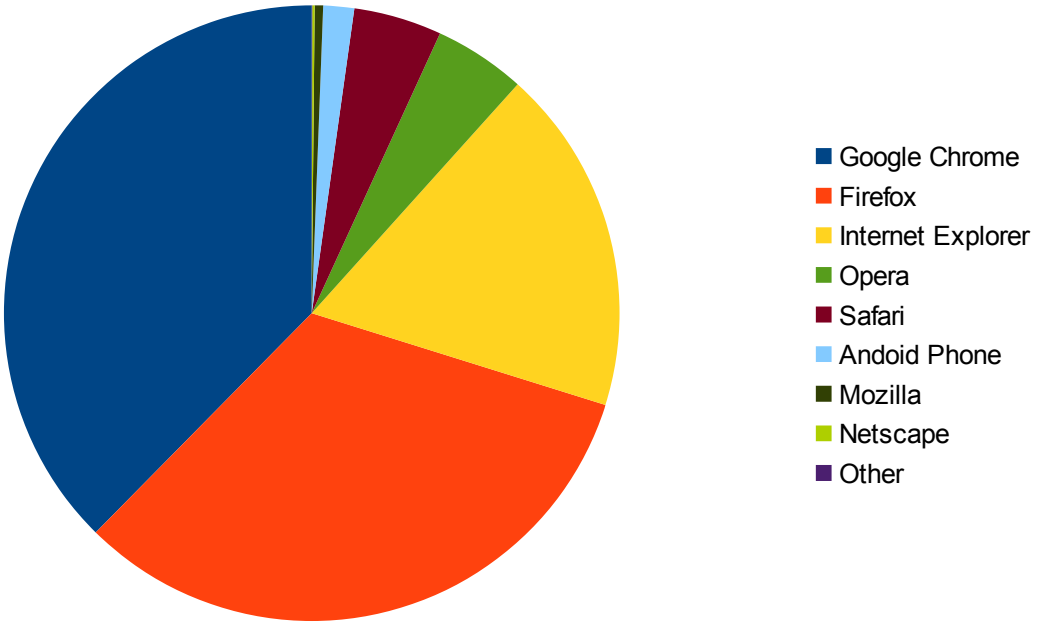
Later, conformance to web-standards became an issue that all browser manufacturers agreed that they had to pay attention to. Yet, because of the earlier history, the many millions of web-pages on the Internet either had no CSS, or had CSS that was largely non-standard. 'Quirks mode' was an attempt by those manufacturers to try to support those old pages whilst providing standards-compliance for new pages.

One curiosity of the DOCTYPE declaration is that it does not appear within any of the formal HTML standards, yet is mentioned within the comments of all of the DTDs. Microsoft started switching mode with MSIE 5 on the Mac, and then with MSIE 6 on all platforms, and all other browsers followed suit. If the top of the HTML contains a DOCTYPE then the rendering mode is 'Strict' (*"Standards compliance mode"*), else it is Quirky. Meanwhile, MSIE 5.5 Windows + Netscape 4 (and earlier) are in permanent *Quirks* mode.

The picture today:

With the earlier mention of the "*Browser Wars*" (it flared up near the end of the last millenium, and near the start had Netscape & Microsoft essentially sharing the browser market 50:50), I thought that you may be interested to see a snapshot of the web browsers used by humans today (drawn from my own website☞, so not necessarily typical across the web):

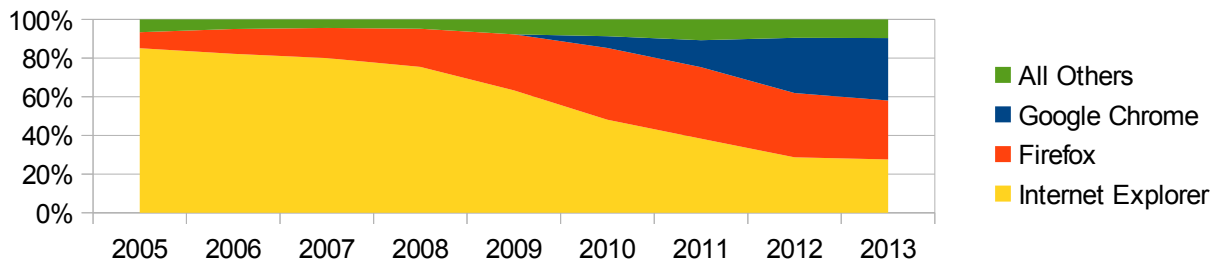
Browser Use for Modem-Help, UK; October 2013
Grand total = 100%
(individual totals are for all versions of that browser)



1. Google Chrome: 37.62%
2. Firefox: 32.56%
3. Internet Explorer 18.15%
4. Opera: 4.83%
5. Safari 4.63%
6. Android Phone 1.61%
7. Mozilla 0.44%
8. Netscape 0.13%
9. Other 0.03%

Microsoft Internet Explorer:

- January 2013: 27.46%
- January 2012: 30.78%
- January 2011: 37.10%
- January 2010: 48.10%



Column Layouts, css1

The bottom line for this is, in my personal view, not to bother. I spent 7 days in 2003, 18 hours a day, trying my best to produce a css1-only file; I discovered in the process that css1 is not fit for purpose – a deeply frustrating experience, particularly as I had fully committed to the principle of the separation of style from content that is at the heart of css. However, I had a new site to construct & zero income until it appeared...

Using a top-level table made page-layout a doddle to accomplish, and it was consistent across all browsers. Trying to do the same using pure css1 required non-standard hacks; worse, those hacks would need updating with each new version release from one or another Browser. It was a Gordian Knot of epic & growing proportions, and using tables was the sword that solved them all in one fell (and foul) swoop.

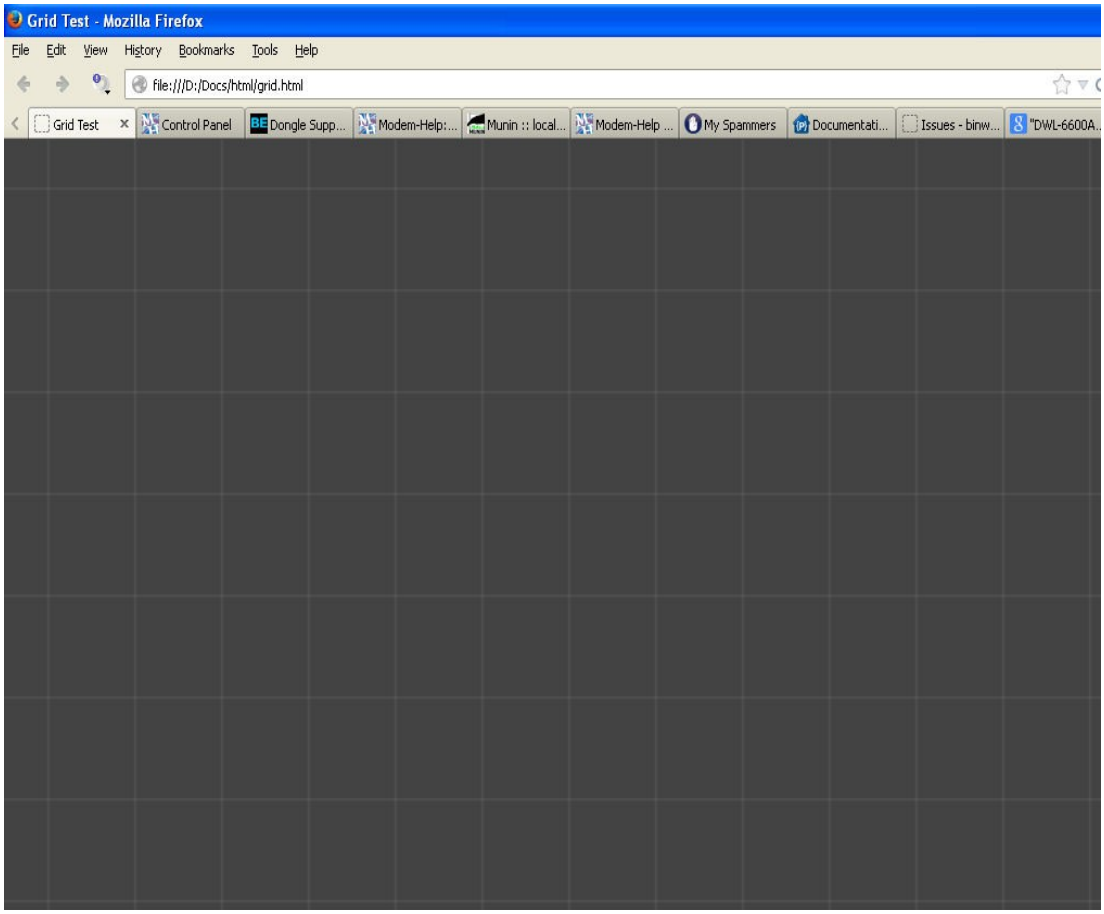
You will understand from the previous paragraphs, therefore, that I consider the following links of historical interest only (I researched them fresh in November 2013, and found the same links as in 2003):

- Little Boxes☞
  A plethora of different layouts from Owen Briggs (marvellous design)
- Blue Robot Layouts☞
- glish.com☞
  Warning! Broken .asp pages!

Text grids using only css

A background grid pattern on the page is mighty useful for testing purposes whilst designing your pages, and especially for text layout. The usual way of doing this pre-CSS was to use specially-made *.gif* files attached to the `body` element using the 'background' property, but that is just *so* last millenium.

The picture of the webpage below uses CSS3 properties to achieve a pure-CSS grid (the HTML, which includes embedded CSS, is below it). Any modern browser should show it fine (the pic is of Firefox 24.0). Other pages on text within this miscellany use the identical CSS.



```
<!DOCTYPE html>
<html>
  <head>
  <title>Grid Test</title>
  <style type="text/css">
    html {
      height: 100%;
    }
    body {
      margin: 0;
      padding: 0;
      height: 100%;
      background-color: #434343;
/*    background-image:linear-gradient(#434343, #282828);*/
      color: white;
      font: 100pt/120pt serif;
    }
    #content{
      background-color: transparent;
      background-image: linear-gradient(0deg, transparent 24%, rgba(255,
255, 255, .05) 25%, rgba(255, 255, 255, .05) 26%, transparent 27%,
transparent 74%, rgba(255, 255, 255, .05) 75%, rgba(255, 255, 255, .05)
76%, transparent 77%, transparent), linear-gradient(90deg, transparent
24%, rgba(255, 255, 255, .05) 25%, rgba(255, 255, 255, .05) 26%,
transparent 27%, transparent 74%, rgba(255, 255, 255, .05) 75%, rgba(255,
255, 255, .05) 76%, transparent 77%, transparent);
      margin: 0;
      padding: 0;
      height: 100%;
      background-size: 120pt 120pt;
      padding-top: 55pt;
    }
    </style>
  </head>
  <body>
    <div id=content></div>
  </body>
</html>
```

There are two keys to getting text to line-up upon the grid, so that the text baseline sits upon a gridline:

1. `background-size` (within '#content')
   Both x & y need to be set to the same size as the font line-height
2. `padding-top` (within '#content')
   This needs a small calculation.

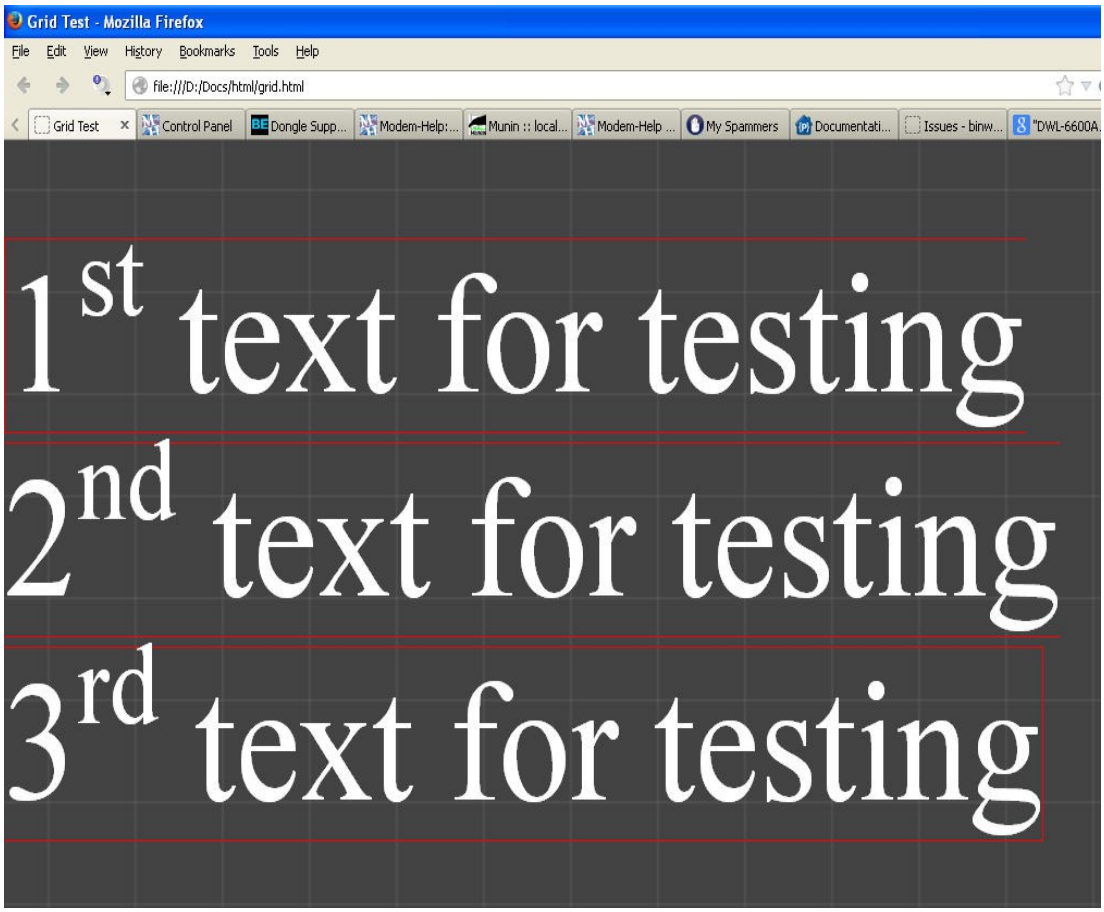
   To help you understand it, first recall that css places the text centrally within the line-box, and that effectively means that half the leading is added *above* & half added *below* the text. Next, recall that the leading is the difference between the line-height & the font height (which is the font-size). Finally, recall that when a font is described as "100pt/120pt", that means "`font-size/line-height`", (there are 72 points in an inch (2.54cm)).

   The effect of top padding is to push the content (but not the background) down. The amount required here is:

   = ½ * (font-size + (½ * leading))
   = ½ * (100pt + (½ * 20pt))
   = ½ * 110pt
   = 55pt

Accurate Superscript & Subscripts

Using either the HTML `sup` and/or `sub` will make a dog's dinner of your carefully-crafted text pages (examples later). Unfortunately, using CSS "vertical-align=super" and/or "vertical-align=sub" on their own will do exactly the same. In this instance, both the W3C *and* the browser designers have conspired together to achieve that. Here's how to do it right:



the CSS:

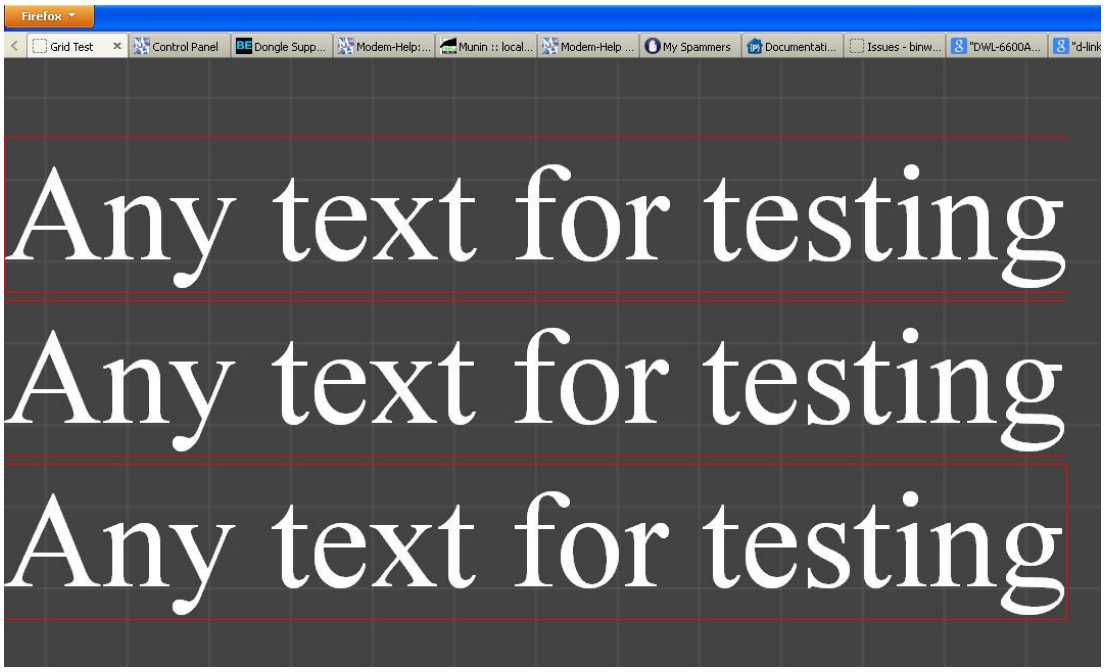
```
sup {
  font-size: 68%;
  line-height: 68%
}
```

the HTML:

```
<body><div id=content><span style="border: thin solid red;">
  1<sup>st</sup> text for testing<br />
  2<sup>nd</sup> text for testing<br />
  3<sup>rd</sup> text for testing
</span></div></body>
```

The above uses a css text-grid. It all looks perfectly normal, until you see the havoc that will ensue otherwise (and you will then also understand the reason for the red border box) (next pages):

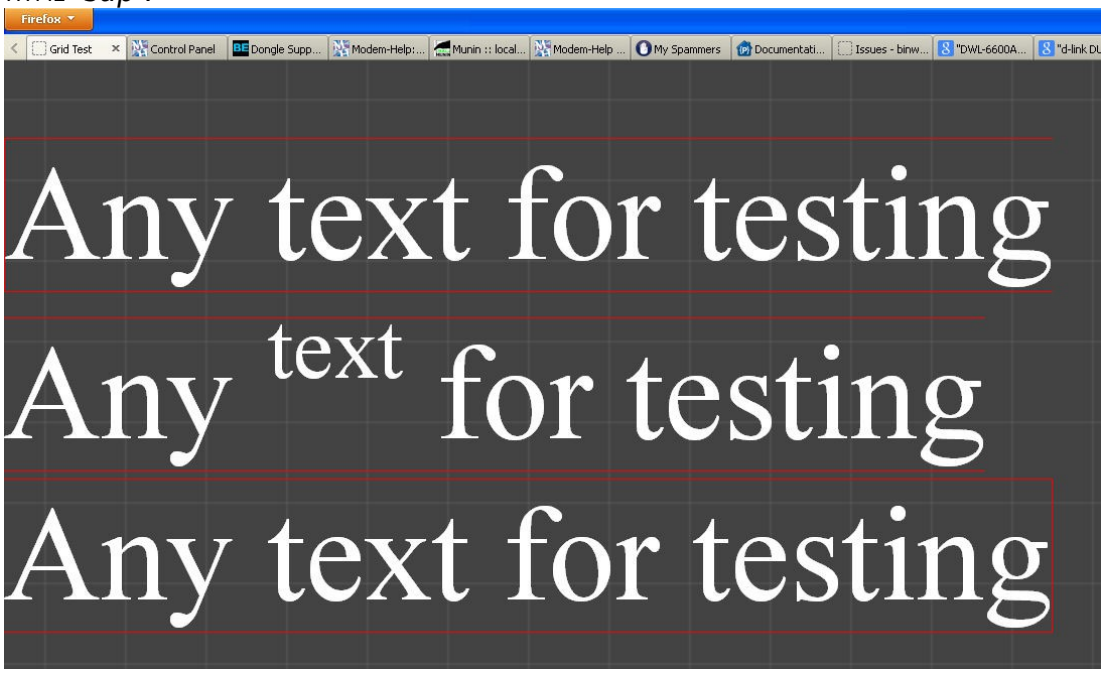
Here is some text using a css text-grid. The earlier css for `sup` is now missing, so this is default browser formatting from now on (Firefox 24.0):



the HTML:

```
<body><div id=content><span style="border: thin solid red;">
  Any text for testing<br />
  Any text for testing<br />
  Any text for testing
</span></div></body>
```

The effect of HTML 'sup':



the HTML:

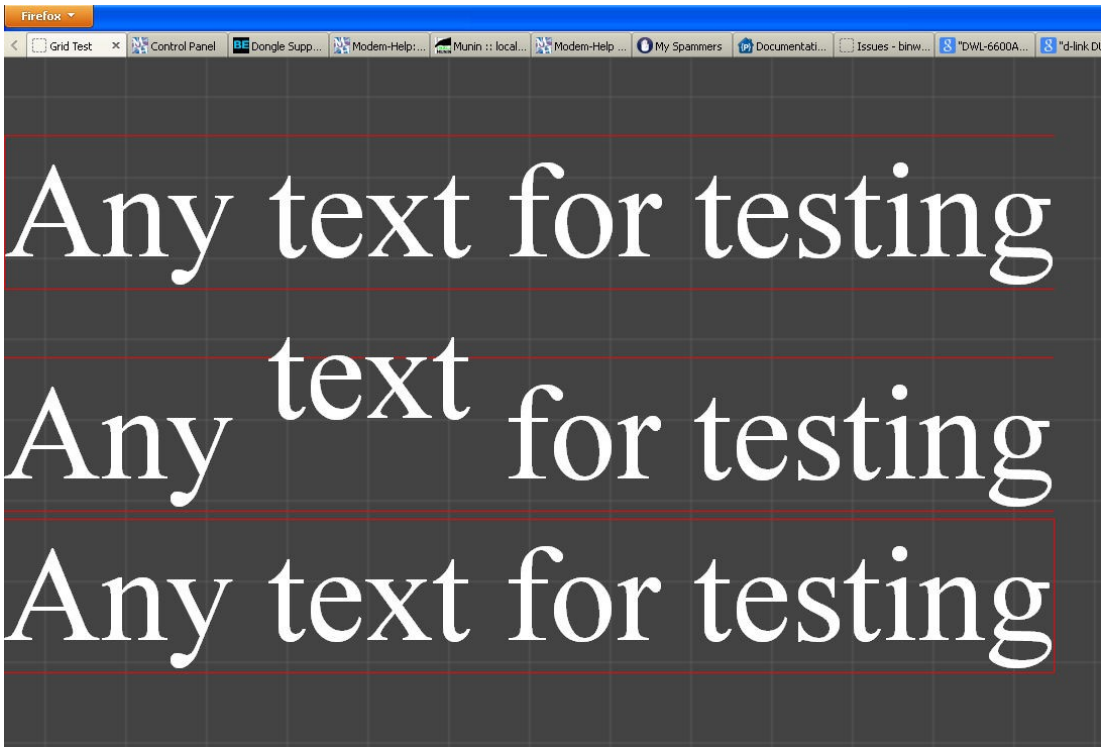
```
<body><div id=content><span style="border: thin solid red;">
  Any text for testing<br />
  Any <sup>text</sup> for testing<br />
  Any text for testing
</span></div></body>
```

If you look closely, you will see that the browser has shrunk 'text' & raised it into superscript position. Unfortunately, in the process it has pushed down the rest of the text-line, so that neither it nor the last line of text sit upon their original baselines any more (only the top line is unaffected).

Imagine if you had 2 adjacent columns of text; following the superscript, the lines of text would be out of kilter, and the whole page would look most unprofessional.

The effect of css 'vertical-align: super':

You would hope that css formatting would fix the above; if anything, it is worse (and also shows that the Firefox designers may be ignoring the W3C guidelines for line-box formatting):



the HTML:

```
<body><div id=content><span style="border: thin solid red;">
  Any text for testing<br />
  Any <span style="vertical-align: super;">text</span> for testing<br />
  Any text for testing
</span></div></body>
```

Look closely, and you will see that the middle 'text' is upon it's original baseline, whilst the rest of the text-line has been dropped until 'text' is in superscript position with regard to the rest of the text-line. *Gordon Bennett! Wot a brain-dead way of arranging matters!* Well, whatever their reasons for doing things that way, they have ensured that the entire page will just look silly.

Google Guidelines for HTML & CSS

The words below come from a document☞ obtained on 6 October 2013, and saved as plain-text. It is guidelines issued by Google to it's minions on useage of HTML + CSS within collaborative projects. I thought that it had some useful things to say, and that folks reading this PDF may also find it useful.

Revision 2.23:

This style guide contains many details that are initially hidden from view. They are marked by the triangle icon, which you see here on your left. Click it now. You should see "Hooray" appear below.

Hooray! Now you know you can expand points to get more details. Alternatively, there's a "toggle all" at the top of this document.

Purpose:

This document defines formatting and style rules for HTML and CSS. It aims at improving collaboration, code quality, and enabling supporting infrastructure. It applies to raw, working files that use HTML and CSS, including GSS files. Tools are free to obfuscate, minify, and compile as long as the general code quality is maintained.

Omit the protocol from embedded resources:

Omit the protocol portion (|http:|, |https:|) from URLs pointing to images and other media files, style sheets, and scripts unless the respective files are not available over both protocols.

Omitting the protocol—which makes the URL relative—prevents mixed content issues and results in minor file size savings.

```
<!-- Not recommended --> <script
src="http://www.google.com/js/gweb/analytics/autotrack.js"></script>

<!-- Recommended --> <script
src="//www.google.com/js/gweb/analytics/autotrack.js"></script>

/* Not recommended */ .example { background:
url(http://www.google.com/images/example); }

/* Recommended */ .example { background:
url(//www.google.com/images/example); }
```

Indent by 2 spaces at a time:

Don't use tabs or mix tabs and spaces for indentation.

Use only lowercase:

All code has to be lowercase: This applies to HTML element names, attributes, attribute values (unless |text/CDATA|), CSS selectors, properties, and property values (with the exception of strings).

```
<!-- Not recommended --> <A HREF="/">Home</A>

<!-- Recommended --> <img src="google.png" alt="Google">

/* Not recommended */ color: #E5E5E5;

/* Recommended */ color: #e5e5e5;
```

Remove trailing white spaces:

Trailing white spaces are unnecessary and can complicate diffs.

```
<!-- Not recommended --> <p>What?_

<!-- Recommended --> <p>Yes please.
```

Use UTF-8 (no BOM):

Make sure your editor uses UTF-8 as character encoding, without a byte order mark.

Specify the encoding in HTML templates and documents via |<meta charset="utf-8">|. Do not specify the encoding of style sheets as these assume UTF-8.

(More on encodings and when and how to specify them can be found in Handling character encodings in HTML and CSS <http://www.w3.org/International/tutorials/tutorial-char-enc/☞>.)

Explain code as needed, where possible:

Use comments to explain code: What does it cover, what purpose does it serve, why is respective solution used or preferred?

(This item is optional as it is not deemed a realistic expectation to always demand fully documented code. Mileage may vary heavily for HTML and CSS code and depends on the project's complexity.)

Mark todos and action items with |TODO|:

Highlight todos by using the keyword |TODO| only, not other common formats like |@@|.

Append a contact (username or mailing list) in parentheses as with the format |TODO(contact)|.

Append action items after a colon as in |TODO: action item|.

```
{# TODO(john.doe): revisit centering #} <center>Test</center>
<!-- TODO: remove optional tags --> <ul> <li>Apples</li> <li>Oranges</li>
</ul>
```

Use HTML5:

HTML5 (HTML syntax) is preferred for all HTML documents: |<!DOCTYPE html>|.

(It's recommended to use HTML, as |text/html|. Do not use XHTML. XHTML, as |application/xhtml+xml| <http://hixie.ch/advocacy/xhtml☞>, lacks both browser and infrastructure support and offers less room for optimization than HTML.)

Although fine with HTML, do not close void elements, i.e. write |
|, not |
|.

Use valid HTML where possible:

Use valid HTML code unless that is not possible due to otherwise unattainable performance goals regarding file size.

Use tools such as the W3C HTML validator <http://validator.w3.org/nu/☞> to test.

Using valid HTML is a measurable baseline quality attribute that contributes to learning about technical requirements and constraints, and that ensures proper HTML usage.

```
<!-- Not recommended --> <title>Test</title> <article>This is only a test.

<!-- Recommended --> <!DOCTYPE html> <meta charset="utf-8">
<title>Test</title> <article>This is only a test.</article>
```

Use HTML according to its purpose:

Use elements (sometimes incorrectly called “tags”) for what they have been created for. For example, use heading elements for headings, |p| elements for paragraphs, |a| elements for anchors, etc.

Using HTML according to its purpose is important for accessibility, reuse, and code efficiency reasons.

```
<!-- Not recommended --> <div onclick="goToRecommendations();">All
recommendations</div>

<!-- Recommended --> <a href="recommendations/">All recommendations</a>
```

Provide alternative contents for multimedia:

For multimedia, such as images, videos, animated objects via |canvas|, make sure to offer alternative access. For images that means use of meaningful alternative text (|alt|) and for video and audio transcripts and captions, if available.

Providing alternative contents is important for accessibility reasons: A blind user has few cues to tell what an image is about without |@alt|, and other users may have no way of understanding what video or audio contents are about either.

(For images whose |alt| attributes would introduce redundancy, and for images whose purpose is purely decorative which you cannot immediately use CSS for, use no alternative text, as in |alt=""|.)

```
<!-- Not recommended --> <img src="spreadsheet.png">

<!-- Recommended --> <img src="spreadsheet.png" alt="Spreadsheet
screenshot.">
```

Separate structure from presentation from behavior:

Strictly keep structure (markup), presentation (styling), and behavior (scripting) apart, and try to keep the interaction between the three to an absolute minimum.

That is, make sure documents and templates contain only HTML and HTML that is solely serving structural purposes. Move everything presentational into style sheets, and everything behavioral into scripts.

In addition, keep the contact area as small as possible by linking as few style sheets and scripts as possible from documents and templates.

Separating structure from presentation from behavior is important for maintenance reasons. It is always more expensive to change HTML documents and templates than it is to update style sheets and scripts.

```
<!-- Not recommended --> <!DOCTYPE html> <title>HTML sucks</title> <link
rel="stylesheet" href="base.css" media="screen"> <link rel="stylesheet"
href="grid.css" media="screen"> <link rel="stylesheet" href="print.css"
media="print"> <h1 style="font-size: 1em;">HTML sucks</h1> <p>I've read
about this on a few sites but now I'm sure: <u>HTML is stupid!!1</u>
<center>I can't believe there's no way to control the styling of my website
without doing everything all over again!</center>

<!-- Recommended --> <!DOCTYPE html> <title>My first CSS-only
redesign</title> <link rel="stylesheet" href="default.css"> <h1>My first
CSS-only redesign</h1> <p>I've read about this on a few sites but today I'm
actually doing it: separating concerns and avoiding anything in the HTML of
my website that is presentational. <p>It's awesome!
```

Do not use entity references:

There is no need to use entity references like |—|, |”|, or |☺|, assuming the same encoding (UTF-8) is used for files and editors as well as among teams.

The only exceptions apply to characters with special meaning in HTML (like |<| and |&|) as well as control or "invisible" characters (like no-break spaces).

```
<!-- Not recommended --> The currency symbol for the Euro is
&ldquo;&eur;&rdquo;.

<!-- Recommended --> The currency symbol for the Euro is "€".
```

Omit optional tags (optional):

For file size optimization and scannability purposes, consider omitting optional tags. The HTML5 specification <http://www.whatwg.org/specs....html#syntax-tag-omission☞> defines what tags can be omitted.

(This approach may require a grace period to be established as a wider guideline as it's significantly different from what web developers are typically taught. For consistency and simplicity reasons it's best served omitting all optional tags, not just a selection.)

```
<!-- Not recommended --> <!DOCTYPE html> <html> <head> <title>Spending
money, spending bytes</title> </head> <body> <p>Sic.</p> </body> </html>
```

```
<!-- Recommended --> <!DOCTYPE html> <title>Saving money, saving
bytes</title> <p>Qed.
```

Omit |type| attributes for style sheets and scripts:

Do not use |type| attributes for style sheets (unless not using CSS) and scripts (unless not using JavaScript).

Specifying |type| attributes in these contexts is not necessary as HTML5 implies |text/css| <http://www.whatwg.org/specs...html#attr-style-type☞> and |text/javascript| <http://www.whatwg.org/sp...html#attr-script-type☞> as defaults. This can be safely done even for older browsers.

```
<!-- Not recommended --> <link rel="stylesheet"
href="//www.google.com/css/maia.css" type="text/css">

<!-- Recommended --> <link rel="stylesheet"
href="//www.google.com/css/maia.css">

<!-- Not recommended --> <script
src="//www.google.com/js/gweb/analytics/autotrack.js"
type="text/javascript"></script>

<!-- Recommended --> <script
src="//www.google.com/js/gweb/analytics/autotrack.js"></script>
```

Use a new line for every block, list, or table element, and indent every such child element:

Independent of the styling of an element (as CSS allows elements to assume a different role per |display| property), put every block, list, or table element on a new line.

Also, indent them if they are child elements of a block, list, or table element.

(If you run into issues around whitespace between list items it's acceptable to put all |li| elements in one line. A linter is encouraged to throw a warning instead of an error.)

```
<blockquote>
  <p><em>Space</em>, the final frontier.</p>
</blockquote>
<ul>
  <li>Moe <li>Larry <li>Curly
</ul>
<table>
  <thead>
    <tr>
      <th scope="col">Income
      <th scope="col">Taxes
  <tbody>
    <tr>
      <td>$ 5.00
      <td>$4.50
</table>
```

When quoting attributes values, use double quotation marks:

Use double (|""|) rather than single quotation marks (|''|) around attribute values.

```
<!-- Not recommended --> <a class='maia-button maia-button-secondary'>Sign
in</a>

<!-- Recommended --> <a class="maia-button maia-button-secondary">Sign
in</a>
```

Use valid CSS where possible:

Unless dealing with CSS validator bugs or requiring proprietary syntax, use valid CSS code.

Use tools such as the W3C CSS validator <http://jigsaw.w3.org/css-validator/☞> to test.

Using valid CSS is a measurable baseline quality attribute that allows to spot CSS code that may not have any effect and can be removed, and that ensures proper CSS usage.

Use meaningful or generic ID and class names:

Instead of presentational or cryptic names, always use ID and class names that reflect the purpose of the element in question, or that are otherwise generic.

Names that are specific and reflect the purpose of the element should be preferred as these are most understandable and the least likely to change.

Generic names are simply a fallback for elements that have no particular or no meaning different from their siblings. They are typically needed as "helpers."

Using functional or generic names reduces the probability of unnecessary document or template changes.

```
/* Not recommended: meaningless */ #yee-1901 {}

/* Not recommended: presentational */ .button-green {} .clear {}

/* Recommended: specific */ #gallery {} #login {} .video {}

/* Recommended: generic */ .aux {} .alt {}
```

Use ID and class names that are as short as possible but as long as necessary:

Try to convey what an ID or class is about while being as brief as possible.

Using ID and class names this way contributes to acceptable levels of understandability and code efficiency.

```
/* Not recommended */ #navigation {} .atr {} /*

Recommended */ #nav {} .author {}
```

Avoid qualifying ID and class names with type selectors:

Unless necessary (for example with helper classes), do not use element names in conjunction with IDs or classes.

Avoiding unnecessary ancestor selectors is useful for performance reasons <http://www.stevesouders.com/blog/...simplifying-css-selectors/☞>.

```
/* Not recommended */ ul#example {} div.error {}

/* Recommended */ #example {} .error {}
```

Use shorthand properties where possible:

CSS offers a variety of shorthand <http://www.w3.org/TR/CSS21/about.html#shorthand☞> properties (like |font|) that should be used whenever possible, even in cases where only one value is explicitly set.

Using shorthand properties is useful for code efficiency and understandability.

```
/* Not recommended */ border-top-style: none; font-family: palatino,
georgia, serif; font-size: 100%; line-height: 1.6; padding-bottom: 2em;
padding-left: 1em; padding-right: 1em; padding-top: 0;

/* Recommended */ border-top: 0; font: 100%/1.6 palatino, georgia, serif;
padding: 0 1em 2em;
```

Omit unit specification after "0" values:

Do not use units after |0| values unless they are required.

```
margin: 0; padding: 0;
```

Omit leading "0"s in values:

Do not use put |0|s in front of values or lengths between -1 and 1.

```
font-size: .8em;
```

Use 3 character hexadecimal notation where possible:

For color values that permit it, 3 character hexadecimal notation is shorter and more succinct.

```
/* Not recommended */ color: #eebbcc; /*

Recommended */ color: #ebc;
```

Prefix selectors with an application-specific prefix (optional):

In large projects as well as for code that gets embedded in other projects or on external sites use prefixes (as namespaces) for ID and class names. Use short, unique identifiers followed by a dash.

Using namespaces helps preventing naming conflicts and can make maintenance easier, for example in search and replace operations.

```
.adw-help {} /* AdWords */
#maia-note {} /* Maia */
```

Separate words in ID and class names by a hyphen:

Do not concatenate words and abbreviations in selectors by any characters (including none at all) other than hyphens, in order to improve understanding and scannability.

```
/* Not recommended: does not separate the words "demo" and "image" */
.demoimage {}
/* Not recommended: uses underscore instead of hyphen */ .error_status {}

/* Recommended */ #video-id {} .ads-sample {}
```

Avoid user agent detection as well as CSS "hacks"—try a different approach first:

It's tempting to address styling differences over user agent detection or special CSS filters, workarounds, and hacks. Both approaches should be considered last resort in order to achieve and maintain an efficient and manageable code base. Put another way, giving detection and hacks a free pass will hurt projects in the long run as projects tend to take the way of least resistance. That is, allowing and making it easy to use detection and hacks means using detection and hacks more frequently—and more frequently is too frequently.

Alphabetize declarations:

Put declarations in alphabetical order in order to achieve consistent code in a way that is easy to remember and maintain.

Ignore vendor-specific prefixes for sorting purposes. However, multiple vendor-specific prefixes for a certain CSS property should be kept sorted (e.g. -moz prefix comes before -webkit).

```
background: fuchsia;
border: 1px solid;
-moz-border-radius: 4px;
-webkit-border-radius: 4px;

border-radius: 4px;
color: black;
text-align: center;
text-indent: 2em;
```

Indent all block content:

Indent all block content <http://www.w3.org/TR/CSS21/syndata.html#block☞>, that is rules within rules as well as declarations, so to reflect hierarchy and improve understanding.

```
@media screen, projection {
  html {
    background: #fff;
    color: #444;
  }
}
```

Use a semicolon after every declaration:

End every declaration with a semicolon for consistency and extensibility reasons.

```
/* Not recommended */ .test { display: block; height: 100px }

/* Recommended */ .test { display: block; height: 100px; }
```

Use a space after a property name's colon:

Always use a single space between property and value (but no space between property and colon) for consistency reasons.

```
/* Not recommended */ h3 { font-weight:bold; }

/* Recommended */ h3 {font-weight: bold; }
```

Use a space between the last selector and the declaration block:

Always use a single space between the last selector and the opening brace that begins the declaration block <http://www.w3.org/TR/CSS21/syndata.html#rule-sets☞>.

The opening brace should be on the same line as the last selector in a given rule.

```
/* Not recommended: missing space */
       #video{
         margin-top: 1em;
       }
/* Not recommended: unnecessary line break */
       #video
       {
         margin-top: 1em;
       } /*

Recommended */
       #video {
         margin-top: 1em;
       }
```

Separate selectors and declarations by new lines:

Always start a new line for each selector and declaration.

```
/* Not recommended */
       a:focus, a:active {
         position: relative; top: 1px;
       }

/* Recommended */
        h1, h2, h3 {
         font-weight: normal;
         line-height: 1.2;
       }
```

Separate rules by new lines:

Always put a blank line (two line breaks) between rules.

```
html {
  background: #fff;
}

body {
  margin: auto;
  width: 50%;
}
```

Use single quotation marks for attribute selectors and property values:

Use single (|'|) rather than double (|"|) quotation marks for attribute selectors or property values.

Do not use quotation marks in URI values (|url()|):

Exception: If you do need to use the |@charset| rule, use double quotation marks—single quotation marks are not permitted <http://www.w3.org/TR/CSS21/syndata.html#charset☞>.

```
/* Not recommended */
      @import url("//www.google.com/css/maia.css");
      html {
        font-family: "open sans", arial, sans-serif;
      }

 /* Recommended */
      @import url(//www.google.com/css/maia.css);
      html {
        font-family: 'open sans', arial, sans-serif;
      }
```

Group sections by a section comment (optional):

If possible, group style sheet sections together by using comments. Separate sections with new lines.

```
/* Header */
#adw-header {}
/* Footer */
#adw-footer {}
/* Gallery */
.adw-gallery {}
```

/Be consistent./

If you're editing code, take a few minutes to look at the code around you and determine its style. If they use spaces around all their arithmetic operators, you should too. If their comments have little boxes of hash marks around them, make your comments have little boxes of hash marks around them too.

The point of having style guidelines is to have a common vocabulary of coding so people can concentrate on what you're saying rather than on how you're saying it. We present global style rules here so people know the vocabulary, but local style is also important. If code you add to a file looks drastically different from the existing code around it, it throws readers out of their rhythm when they go to read it. Avoid this.

Revision 2.23