

A Date With HTML

Fanen Ahua

```

                                <ht                                ml>
                                <head>                                <meta
                                name =                                "keywor
                                ds" c                                ontent
                                = "                                learn
                                html,                                author,fa
ll   nen a.,b                                ooks"> <!-- ll
ll   --> <styl                                e>b                                ody                                (backgrou ll
ll   nd-col0r:rgb(128,1 9 0 ,200);color:gre ll
ll   en;fon t-family:ve r d ana,times new ro ll
ll   an,ta homa ----- ,comic ll
ll   sans,s e ----- rif; m ll
ll   arg in-                                ===== left: 1 0 ll
ll   pt;)< /s ty                                ===== le>< /head> ll
ll   <body><h1>Webdes                                ===== ign: a date wit ll
ll   h html</h1><p>thet                                ===== extofthebookfol ll
ll   lowsafterwardsan                                ===== lookatthepictu ll
ll   llllllllllll reand                                ===== nott llllllllllll
ll   llll het                                ===== ext.t llll
ll   llll was                                ===== myw llll
ll   llll ork                                ===== tnx llll

```

Table of Contents

A DATE WITH HTML.....	1
ONCE UPON A TIME (AS USUAL).....	3
The Ultimate Art Form.....	3
The Head.....	5
The Body ;).	6
Body Language.....	7
RULES OF ENGAGEMENT.....	8
Behind The Scenes.....	10
Make-Up!.....	11
Kwagh U Gen Yô.....	12
Now What?.....	13
THE DATE.....	14
I'll Set The Table.....	14
Break Up To Make-up.....	18
Detour.....	19
A Pinch Of Salt.....	20
How May I Address You Ma'am?.....	21
Framing It For The Sake Of Charity.....	24
Chics Without Bodies.....	24
The Request Line.....	24
To Cut A Perfect Image.....	28
THE VARIOUS SHADES OF GREY.....	29

Once Upon A Time (As Usual)

Only this time, it doesn't go as usual.

HTML is an abbreviation for HyperText Mark-up Language, which is a language widely used to format (mark-up) documents for display on the web. Since we're not going to delve into its history, it's fair enough to say that it first appeared in the early 90s and initial work on it is credited to a man known as Tim-Berners Lee, who also wrote the first HTML browser. You may also find it useful to know that HTML draws its roots from SGML (Standard Generalized Mark-up Language), [sigh] which is not our concern here.

From its first appearance to date, html has undergone several changes and improvements, notably by the World Wide Web Consortium (W3C), and is currently at version 4. There are indications that its place will be taken by XHTML (eXtensible HTML) in the near future.

I don't have to bore you with details, I'll get straight to the point as soon as you get these things ready:

1. A computer (naturally);
2. A web browser (currently, every version of Windows Comes with Internet Explorer. Other browsers include Opera, Netscape and Mozilla.) Get as many as you can. (search the Internet of course!)
3. A text editor. I'd recommend notepad, because html files are text files and standard word processors like MS Word do a great job of converting your files to cryptic binary if you're not very familiar with them.

These should get you well on your way to html-dom (as in wisdom) if you keep your interest up, and if you have a lot of brain space, say 20GB, 'cos you're gonna be sticking a lot of stuff in there, and in case you didn't figure this out, you should be comfortable with computers, or get comfortable with them before you continue (if I had to offer a suggestion, I'd say you should become a freak (at least 4 hours a day with a PC) for three weeks or more and make sure you experiment with every thing that comes your way. Have a mentor handy in case you do something you can't undo (like uninstalling your mouse and keyboard—hey, how do you get out of that one?)).

The Ultimate Art Form

“HTML is so much fun” that you'll get into an argument with the girl standing next to you-and she'll resign (if at all) with the line: “for guys like you!” -- all that scaring!

If you watch movies as irregularly as I do, you've prolly (probably) heard Bruce Willis say something like “Women are the ultimate art form”. It doesn't matter what your position is on that, let's just *strip* the html structure and see if it's something we can compare...

`<elementName>` html documents are made up of tags which are element and optionally attribute-value pairs enclosed within less-than (<) and greater-than (>) symbols (some people call these angle brackets). These tags serve to divide the document structurally and each section contains its data within the start (`<elementName>`) and end tags (`</elementName>`).

Elements are members of a collection which is predefined in the DTD¹ of the version of html you are writing against. The HTML 4 DTD for instance defines an element “P”, which can be mapped to the English “paragraph” and its attribute (not the only one though) `align` which needless to say, aligns the paragraph left, right or justifies the text if the values “left”, “right” or “justify” respectively, are assigned to it.

Drawing from this, it's easy to see that in order to mark up a paragraph of text, you would type:

`<p>paragraph text here.</p>` and a right aligned paragraph would go:

¹ DTD = Document Type Definition: defines rules a valid document must conform to by specifying data that it must/can contain and the structure of the data.

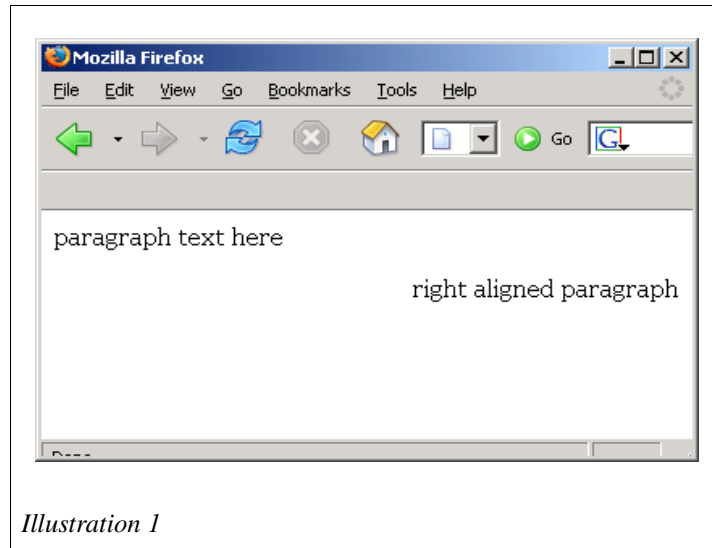


Illustration 1

```
<p align="right">right aligned paragraph. </p>
```

Go ahead, do it. Run notepad, in a new document, type in any (or all) of the code examples above. Select File> Save As, and select a path (I'd choose the desktop), then type in the file name "page.html" (including the quotes), then save. (If you're lost, find someone to ask).

Minimise all open Windows (if you saved on the desktop) and look for a file named "page" (or "page.html"). It should have an icon representing your default browser. Double-click it. It should open in your browser shortly; observe it.

Congratulations [in French accent]! You have written a functional page, but your document is quite far from being a valid html doc (that certainly doesn't mean doctor).

If you observe code_sample 1, you can see a basic and valid html page, which displays nothing, but brings these ideas to mind:

1. A html document begins with `<html>` (start tag) and ends with `</html>` (end tag).
2. Within the `<html>` tag is one head section (`<head>`) with a corresponding end tag (`</head>`).

```
<html>
  <head></head>
  <body>

  </body>
</html>
```

Code_Sample 1

3. Still within the `<html>` tag is one `body2` element.

The tree diagram and venn diagram shown in illustrations 2 and 3 are useful in showing the way you should view the document we described above. Everything in the html document is contained within `<html>` `</html>`.

2 Some documents don't contain a **BODY** element, they contain instead, a **FRAMESET**. Framesets will be discussed in detail in a later section.

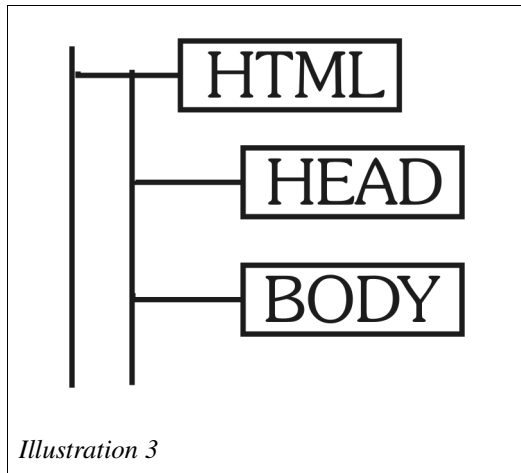


Illustration 3

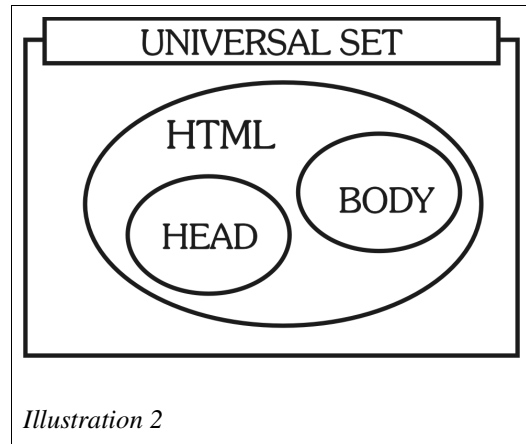


Illustration 2

That is all that is legally supposed to go into (be a direct descendant of) the html element. Anything else should fall inside the head or body section.

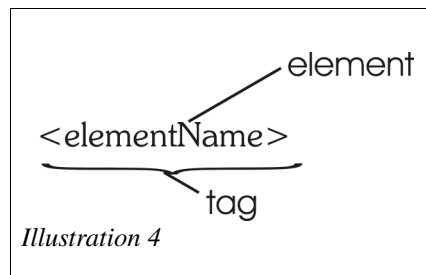


Illustration 4

The art form analogy continues...

The Head

The *first child* or element of the head (child means it goes between the `<head>` and `</head>` pair)--it should be pretty clear by now that a tag is closed or ended by the same tag name with a forward slash in front of it (`</elementName>`)... I'll take that again: the first child of the head, which will be of interest to a beginner is the **TITLE**. The title element is used to specify the title of the document and the text that is displayed in the browser's title bar.

Let's go on another tour of the faithful notepad.

Run another copy of notepad or create a new document in an existing copy, then type the following:

```
<html>
  <head>
    <title>titlebar</title>
  </head>

  <body>

    </body>
</html>
```

Save the file, then view it in your browser. You should see "titlebar" written on the browser's title bar. The title

bar is that (usually) blue bar at the very top of the program (browser) which often displays the program's name.

Other elements that can be contained in (be children of the head element include *inter alia* (use a dictionary, that's one thing you gain from reading) meta and link. We'll come across them later.

Not a bad face she's got on that cute head of hers...

The Body ;)

The first thing I'd like us to talk about is the heading collection, which provides a range of pre-formatted output, which can serve as headings for your write-ups. They are elements H1, H2, H3, H4, H5 and H6. There's

```
<html>
  <head>
    <title>My Book</title>
  </head>

  <body>
    <h1>Once upon a time (as
      usual)
    </h1>

    <p>only this time, it
      <b>doesn't</b> go as
      usual...</p>

    <h2>Heading 2, h2</h2>

  </body>
</html>
```

Code_Sample 2

no such thing as h2.5 [snarl]. Let's try some out (Code_Sample 2).

The page represented by code_sample 2 shows that it's in the BODY element that the content of your document go. We are writing a kind of book, so the page's title (set by the title element in the head) is "my book". The page begins with a heading "Once upon...", which is followed by a paragraph. The only strange thing in the mark-up presented is the tag. What it simply does is embolden the text it contains (in this case, the text is "doesn't"), as you might have seen in the preview of the page. Once again, you come across an element that resembles its English meaning (b for bold). That's how many HTML elements are.

If you are still confused by this 'tag' and 'element' stuff, take a look at illustration 4. An element is simply that word (or letter, or group of letters) that has a specific meaning attached to it. A tag is simply that element (element name) placed between angle brackets (< and >). A tag may also include its corresponding end tag (</elementName>).

Body Language

Using the “inference technique”, you shouldn't find it hard to remember that `<i>` will italicise text it contains, `<sup>` will format its contents as a superscript, `<sub>` as a subscript, such that you can visualise the output of the following:

`2²` = 4

`O₂` = Oxygen molecule

Rules Of Engagement

Since you're getting more familiar with HTML, let's start introducing some validity constructs.

These folks at W3C insist that a valid HTML document should contain the version info. It must be the first thing in the document, even before the HTML tag! It reads:

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

It can span one line or many lines, HTML ignores “whitespace” and collapses a sequence of whitespace characters into a single space. Whitespace characters include tab, line feed and space.

The declaration (of HTML version/DTD) comes in three flavours just like we have blondes and the like. There is the HTML 4.01 strict DTD, transitional DTD and frameset DTD, and to specify any one of them, you use:

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

for the strict DTD;

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

for the transitional DTD;

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

for the frameset DTD.

Don't mind the headings I use in this book. I just like to use eye (or ear) catching expressions. Also, these headings are getting too many, and I want this write-up to be like a novel, relaxed language, so much talk and suspense, and leaving you to figure out a lot of stuff so it sticks in your head long enough for you to discuss with your friends while strolling or something...

...so i removed the heading that this was to fall under... trailing the evolution of html (one sentence, but joining the last part of that sentence with the following) will reveal that certain elements/attributes (whose use will be minimal here) like the “version” attribute of the html element, which used to specify the HTML version/dtd have been given up in favour of newer ways of achieving the same goals. Your version attribute now has its job taken over by that “!DOCTYPE” stuff. Oh, I also don't know that (doctype definition) off heart, I just copy and paste it in the pages I write.

On the issue of deprecated elements/attributes (deprecated because you are no longer expected to use them) the transitional DTD supports them, basically for compatibility with older html documents. The strict DTD drops support for these elements and attributes, hence, a standards compatible browser will (should) ignore deprecated elements in a document that is labeled with the strict DTD. The frameset DTD is a basis for documents that contain other documents (quite interesting, but not so popular). Framesets will be discussed later.

I'd like to point here that most HTML constructs are not enforced. Thus you can write a document without any `<html>` tag in it but it might display properly. This is putting yourself at the mercy of programmers who made the browser. You should try as much as possible to do things the right way.

Revisiting the head of our document, let's introduce another element; the META element. There can be any number of this element as necessary to supply meta data for the document in question but this is one element which FORBIDS an end tag. In other words, just specify the opening tag, with the attributes you want, and continue with the next element you want. The meta element contains all its information in its attributes. It's an empty element. An example of its use goes thus:

```
<meta name="author" content="fanen">
```

Before I explain what that line does, let me leave you with a little something to broaden your thinking with:

create a file from code_sample 3 and view it in your browser. Elements such as H1 should have both start and end tags, while others such as P may or may not have end tags since the end tag may be inferred at the next “block-level” element.

A-ha! Elements are classified in HTML into inline and block-level elements. A typical inline element is the **B** element. It doesn't cause the text it contains to begin on a new line as in the example above. The text may only wrap to a new line if the output media is not wide enough to contain it on a single line, and it won't necessarily be at the **B** element.

Exceptions as usual. The `p` element is labeled as inline in the html 4 DTDs, but it always causes content to

```
<html>
<head>
  <title>end tag test</title>
</head>
<body>
  <h1>THE UNCLOSED H1 TAG

  <p>that may lead to unwanted
  results</p>

</body>
</html>
```

Code_Sample 3

begin on a new line. This isn't difficult to ~~explain/disprove~~ (whatever) because of a second definition of inline elements. Inline elements may contain other inline elements but not block-level elements. Hence (me and this word), a block level may contain both inline and block level elements (except of course where it doesn't make logical sense, like a heading containing a paragraph. If your browser formatted the paragraph in code_sample 3 as a heading, then it was some oversight by the people who made it. It shouldn't happen like that).

We'll get back to meta soon...

The concept of inline and block-level can be quite tricky to grasp, so I included a little example here, with a little formatting (don't worry yet if you don't understand the formatting code) just to highlight the points. I highlighted the non-html parts so you know what to ignore.

Let's go through the results of code_sample 4. First is a paragraph, which is an inline element (formatted in yellow background and red borders). A heading (**H1**, which is block-level) is nested (contained) within the paragraph. You easily spot that the **H1** element is not wrapped in the red border and surrounded by the yellow background. This proves the fact that the `P` element is an inline element, and **H1** block level (inline elements cannot contain block-level elements). Draw your conclusions from the output of the others. You shouldn't find it too hard.

```

<html>
<head>
  <title>inline-block elements</title>
</head>

<body>
  <p style="background-color:yellow;border:thin      solid
red;">inline paragraph with a nested
  <h1>block level heading</h1>
  </p>

  <p style="color:blue;border:thin solid blue">the      inline
paragraph with a nested
  <div style="color:red">block level div</div>

  <p style="color:white;background-
color:pink">inline paragraph
  with a nested <span style="color:maroon">inline      span
element</span>
  </p>

  <h1 style="color:blue;border:thin solid
red;">block h1 with a
  <h1 style="color:red;border:thin dotted
red;">nested h1</h1></h1>

  <div style="background-color:yellow;">Block      level
div with <p style="color:maroon">
  a nested inline paragraph</p></div>
</body>
</html>

```

Code_Sample 4

Illustration 5 is an image of the output in Mozilla Firefox.

Behind The Scenes

Now, we were talking about the meta element. Yes, your documents can do without it (them) and its effect is not immediately obvious but there are a few reasons why we should use and specify them.

Our first use-case will be an analogy between authors of books and html documents. If you want to let the world know that you are the author of a document, sure thing, you can simply write it visibly in one of your pages, but using the meta element allows interesting things to be done with that information. Search engines for instance, can index that page with your name, so that when someone searches the Internet with your name, that page shows up in the results. Meta also has some other uses, but that is best left as a topic of research for you.



Illustration 5

To specify the author in meta data, you may use the following code in the head of your document:

```
<meta name="author" content="fanen">
```

The meta element forbids and end tag. Remember that.

Staring is not polite, so let's shift our gaze. Now, what haven't we seen about this girl? We've even given her beautician a tip about her hair (validity/doctype definition), but we're not satisfied. What could be the missing link?

Link. That's it! We've not talked about the link element. The link element is another ornament for the head, which simple as it looks leaves a big impression.

Make-Up!

Don't get confused. While to a girl, make-up includes such things as "lip-paint" and blah-blah... make-up to a html doc most commonly refers to [CSS](#) (Cascading Stylesheet Language) instructions packaged in ".css" files, within which there is beauty to be applied to your markup.

What I'm trying to say is: what makes the make-up manifest on your html doc is the "link" between make-up and document. Take for instance, a simple CSS file named "roll-on.css", which contains the CSS instruction:

```
body {
    background-color: green;
}
```

forget what it means for now;

and a html file. Just create one to your taste. Put them in a folder (the same folder) and double-click the html file. You see nothing special.

Now, open your html file in notepad and add the following line to the head:

```
<link rel="stylesheet" type="text/css" href="roll-on.css">
```

Link is another element which forbids end tags!

Save your file and double-click it to open it again. You should see a green background in the document! The missing link at work!

Kwagh U Gen Yô

Translation

Not of that line though!

For all of you “wannabe” multilingual publishers out there, suppose you have an English language document, and you have another copy in Tiv, and a host of other languages. You want to let your readers know (for one case) that there are translations of this document for other languages too. You can do this by specifying a relationship between two of such documents.

You may have noticed the “rel” attribute of the link element in the previous example. That's the relationship and the “stylesheet” value in one of the accepted values for that attribute. Let's examine a few of them:

Start

Tells whatever (or whoever) is using the document that this is a document in a series of related or dependent documents, and that the first document in this series can be found at the location specified by the href attribute.

Take our “missing link” example. The relationship said this was the stylesheet for this document and that it was located in “roll-on.css” as specified by the href attribute.

Prev

Says the document specified by href is the one preceding the current one and they are all in a related series.

Next

Similar to prev. What is the real meaning of next?

Content

Specifies the “Table of Contents” for the current document, implying also that there may be other related documents.

Alternate

Which is our concern here, has a number of interesting applications. The “alternate” relationship can be used to specify another copy of your document suitable for other media.

```
<link rel="alternate" media="aural" href="for-aural.html">
```

will specify a file suitable for blind people, who are “viewing” this document with the aid of a text-to-speech device. Other values the media attribute can take include screen and print.

Remember that the link element always goes in the head of the document.

Another application of the alternate relationship goes with the `lang` attribute (in the link element) to specify a translation of the document to other languages. Consider the following mark-up.

```
<link rel="alternate" lang="fr" href="french-trans.html">
```

which simply says a French translation of this document is available at “fr-trans.html” (which is a relative address. More on addresses later.)

The values of the `lang` attribute are specific abbreviations for many widely used languages. US english has a representation “en-US”, German: “de” etc. You may find more information on this by perusing the HTML4 specification (www.w3.org).

Now What?

That's not all that can go into the head of the document or the link element for that matter, and we also didn't say all about meta too. That just gives you an idea about their use, and you don't have to know all their possible uses in order to make good HTML pages. All the other stuff would become obvious as your webdevelopment knowledge expands.

All that hardly answers the question. The question was “now what?” Oh, now let's pick up paint and brush, and begin to paint the “skeleton” of our supermodel...

The Date

Now that you have (hopefully) some good understanding of how HTML is meant to be used, let's start putting your skills to work.

Let's see, eh... where do we start? Great a letter.

Consider this:

```
<html>

<head>
  <title>No title</title>
</head>

<body>
  <p align="right">The Address of My Boss<br />
  My Boss's Location<br />
  State etc...<br />
  </p>

  <p>No, My boss's address here<br />
  And my own on the top right<br />
  etc...</p>

  <u><!-- this shud underline the headin-->
  <h1>REQUEST FOR PROMOTION</h1></u>

  <p align="justify">
  This here is the content of the letter,
  and our writing it means it won't be only
  one paragraph long, so there's a few points
  i'd like to point out.</p>

  <p align="justify">
  Don't get tempted to use the &lt;br&gt; tag
  to create "false" paragraphs, the html dtd
  supplys a &lt;p&gt; tag to take care of that,
  even if that is more work</p>

</body>

</html>
```

opening this document in your browser should display a neatly typed request for promotion. A few new ideas were introduced. We'll discuss them over dinner.

I'll Set The Table

```
<table></table>
```

There, that's our incomplete or invisible table, but it has the four corners-sorry, start and end tags that make up one.

Going straight to the point, we introduced two new ideas just previously. The first was (maybe not, but I want

to discuss it first) the “comment”. If you're knowledgeable in programming, this is no different from the comments in programming languages (or programs). They are just some statements you insert alongside the actual code, which doesn't affect the program or its output. They are usually put there just to explain some tricky point, or remind you of what you were trying to do at the time you put them... (mmm...) the comment we inserted there was “this shud underline the headin”. That simply says what the `u` element there meant. The comment was like **so**:

```
<!-- comment text -->
```

The start tag in this case was simply: “<!-- “ put a space between the symbols and your actual text- and the end tag was “-->” (put a space before this too.) The comment text can span multiple lines, just use the “end tag” when you are done. You can think of whatever use you want to use comments for. Just remember that they are ignored by the program parsing your html code.

Like I said, the `u` element simply underlines its content, whatever they be. Of course, only text can get underlined!

Oh! silly me. I did all the talking before setting the table. Here's the table:

<i>Ingredients</i>	<i>Quantity</i>
Beef	1kg
Vegetable oil	1 ltr
Rice	1 cup
Beans	½ cup

Did I mention you were gonna be the cook? Here's the procedure³:

```
<table border="1">
  <thead>
    <tr>
      <th>Ingredient</th>
      <th>Quantity</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Beef</td>
      <td>1kg</td>
    </tr>
    <tr>
      <td>Vegetable oil</td>
      <td>1 ltr</td>
    </tr>
    <tr>
      <td>Rice</td>
      <td>1 cup</td>
    </tr>
    <tr>
      <td>Beans</td>
```

³ I just specify the code that is relevant to the discussion. When making a page from such code however, specify all the necessary data needed to make a valid HTML file.

```

        <td>&frac12; cup</td>
    </tr>

</tbody>

</table>

```

Voila! Our table. This is a simple example of a table in html and it does introduce some new elements and semantics.

The **TABLE** element can be considered to be “miniature html” or “embedded html” since it divides its structure into head (**THEAD**), body (**TBODY**) and the unique one, the foot (**TFOOT**). **THEAD** and **TBODY** share the same semantics as the html head and body, and hence, quite naturally, **TFOOT** should easily fall in place.

Quite unnaturally though, the foot of the table can come before the body, such that a table's skeleton could be:

```

<table>
    <thead> </thead>

    <tfoot> </tfoot>

    <tbody> </tbody>
</table>

```

This is done in order to allow a browser to begin displaying a table without receiving all the potentially large data that the body contains.

Another set of strange elements that our previous table exposes are the cell (table cell) definition elements: **TH** and **TD**. The **TH** element identifies “header” information (ingredient and quantity in our case) and browsers are expected to format them accordingly (bolder text as can be seen in our example), while the **td** element defines ordinary entries in our table. One mnemonic that could be useful in differentiating between **TH** and **TD** is **TH** for **table heading**, **TD** for **table data**. These two elements go into a **TR** “parent”.

Infer from the last sentence that **TR** is the only way to create rows in a table. In contrast, **TH** and **TD** are the only ways to create columns. Needless to say, **THEAD**, **TBODY** and **TFOOT** should only contain the **TR** element, which in turn should only contain **TH** and/or **TD** elements. I ruled out “and” because it is possible to have both of them in one **TR** element but it hardly makes sense.

Head still spinning? It's quite common. The **HTML** table model came as a surprise to me. I just couldn't agree this was how to design a table—it was so illogical -but you start designing perfect tables almost “automatically” once you get the hang(over) of it. For this reason, we'll look at it again, step-by-step.

```

<table>

our opening table element.

<thead>

table header

<tr>
    <th>Ingredient</th>
    <th>Quantity</th>
</tr>

```

Notice the **TR**, containing two **TH** elements, meaning two columns and the single **TR** element, meaning one row—yet.

```

</thead>

close the head.

```

```
<tfoot>
</tfoot>
```

the tfoot element, empty now because we don't need it, but there, so you get used to seeing and putting it there.

```
<tbody>
  <tr> <td>Beef</td>
      <td>1kg</td>
```

we'll stop there for brevity.

The body as usual contains the actual table data. It follows the rules, one TR, with two TDs, because our table is a two-column one. If you started with three TDs, you should maintain that number, otherwise, you just re-live an expression which I find very blunt and funny-Garbage In, Garbage Out- or even invent a new one, TITO (Thrash in, Thrash Out. No disrespect to the original trademark, for Makurdi residents.).

All that talking. What that simply means is: stick in an extra TD in any one of the rows (trs), save and view your document and decide if you like it or not.

[yawn!]

You still might be having some difficulty with that, so go on and experiment now. Try something like the age of members of your family in “name” and “age” format – or think of something else if you're concerned about your(s) (why doesn't that word have a plural in English) privacy(ies). In the mean time, let me stray a little, and try to see if I can cook up a good joke.

This was a real occurrence though.

In my all boys school-many of you know how boys behave when there are no girls around for them to try to impress, hence check their “refreshingly” comic activities and utterances. One of such activities called stroking involves a group of students -in this case, on their way to the “market square”- each attempt to make humourous remarks on anything from a passer-by through a popular teacher, to the guy you're talking to, so that everyone can laugh.

On this day, one of the guys suddenly brings up a true topic (about girls of course): a mathematics teacher is holding extra lessons for a group of girls!

They talk about this all the way to the market square and are headed back for their classroom. Regular school hours are over for the day and they all know the place so well that no one needs to look up in order to find the way- they are all giggling and asking questions:

“Where?”

“From which school?”

“Na true?”

They enter the class and the questions are still pouring:

“Which teacher?”

“Our mates?”

they they start to realise that they are not alone in the class and silence begins to descend.

One of the guys, unaware that the fun is over, asks loudly, amidst the silence:

“fine-fine girls?”

Getting no response, he looks up—to a dozen feminine faces and a math teacher laughing at the tops of their voices!

Break Up To Make-up

Before I get **criticised**, if you saw the grammatical blunder there (and the one very soon), it was (and will be) on purpose. Also, typing the letter “z” is much more difficult than “s”, that’s why I use “z” less often than not.

Oh how that expression (the heading) captures the goal of CSS (Cascading Stylesheet Language) a.k.a make-up!

The goal of CSS is to separate the formatting of html documents from the content (data) to make for easier maintenance of document and application of similar formatting to numerous documents. It also brings more formatting options than conventional HTML offers (or offered).

And how it works?

Well, I don’t know about ladies’ make-up, but you apply CSS by making “declarations” that act on certain HTML elements (selectors)...

```
body {
    background-color: yellow;
    color: red;
}
```

This is a CSS **rule**. A CSS rule contains a valid HTML element known as the **selector**, followed by a left-pointing curly bracket (or brace). After the brace are **declarations**. We have two declarations in the rule above: “background-color: yellow” and “color: red”. A declaration in turn is made up of property, and value. Clearly, in the case just presented, our properties are “background-color” and “color”, while our values are “yellow” and “red”. The rule ends with a matching right-pointing curly bracket—just like start and end tags in html. Notice also that each declaration is delimited from any other declaration by a semi-colon (;) and properties are separated from their values by a colon (:). CSS also ignores whitespace, so you are free to place your text however you like, as long as you use the delimiters (“{”, “}”, “:” and “;”) appropriately.

The CSS rule we just described makes the body element of the document to which it is applied have a yellow

```
<html>
  <head>
    <title>CSS example</title>

    <style type="text/css">
      body {
        background-color: yellow;
        color: red;
      }
    </head>
  <body>
    <p>Red text on yellow
      background.</p>
  </body>
</html>
```

Code_Sample 5

background with red text.

Excellent. Now, how do we apply it?

Examine code_sample 5.

One good thing about CSS is its strongly English-language-like syntax. The English speaking person gets a boost for that, except you have to remember that the spellings have to be in American English.

Detour

You know, there are a few video games on consoles like PS2 and Xbox which take their (artificial) intelligence further by reducing the difficulty of a level in which you failed (woefully). Take for instance a scene in which you are in a car and (naturally) there are many roads to take, but you have a compass that shows you the right way and since you're new (or commonsense challenged), you keep taking the wrong turns until you lose. Then you re-play the level only to find out that the wrong turns are blocked. The machine is actually saying: "you big dope! You're not supposed to go that way!"

[end detour]

If you were one of those who criticised the heading "break up to make-up", I think it's time I let you in on why I chose that heading. You (should) know that CSS applies formatting to associated documents. That's similar to a girl making-up her face with all those things... compare "make up" and "make-up". I believe they have different meanings. Then, as we shall see in due course, CSS allows you to separate the make-up and the HTML into two files, keeping the html in one file and the css rules in another file. Break up to make-up! You don't wanna be a big dope now do you?

Big dope or no big dope, you may have noticed that our css rule was placed placed in the head of the html document, under an element named style. Now this poses a serious problem if you have a large number of pages that need to look similar – even with copying and pasting. For one, the size of each page grows bigger (since your document will usually contain much more than one css rule) , consequently increasing the download time if it is served over the internet.

The elegant (well?) solution to this problem is the css file. You simply cut and paste (or rewrite) your css in a fresh file (this time omitting the style element) and save it, preferably with an extension ".css", then "link" it to your html document(s) *like so*:

```
<link rel="stylesheet" href="the_file_name_you_gave.css" type="text/css">
```

You gotta know where to put that if all we've been discussing for the last 20 pages has got into you at all.⁴

You know the body in our css file? It sounds similar to the body of our html document right? It is. I mentioned that a few pages ago, in case you are still battling with it. That's the selector. A selector is a (any) valid HTML element for which the enclosed declarations are applicable (applied).

Take some time out to examine all the selectors you now know, then realise that the declarations are even much more fun to learn (like I said) because they are very english-like; hence we create a broader css file:

```
body {
    background-color: yellow;
    background-image: url('existing_picture.png');
}

p {
    font-family: cursive;
}
```

H1, H2, H3 {text-transform: uppercase;}

The third rule above is a comma-separated list of selectors. This means that the enclosed rules apply to H1, H2 and H3 elements. Pretty straightforward. The font-family property simply sets the display font of the selected element. Cursive is not a kind of font, it's a kind of "font-family". Font families include "serif", "sans-serif", "monospace" and "cursive". If you're familiar with fonts, the "serif" family includes fonts with decorated edges, like times new roman. The "sans-serif" family like the french word "sans" suggests do not have decorated edges. Examples include arial, verdana. The monospace family includes fonts whose characters have equal widths (like typewriter fonts). Members of this family include "courier new, lucida console". Finally, the "cursive" fonts are simply decorative fonts, like comic sans ms. The font family rule given above excludes the use of individual font names because they are not guaranteed to be found on all systems that the page will be

⁴ And if you don't know, I strongly suggest you re-read the *head* section of this text. If you don't know what the head is, ... hmnn.

viewed. We gave the family name instead, leaving the system to substitute any font from the specified family present on the viewing system.

Of course, we can specify the font names directly, there is even a way to specify them, and then fall back to the font-families, where the specified font is not found:

```
font-family: verdana, arial, sans-serif;
```

Here, the verdana is first asked for. If not found, arial is requested, and finally, any sans-serif font of the system's choice is used if none of the first two are not found.

The text-transform property simply specifies the case of the letters of its selector. Accepted values are uppercase, lowercase, capitalize, none and inherit. Experiment will easily differentiate their effects. Inherit however specifies that the parent element's text-transform property value be applied to this element.

Background-image utilises a “function” url(), which takes the filename of the picture enclosed in quotes (single or double). “url()” is just a way of assigning addresses to css properties that require them.

Background-color specifies the “background colour” of the document in question. (as if that needed explaining). The only thing that needs explaining here is the value of this property. You see, the key, is the RGB code in hexadecimal (base 16, if you're a mathematician), or the rgb(red, green, blue) function (which takes three parameters, red, green, and blue, all numbers, and comma-separated within the parentheses), or as we saw, the name of the colour, if it is popular enough that you think the browser should know it. This last method is not recommended, i just used it for the sake of simplicity. The first two methods are recommended, such that you have to know the ratio of the combination of Red and Green and/or Blue that will give you yellow (artists, your specialty) if you wanted to use any of those two methods.

A quick example of this use is thus:

```
background-color: #ff0000;
background-color: rgb(255,0,0);
```

The colour specified in the two rules above is Red. In hexadecimal, “F” is the highest number (as opposed to 9 in decimal or base 10). In the rgb function, 255 is the highest number, and 255 in base 10 converts to FF in base 16. This maximum number means “red at the highest intensity that this medium can display!”, the other values, green = 0, blue = 0 simply indicate the absence of the respective “channels”.

The Hexadecimal code is always preceded with a “#” and consists of 6 digits (base 16 digits) thereafter, two per channel, the channels being Red, Green, Blue (as in RGB) respectively.

This chapter is primarily intended to knock this into your head: always separate the formatting (which should be CSS) from your HTML (put them in separate files as shown). This makes your pages easier to maintain and reuse since each contains relevant data, but don't feel guilty if you really need to slap in some CSS alongside your markup.

A Pinch Of Salt

... in your cornflakes.

For starters, that's all we're gonna mention about CSS. Fortunately for you, CSS is so easy to learn that what I said about it, plus a little research and practice would make you an expert.

That ought to have quelled your CRITICAL hormones... and made you realise why our streets are paved with

damsels (in distress), and why they are in distress is a result of a worrying phenomenon called FOES (Fear of Embarrassment Syndrome), which is getting increasingly prevalent. You see, the guy on sighting a DID and obviously shutting down the train of thought leaves it at the mercy of gravity (culled from grave, meaning bad...), and if you remember your elementary physics, the train accelerates downwards at 10m/s (meters per second), whilst the feet simultaneously LEVERAGE the guy towards *la dame*, and this kind of instant pressure causes the mouth to utter:

How May I Address You Ma'am?

Excuse me? (Do i know you?)

Then of course our little guy realises the GRAVITY of his hurried (accelerated) action, and wishes the ground would swallow him. He vows never to go through this again.

Our guy obviously wanted to start a relationship... yep! RELATIONSHIP

```
<link rel="stylesheet" href="the_filename_you_gave.css" type="text/css">
```

That's the way you link a stylesheet that has been put in a separate file to a particular HTML page. You specify a link in the head of the html document, with the relationship as "stylesheet".

I believe you know also that relationship can mean something other than the common boy-girl type. Also, the "big dope" phenomenon applies here too. You may choose to be a big dope and settle simply for the "stylesheet" relationship, or you can speculate, and let your curiosity lead you into research.

Curiosity killed the cat... whoever it was, whatever year it was.

Satisfaction bore it back... Lefteye (from TLC) in Curious, by LSG, Lefteye and Busta Rhyme, 1998 (in R & B Album by LSG (gerald Levert, keith Sweat and johnny Gill).

A little pep talk:

Big dope: /big doup/¹ a person whose first question to a completely strange girl takes the form "what is your name?"
² a person...

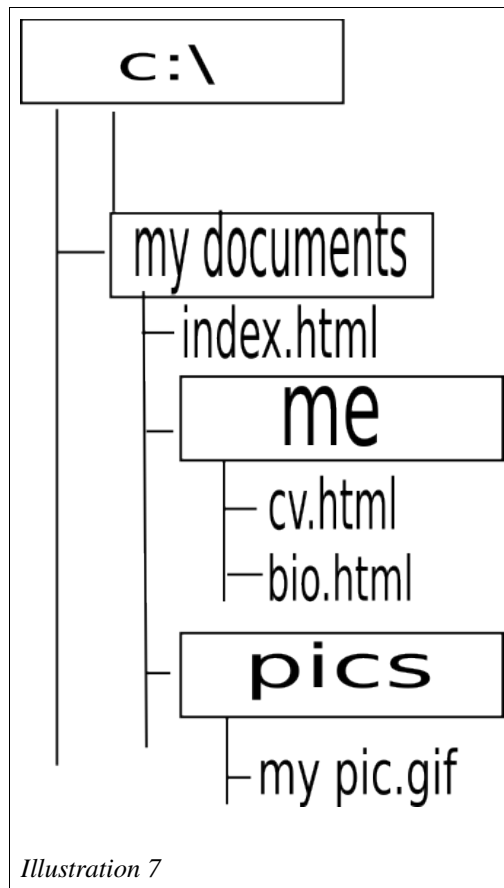
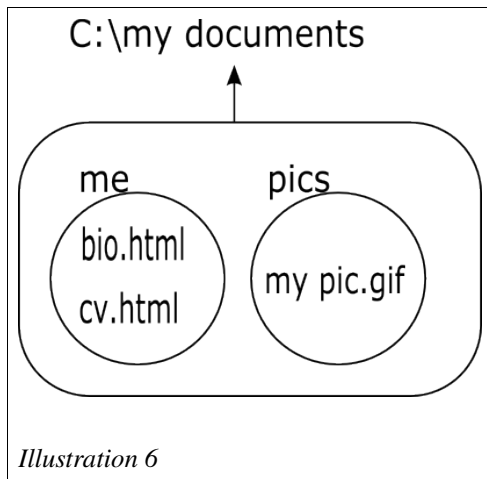
Statistics experience shows that a good way to get a girl's name "passively" when meeting her for a second time is to ask this simple question: "what did you say your name was?"

Statistics was ruled out in the last sentence because, *Pressing Issues* would have it that, statistics, like many other elegant mathematical disciplines, prove to be irrelevant.

The concept of "address" is a nebulous (I just learn't that word) one. What's your address? How may I address you (ma'am)?

You often hear of terms like URL and URI (pronounced letter-by-letter). I'm not in a linguistic or technical position to start discussing URL and URI, but I will try to create a picture of how files "lie" (not related to truth) in a computer and how to refer to them from "links" (link elements) or anchors (A element).

First, we start with your local computer. You may view the diagram as a venn diagram with no intersections, but what I'm trying to show is folders and files: similar shapes depict folders on the same level, while file names are written. The tree structure is just an alternate visualisation of the same concept.



These pictures represent my “my documents” folder, which is where i decided to store my html pages and a picture of myself. I have my biography and CV in a folder named “me” and I have my picture in another folder named “pics”.

With this hierarchy, if i wanted to view my biography in a browser on a Windows PC, I would type “c:\my documents\me\bio.html” in the browser's address bar.

That is an example of an absolute URL (uniform resource locator). Sound like English classes? An absolute url “fully qualifies” the resource or file in question. That is, the address is complete, you don't need more information than this if you are looking for my biography.

Then, say you were reading my biography and I made mention of my curriculum vitae, and you wanted to read it. Of course, I anticipated that, so I provided a “link” in the document (bio.html) that you could click to retrieve and display my CV (cv.html).

The link I provide in this case is what is called an anchor (**A** element) and is different from a “link” (**LINK** element) in that its (anchor) presence is shown in many browsers by underlining its text, and a change of cursor type when the mouse cursor hovers on it. It is also available for interaction (clicking, or activating by any other means) and it retrieves the resource identified by its “href” attribute.

In this case of my CV, the anchor's href attribute is set to just “cv.html”. This is what is called a relative url.

Since both the bio and the cv are in the same folder, it is easier to have “href='cv.html'”, which means “locate the cv in the same folder”. Relative urls are not limited to files in the same folder as. They can also be used for files in a folder inside the same folder as...

looking at our drive c from the recent past, we can use the relative uri syntax for the bio from “index.html” (the tree diagram is more lucid):

```
<a href="./me/bio.html">Biography</a>
```

This line will do the job of placing a line inside index.html (absolute url= c:\my documents\index.html)),

which when activated would open the biography in folder “me” named “bio.html” (c:\my documents\me\bio.html)

It doesn't end there.

```
<a href=" ../index.html">Home</a>
```

will insert a link into “bio.html” that displays the page index.html, located one folder above “me” (that is, c:\my documents\index.html).

Relative urls make it easier to move a page or a set of pages to a new location, so long as their relative positions are not altered. For instance, if I move “bio.html” and “cv.html” to the root folder “c:\”. the url that links between the two pages will still function, since they are still in the same folder or on the same level in the file system tree.

I won't try to be political, so I'll let you discover that relative urls are a helpful way to go (ha ha).

Also, you can modify the resolution of relative urls by specifying the “base” element in the head of the document:

```
<base href="c:\my documents\index.html">
```

no end tag allowed.

What this means is that any relative uri (uniform resource identifier) specified in a document that contains this base declaration will be resolved as if it was in the document index.html.

If you choose to go the way of the BASE (ment), the element (base) must appear before any element that refers to an external source. This makes sense to me cos the base url will be known before any urls are resolved, so some don't behave differently.

Over to you now. You'll gain some more insight by experimenting... think of something. Maybe a group of pages for members of your family organised in this hierarchy:

FAMILY NAME

 family home page

 MALES

 pages for each male family member

 FEMALES

 pages for each female family member

Capital letters indicate folders, indentation indicates folder level.

Try linking to each of the pages from any and/or each page.

Well, now you know how to address ladies when you meet them – I can hear that nod and hum of acknowledgment, so I might be forgiven if I made this look more difficult than it turns out to be ...

Back to our hypothetical train of thought, this lady gets more attractive as you observe her more carefully... and now you know her name, you know what's in her head (and its obvious she likes you) but that's the thing with most ladies: she won't admit it (I think it hurts her pride), so she can't agree with you. You have to frame it such that it looks as if you instigated everything, and she fell helplessly into it – even though she knew all along that she was going to fall –ladies... oh well.

Framing It For The Sake Of Charity

I actually knew a chic by that name... (Charity).

Getting to date a chic is a process that I like to regard as child's play. If you don't get blasted off on the first attempt, then you both know where you're heading, but pretend like that's not it until there comes a time when you have to stop kidding and frame it such that it means a number of different things, and the obvious one is what you intend, but there is not enough evidence to pass the verdict because you're a good actor. At that moment, she asks you what you mean, then you, the historian, give a brief recount of your encounters with her so far.. If she's gonna agree with you, she asks further “are you trying to say that...”, then she remembers it's your job, and pauses –then you say “yes” - “yes what?” -you're getting impatient and tense... “I wanna be more than *just* a friend”.

Of course she won't say “yes”. Anything in the realms of “I'll think about it” mean yes. Even silence does. From that point on you have to stop acting... you've got it made! How many girls would agree with me on that one?

[clears throat]

I'm terribly sorry... we were talking about framing chics – sorry, **frameset documents**. Recall when we talked about DTDs and we mentioned the frameset DTD? Now's when we discuss it.

The frameset document is one which does not contain a **BODY** element.

Chics Without Bodies

Nobody? (I mean, no_body?) Yes! The **BODY** element is replaced by `<frameset>`. Recall also that I said the frameset DTD (document type definition) is defined as the Strict DTD + Transitional DTD; frameset documents are documents that contain other html documents. The “contained” documents follow the HTML constructs we have discussed this far, and thus can be defined in either the transitional or strict dtd – or even frameset. This may be why the frameset dtd is said to include both the transitional and strict dtds.

Into the frameset document, it's structure follows the regular HTML document, but its body part is replaced by **FRAMESET**. This allows the document to be divided into independent sections (frame element). Since these sections can each contain other documents (which is why I called them independent), they give you the ability to display more than one html document in the same page. You can for instance display the links on your site in one frame (section) and the actual site content in another frame.

The Request Line

Let's take some calls from the request line.

Caller number one:

Yo! “This is tesem kwagh u u lu oron la” from “Takeda wou!”

Translated, that line means:

Yo! This is “show me what you're saying” from “your book”.

```
<html>
  <head> </head>
  <frameset cols="*, 4*">
    <frame src="toc.html">
    <frame src="content.html">
  </frameset>
</html>
```

That's all our frameset document contains. You can save and preview it if you like, but no pages will be displayed, since the two source (src) files we specified do not exist yet. Just read on.

This document contains a html element, a head element, then a frameset element. The frameset element has an attribute cols which is synonymous with columns. It enables you to divide the document into a number of columns by specifying their widths in a comma separated format. The widths may be relative, precise or percentage values. In our frameset element above, we used relative widths: * and 4*. It means that the first column is 1/5 times the size of the view port. That is, the view port is divided into 5 equal sections, the first column's width is equal to the width of one such section, and the second column takes the width of the remaining four.

The same results could be achieved by

```
<frameset cols="20%, 80%">
```

which are percentage values. Precise values simply accept numbers, which represent the width of each column in pixels.

You may also say:

```
<frameset cols="20%, *, 4*">
```

in which case, you create 3 columns. The first one 20% wide, and the remaining part divided among the last two columns in the ratio 1:4.

Inside the frameset element, you include the **FRAME** element whose effective attribute is the source html file to display in each frame.

```
<frame src="toc.html">
```

In our example, we created two columns, so we specified two **FRAME**s.

Frame number one is the one with "*" width, while frame 2 is the one with "4*" width, so we take note of that as we assign the source files to them. The first frame element goes for the first "col" (frame) and so on.

You can also divide the document into rows.

```
<frameset rows="60%, 40%">
```

This divides the document into two rows, the first 60% high and the second 40% high. The same semantics for columns apply to rows.

A mixture of rows and columns may be achieved by treating each entity (row or column) as a viewport that can be divided further.

```
<frameset rows="70%, 30%">
  <frameset cols="20%, 80%">
    <frame src="toc.html">
    <frame src="content.html">
  </frameset>
  <frame src="description.html">
```

```
</frameset>
```

Here, the viewport is divided first into two rows, 70% and 30% tall each. The 70% tall one is further divided (instead of a frame element, there is another frameset element) into two columns, 20% and 80% wide. The second of those first two rows remains unchanged. The result is shown in illustration 8

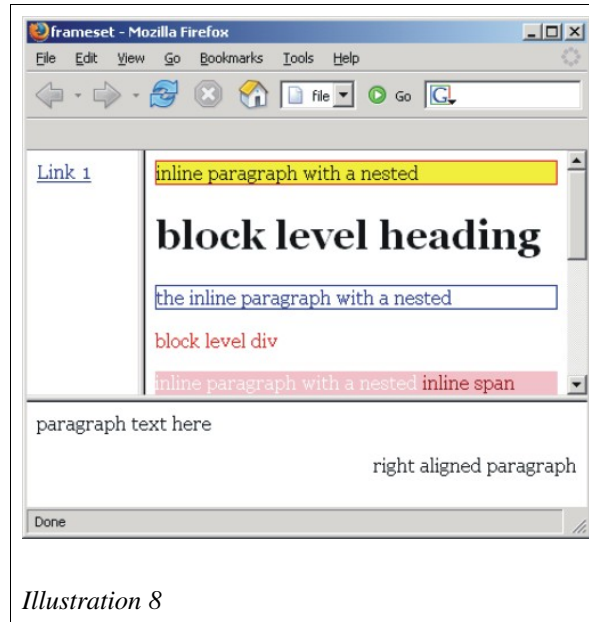


Illustration 8

Keeping track of which frame is which is simple but can be confusing. It's just a matter of keeping track of their creation hierarchy. In the example above, we had two rows. If it stopped there, *they* would have been frame 1 and frame 2. However, frame 1 was divided into two more frames. They became the *first* two frames, frame 1 and frame 2, and the original frame 2 (second row) became frame 3.

When writing markup, you may keep track of them by remembering that any frameset should contain an equal number of frame elements to the number of rows or columns it specifies. Thus our example goes:

the first frameset defines two rows

the first row contains another frameset

this frameset defines two columns (20 and 80 percent wide)

the first column contains "toc.html" (frame src="toc.html")

the second column contains "content.html" (frame src="content.html")

Let's complete our lecture on framing people- oops, documents by making a functional document and then explaining a few points.

First we create the documents that the **FRAMESET** will display (contain).

Toc.html

```
<html>
  <head></head>
  <body>
    <a href="link1.html" target="content">Link 1</a>
    <a href="link2.html" target="content">Link 2</a>
  </body>
</html>
```

Content.html

```
<html>
  <head> </head>
  <body>
    <h1>The contents!</h1>
    <p>page displays whatever i want on this site</p>
  </body>
</html>
```

Description.html

```
<html>
  <head> </head>
  <body>
    <p>this could contain finer details...</p>
  </body>
</html>
```

Frameset.html

```
<html>
  <head> </head>
  <frameset rows="70%, *">
    <frameset cols="20%, 80%">
      <frame src="toc.html">
      <frame name="content" src="content.html">
    </frameset>
    <frame src="description.html">
  </frameset>

  <noframes>
    <p>Please use a browser that is can handle frames</p>
  </noframes>
</html>
```

In the samples above, the new entities are highlighted. `target` is another attribute of an anchor (the `A` element). It specifies where the file indicated in href should be displayed.

That brings up the name attribute in frameset.html. Name gives an identifier to any element it is applied to (many other elements, not just frame). We explicitly chose one frame and named it “content” so that we could use it as a **target** for the links in our “toc.html” file. Create all the files mentioned in this discourse about framesets (you can apply whatever knowledge you have gained so far, you don't have to duplicate the examples), then open “frameset.html” in your browser. If you open a file other than frameset.html, then you will only see that individual file, not the frames. Play around with the links, make additional pages—experiment, you'll get a firm grasp of frames, targets and names.

The name element applies to a lot more **HTML** elements but its other uses are mostly for people who write scripts. Scripting is another aspect of web design which is (I hate that expression “beyond the scope of this book”: it makes the topic sound too difficult to learn) not covered in this text.

To Cut A Perfect Image

Nothing cuts an image as becoming as a tall guy in well-tailored suits and a tie...ties. How do you wear one? Gosh, I don't even wear suits –yet.

All along our discussion, we've been looking at **html**'s intricacies. I've been focusing too much on its technicalities all in the idea that other things are easy enough to be left untaught. One of such is how to insert images in **HTML** documents.

You simply include an **IMG** tag. **IMG** has **src** (just like **frame**) which indicates the source picture. The picture has to be in a format that the viewing browser supports. Popular image formats include **JPEG**, **GIF** and (my personal favourite) **PNG**.

```

```

The **IMG** element forbids an end tag, hence has no content, but it has an **align** attribute, which can take values “left” or “right”. This attribute does a couple of interesting things (when there is text close to the image) which are easy enough for you to find out. **img** simply inserts the specified image at the point of its occurrence (except when **align** is used).

Surprise! surprise! **IMG** and **FRAME**'s **src** attributes accept urls with the same semantics as the anchor's **href**! (see “How may I address you ma'am?”)

The Various Shades Of Grey

The golden years have come, and are going.

The word “form” has a number of meanings attached to it. First, it implies some thing's physical attributes as in “the various forms of life” and in formal circles, implies a device used to collect information, especially personal, from people. We are concerned with such devices.

Html forms contain controls in addition to necessary markup. Think of controls as those things with which you interact with forms.

Controls include checkboxes, radio buttons, menus, text input controls, file select and hidden controls.

We'll illustrate the use and structure of forms with an example:

```
<form method="get" action="mysite.php">
  <label for="name">Name:</label>

  <input type="text" name="name">

      <label for="age">Age:</label>
  <input type="text" name="age">

  <input type="submit" value="Submit">
</form>
```

Typically, a form is delimited with the form element (`<form>` and `</form>`). A **FORM** element usually has attribute `method` (values: `get` or `post`) which specifies how the form data will be sent to the form processor, whose address is specified in the `action` attribute. Both these attributes are dependent upon concepts beyond html, so we won't say much about them, but `action` in the form above was the url to a PHP⁵ script that was written to process the form on submission.

Most form controls are specified using the **INPUT** element (which forbids an end tag). Its major attribute is `type`, accepted values include any of: `text`, `password`, `checkbox`, `radio`, `submit`, `reset`, `file`, `hidden`, `image`, `button`.

`Type="text"` was used in the example above, and `type="submit"`. Text creates a box where you can input text, submit creates a button that submits the form when clicked.

Attribute `value`, used in the submit button was used to set the text displayed on our submit button. Its value should be a clue to the control's function. For many other controls, `value` specifies their initial value, if they are controls that accept input. If `value` was used in a `type="text"` control, it would have specified the initial value of the control.

For form elements, the `name` attribute should be used for all controls you want available for processing. This is only logical, since the name serves as the control's identity. Controls that don't have a name are not submitted.

For radio buttons and checkboxes, there is a “boolean” attribute `checked`, which when present, specifies that the control is checked. By boolean, I mean that the attribute needs (has) no value. Its presence alone does the job.

Controls which accept text input have attribute `maxlength` which places an upper limit on the number of characters that can be entered

```
<input type="text" maxlength="9" name="firstname">
<!-- maximum of 9 characters -->
```

⁵ P Hypertext Preprocessor: <http://www.php.net>

You should be aware that the third line in the code above is a comment, and is thus ignored.

```
<form method="get" action="http://localhost/mysite.php">

    <label for="name">Name</label>
    <input type="text" maxlength="6"><br>

    Password: <input type="password" maxlength="8">

    <label for="location">location</label>
    <select name="location" size="1" multiple>
        <option value="makurdi">Makurdi</option>
        <option value="abuja">Abuja</option>
        <option value="lagos">Lagos</option>
        <option value="Kano">Kano</option>
        <option value="Kaduna">Kaduna</option>
    </select>

    <input type="submit" value="submit">
    <input type="reset" value="reset">

</form>
```

Code_Sample 6

In Code_Sample 6, we use the element **LABEL** again. **LABEL** is simply a formal way of associating a particular label with a particular control. Its attribute **for** takes the name of the control that owns it. The contents of the label element serves as the descriptive text for the control. Presently, using labels does not position the text beside the control for you. You usually have to place it yourself, either by following the label specification with the control, or using tables.

```
<label for="name">Name:</label>
<input type="text" name="name">
```

There is also a password input field, which masks the password from prying eyes.

Next there is a **SELECT** element. This one creates a sort of menu of options, from which one (or more) choices can be selected. We used attribute **size** to specify the number of visible items. Other items are shown when the scrollbar is clicked. If the **size** attribute is absent, only one item is initially visible.

This element (select) also has a boolean attribute **multiple**, which permits more than one item to be selectable. Try not to use **multiple** together with **size="1"**!