Html and CSS

A brief introduction with useful examples

José Santos

November 2010

1 HTML

The basic document structure:

```
<html>
   <head>
      <title> Primeira Pagina </title>
   </head>
   <body>
      the content of the web page
   </body>
</html>
```

The purpose of HTML is to provide meaning and structure to the content. It is not intended to provide instructions for how the content should look. When marking up content the goal is to choose the most meaningful description at hand - *Semantic Markup.*
There are two kinds of *HTML elements*:

- *Block-level elements:* Browsers treat block elements as if they are in little rectangular boxes, stacked up in a page. Each block-level element begins on a new line and some space is also usually added above and below the entire element by default.
- *Inline elements* do not start new lines; they just go with the flow.

The distinction between *inline elements* and *block-level elements* is important, since inline elements cannot contain block-level elements.
Empty elements are elements that do not have text content. They are used to provide a simple directive. The image element (**img**) or the horizontal rule element (**hr**) are examples of inline elements. Empty elements are terminated by adding a trailing slash preceded by a space before the closing bracket, like:

```
<img />  <br /> <hr />
```

Attributes are instructions that clarify or modify an element. For instance, for the img element the **src** (source) attribute is required since it specifies the URL of the image. The syntax for attributes is as follows:

```
<element attribute-name = ‘‘value’’> Content </element>
```

The mark-up for comments is the following:

```
<!-- insert a comment here -->
```

Block-level elements:

- Paragraphs:

  ```
  <p> xxxxx </p>
  ```

- Headings. There are six levels of heading in HTML. Heading are used to provide logical hierarchy to the document. Therefore, it is proper to start with the Level 1 heading (**h1**) and work in numerical oreder.

  ```
  <h1> </h1>
  ```

- Long quotation:

  ```
  <blockquote> </blockquote>
  ```

- Preformatted text elemen (**pre**). The content of this element is displayed exactly as it is typed, including all carriage returns and multiple character spaces.

  ```
  <pre> ola     ola </pre>
  ```

- The address element (**address**) is used to display address information.
- There are three main types of lists in HTML:
  - Unordered list (**ul**). The contents of unordered lists as well as ordered list are list items (**li**).
  - Ordered list (**ol**). If you want a numbered list to start at a number other than 1, you can use the **start** attribute in the ol element to specify another starting number.
  - Definition lists (**dl**) are used for lists of terms with their respective definitions. The content of a dl is some number of terms (**dt**) and definitions (**dd**).

Any list can be nested within another list; it just has to be placed within a list item. The numbering style is not changed by default when you nest ordered lists. You need to set the numbering styles yourself using style sheets.

Inline text elements fall into two general categories: semantic elements and presentational elements. The semantic elements describe the meaning of the text; for example, an acronym or emphasized text. The presentational elements provide descriptions of the element's typesetting or style, such as bold, italic, or underlined. Presentational inline elements are discouraged from use in contemporary web design where style information should be kept separate from the markup. We list the following inline elements:

- Emphasizing the text: **em**, **strong**.
- Short quotations: **q**. Browsers should automatically add quotation marks around **q** elements, so you don't need to include them in the source document.
- Abbreviations and acronyms: **abbr** and **acronym**.
- Citation: **cite**.
- Defining terms: **dfn**.
- Inserted and Deleted text. The **ins** and **del** elements are used to mark up changes to the text and indicate parts of a document that have been inserted or deleted.

XHTML provides two generic elements that can be customized to perfectly describe content. The **div** (division) element is used to indicate a generic block-level element, while the **span** element is used to indicate a generic inline element. The **div** and **span** elements have no inherent presentation qualities of their own, but one can use style sheets to format the content. In fact, generic elements are a primary tool in standard-based web design because they enable authors to accurately describe content. By marking related elements as a **div** and giving it a descripitive name, you give context to elements in the grouping. The **div** element can be seen as a container to group several other elements as the example presented below illustrates:

```
<div class="listing">
  <img src="felici.gif" alt="" />
  <p> <cite>The complete manual of typography </cite>,
      James Felici </p>
  <p> A combination of type history and examples of
      good and bad type. </p>
</div>
```

Another common use of **div** is to break a page into sections for context, strucutre, and layout purposes:

```
<div id="news">
  <h1>New this week </h1>
  <p>We've been working on... </p>
  <p>And last but not least... </p>
</div>
```

A **span** offers all the benefits as the **div** element, except it is used for inline elements that do not introduce line breaks. Because **spans** are inline elements, they can only contain text and other inline elements.

```
<ul>
  <li>Mae: <span class="phone">214302377</span> </li>
  <li>Pedro: <span class="phone">915791001</span> </li>
</ul>
```

The previous examples, show that the attributes **id** and **class** are used to give names to the generic **div** and **span** statements. The **id** identifier is used to identify a unique element in the document. Thus, the value of each **id** can be used only once. The **class** attribute is used for grouping similar elements. Therefore, unlike the **id** attribute multple elements may share the same class name.
Some common characters are not part of the standard set of ASCII characters, which contains only letters, numbers and a few basic symbols. Other characters must be escaped in the source document. Escaping means that instead of typing in the character itself, you represent it by its numeric or named character reference. There are two ways of referring to a specific character: by an assigned numeric value (numeric entity) or using a predefined abbreviated name for the character. All character references begin with "&" and end with ";".

1.1 Links

The element used to specify links is the *anchor*:

```
<a href="http://users.isr.ist.utl.pt/˜jffsantos">José</a>
```

It is important to stress that anchors are inline element, so it is not possible to specify block elements between anchor tags.
There are two ways to specify the URL:

- Absolute URL: provide the full URL for the document, including the protoco (http://), the domain name and the path name as necessary.
- Relative URL: describe the pathname to the linked file realtive to the current document. It does not require protocol or domain name - just the pathname. Without the "http://", the browser looks on the current server for the linked document. Web pathnames follow the Unix convention of separating directory and filenames with forward slashes (/). A relative pathname describes how to get to the linked document starting from the location of the current document. There are two complementary ways to refer to files within the same browser:

  – One can refer to parent directories using "../".
  – All web sites have a root directory which is the directory that contains all directories and files for the site. Instead of specifying the pathname with respecto to the current file, one can specify the pathnames with respect to the root directory:

    ```
    <a href="/recipes/salmon.html">Garlic Salmon</a>
    ```

In order to link to a fragment of a web page one has to identify the destination (using the **id** attribute) and then to make a link to it appending the **id** to the pathname preceeded by the hash symbol "#".
To specify that upon pressing the link a new window should be opened, one has only to set the value of the attribute **target** to **_blank**.
One can specify a reference to an e-mail account in the following way:

```
<a href="mailto:j3fsantos@gmail.com">Contact José Santos</a>
```

1.2 Images

Images appear on web pages in two ways: as part of the inline content or as tilling background images. Background images are added using Cascading Style Sheets. Inline images may be used on web pages in several ways:

- A simple image. An image can be used on a web page page as it is used in print, as a static image that adds information.
- As a link. A image can be used as a link by placing it between anchor tags.
- As an image map. An imagemap is an image that contains multiple links.

In order to be displayed, inline images must be in the GIF, JPEG or PNG file format. In addition to being in an appropriate format, image files need to be named with the proper suffixes - *.gif*, *.jpg* or *.png*. If for some reason you want to keep an image in its original format, one can make it available as an external image by making a link directly to the image file. Images are specified using the image element: **img**. The **src** attribute tells the browser the location of the image file. The **alt** attribute provides alternative text that is displayed if the image is not. Concerning the image element there are the following issues to consider:

- It is an empty element, which means that it doesn't have any content.
- In XHTML empty elements need to be terminated.
- It is an inline element so it behaves like any other inline element in the text flow.
- The image element is what's known as a replaced element because it is replaced by an external file when the page is displayed.

- By default the bottom edge of an image aligns wiht the baseline of the text.

Image attributes:

- Long descriptions: **longdesc**. The value of the **longdesc** attribute is the URL of an external XHTML document containing the long description of the image.
- Specifying dimensions: **width** and **height**. The **width** and **height** attributes indicate the dimensions of the image in number of pixels. These attributes can speed up the time it takes to display the final page. When the browser knows the dimensions of the images on the page, it can busy itself laying out the page while the image files themselves are downloading. Without the **width** and **height** values, the page is laid out immediately and then reassembled each time an image file arrives from the server. Telling the browser how much space to hold for each image can speed up the final page display by seconds for some users. If the dimensions provided do not match the actual dimensions of the image, the browser resizes the image to match the specified size. One should not force a big download on a user when all that is needed is a small image.

1.3 Tables

HTML tables are added for instance when you need to add tabular material (data arranged into rows and columns) to a web page. The elements required to specify a table are the following:

- **table**
- **tr** Table row
- **td**. Table data
- **th** Table header

```
<table>
   <tr>
      <th> Menu Item </th>
      <th> Calories  </th>
      <th> Fat (g)   </th>
   </tr>

   <tr>
      <td> Chicken noodle soup </td>
      <td> 120                 </td>
      <td> 2                   </td>
   </tr>
   <tr>
      <td> Caeser Salad        </td>
      <td> 400                 </td>
      <td> 26                  </td>
   </tr>

</table>
```

Column spans are created with the **colspan** attribute in the **td** or **th** element. They stretch a cell to the right to span over the subsequent columns.

```
  <table>
    <tr>
      <th colspan="2">Fat</th>
    </tr>
    <tr>
      <td>Saturated</td>
      <td>Unsaturated</td>
    </tr>
  </table>
```

Row spans, created with the **rowspan** attribute, work just like column spans, except they cause the cell to span downward over several rows.

```
  <table>
    <tr>
      <th rowspan="3">Serving Size</th>
      <td>Small</td>
    </tr>
    <tr>
      <td>Medium</td>
    </tr>
    <tr>
      <td>Large</td>
    </tr>
  </table>
```

There are two kinds of space that can be added in and around table cells: *cell padding* and *cell spacing*. *Cell padding* is the amount of space held between the contents of the cell and the cell border. It is set using the attribute **cellpadding**. Because the **cellpadding** attribute may be used with the table element only, the **cellpadding** value applies to all the cells in the table. In other words, it is not possible to specify different amounts of padding for individual cells with this attribute. Cell spacing is the amount of space held between the cells, specified in number of pixels. If no value is specified, the browser will use the default value of two pixels of space between cells. Cell spacing is set using the attribute **cellspacing**.

There are two ways for providing additional information about a table: captions and summaries. Both captions and summaries are useful tools in improving table accessibility. The **caption** element is used to give a table a title or a brief description. The **summary** attribute of a table is used to provide a more lengthy description of the table and its contents. It is important to stress that while a caption is a table element, a summary is a table attribute.
There are other elements that can be used to provide table accessibility:

- The **abbr** attribute is used in a table header (th) element to provide an abbreviated version of the header to be read aloud by a screen reader.
- The **scope** attribute explicitly associates a table header with the **row**, **column**, **rowgroup**, or **colgroup** in which it appears.
- The **headers** attribute is used in the **td** element to explicitly tie it to a header.

1.4 Forms

Forms are added to web pages using (no surprise here) the **form** element. The **form** element is a container for all the content of the form, including some number of form controls, such as text entry fields and buttons. It may also contain block elements, however it may not contain another **form** element.

```
<form action="/cgi-bin/mailinglist.pl" method="post">
   <fieldset>
      <legend>Join our email list</legend>
      <ol>
        <li> <label for="name">Name: </label>
              <input type="text" name="name" id="name" /> </li>
        <li> <label for="email"> Email: </label>
              <input type="text" name="email" id="email" /> </li>
      </ol>
      <input type="submit" value="Submit" />
   </fieldset>
</form>
```

The **action** attribute provides the location (URL) of the application or script (sometimes called action page) that will be used to process the form. The **action** attribute in this example sends the data to a script called mailinglist.pl. The **method** attribute specifies how the information should be sent to the server. There are only two methods: POST and GET. When the form's method is set to POST, the browser sends a separate server request containing some special headers followed by the data. Only the server sees the content of the request, thus it is the best method for sending secure information such as credit card or other personal information. With the GET method, the encoded form data gets stacked right onto the URL sent to the server. A question mark character separates the URL from the following data:

```
get http://www.bandname.com/cgi-bin/mailinglist.pl?name=Sally&email=xxx%40example.com
```

The GET method is appropriate if you want users to be able to bookmark the results of a form submission. Because the content of the form is in plain sight. GET is not appropriate for forms with private, personal, or financial information.
Web forms use a variety of controls (also sometimes called widgets) that allow users to enter information or choose options. Control types include various text entry fields, buttons, menus and a few controls with special functions. In the previous example, the text entry fields are used to collect the visitor's email and address. "name" and "address" are two *variables* collected by the form. The data entered by the user is the value or content of the variable. The **name** attribute identifies the name of the variable for the control. It is important to stress that you cannot name the variables of your form anyway you like. The web application that processes your data is programmed to look for specific variable names.
The label of a form control and the form control itself are two separate things. There are however two different methods to tie them together:

- Insert the form control element inside the label element.
- Use the attribute **for** of the label element.

More form elements:

- To enter a large portion of text you can use the **textarea** form element instead of the **input**.
- The **fieldset** element is used to indicate a logical group of form controls. The **fieldset** element may include a **legend** element that provides a caption for the enclosed fields.
- Another option for providing choices is to put them in a pull-down or scrolling menu. Menus tend to be more compact than groups of buttons and checkboxes. The **select** element is used to specify a menu. Each option of the menu is specified using the **option** element. Options can be grouped using the **optgroup** element. In order to specify that several options may be selected you must set the attribute **multiple** of the **select** form element to **multiple**. Otherwise it will only be possible to select a single option.

Most of the inputs of a form are specified using the **input** element. The attribute **type** of the **input** element specifies what kind of input will be passed to that form control. There are several kinds of input to consider:

- **text** is used to specify a text input.
- The most fundamental form control is the **submit** button. When clicked, the **submit** button immediately sends the collected form data to the server

for processing. Note that forms that only contain one form field do not require a submit button; the data will be submitted when the user hits the Enter or Return key. A submit button must be included for all other forms. So, how can one specify a submit button? It is just an input element whose type is set to **submit**.

- An equally important form control is the reset button. It restores all the form controls to the state they were in when the form was loaded. In order to specify a reset button just set the type of the input element to **reset**.
- To specify a password field just set the type attribute of the input element to **password**.
- To specify a file entry field just set the type attribute of the input element to **file**. It is important to note that when a form contains a file selection input element, you must specify the encoding type (**enctype**) of the form as **multipart/form-data** and use the POST method.
- Instead of the select element, you can use a list of **radio** buttons or **check-boxes**. Both are specified using the input form element. The difference is that more than one checkbox may be checked at a time. Naturally, there must be a way to tie a set of radio buttons (or checkboxes) that are logically related together. This is done, specifying the same name for all radio buttons (or checkboxes) which are logically related and specifying for each one of them an unique value for the attribute **value**.

```
<fieldset>
<legend>Qual e a sua faixa etaria</legend>
<ul>
  <li> <label><input type="radio" name="faixaetaria" value="crianca" />Crianca</label> </li
  <li> <label><input type="radio" name="faixaetaria" value="adulto" />Adulto</label> </li>
  <li> <label><input type="radio" name="faixaetaria" value="velho" />Velho</label> </li>
</ul>
</fieldset>
```

Note that in this example both the button input and its text label are contained in a single label element. The advantage to this method is that users may click anywhere on the whole label to select the button.
The **button** element is a flexible element for creating custom buttons similar to those created with the input element. The content of the **button** element (text and/or images) is what gets displayed on the button. There are several types of buttons (as happened with the input form element). Again, it is important to stress the importance of the button whose type is **submit** (for obvious reasons.

```
<form id="chooser" action="chooser.php" method="post">
  <fieldset>
   <legend>Please choose a plan from the following</legend>
   <ul>
```

```
    <li><button type="submit" name="plan" value="basic">
     <h3>Basic Plan</h3>
     <p>
      You get 20<abbr title="gigabytes">GB</abbr> of
      storage and a single domain name for
      <strong>$2.99/<abbr title="month">mo</abbr></strong>
     </p>
    </button></li>
    <li><button type="submit" name="plan" value="pro">
     <h3>Pro Plan</h3>
     <p>
      You get 100<abbr title="gigabytes">GB</abbr> of
      storage and a single domain name for
      <strong>$12.99/<abbr title="month">mo</abbr></strong>
     </p>
    </button></li>
    <li><button type="submit" name="plan" value="guru">
     <h3>Guru Plan</h3>
     <p>
      You get 500<abbr title="gigabytes">GB</abbr> of
      storage and unlimited domain names for
      <strong>$22.99/<abbr title="month">mo</abbr></strong>
     </p>
    </button></li>
   </ul>
  </fieldset>
</form>
```

2 Cascading Style Sheets

2.1 Basics

A *style sheet* is made up of one or more style instructions, called *rules*, that describe how an element or a group of elements should be displayed. Each rule *selects* an element and *declares* how it should look.

```
  p { color: green; }
```

The two main sections of a *rule* are the *selector* that identifies the element or the elements to which the rule applies and the *declaration* that provides the rendering instructions. The *declaration* in turn is made up of a *property* (such as color) and a *property value* (such as green) separated by a colon and a space. If you ever need to apply the same style property to a number of elements, you can group the selectors inot one rule by separating them with commas.

```
  h1, h2, p, em, img { border: 1px solid blue; }
```

Comments in CSS are specified using the Java notation (not the XML) notation. Style information can be applied to an (X)HTML document in one of the following ways: external style sheets, embedded style sheets and inline styles.
External style sheets. An external style sheet is a separate, text-only document that contains an arbitrary number of style rules. It must be named with the *.css* suffix. External style sheets are by far the most powerful way to use CSS, because you can make style changes across an entire site simply by editing a single style sheet document. There are two ways to refer to an external style sheet: the **link** element and the **@import rule**.
The **link** element can be used in the following way:

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

The attribute **rel** defines the linked document's relation with the current document. Naturally, when linking a document to a stylesheet the value of this attribute must always be **stylesheet**. The **href** attribute provides the location of **.css** file. The **type** attribute identifies the data type of the style sheet as 'text/css' (currently the only option). You can include multiple link elements to different style sheets and they will all apply. If there are conflicts, whichever one is listed last will override previous settings due to the rule order and the cascade.
The other method for attaching external style sheets to a document is to import it with the **@import** rule in the style element, as shown in the example:

```
<head>
  <style type="text/css">
    @import url("http://path/stylesheet.css");
    p { font-face: Verdana; }
  </style>
  <title>Titles are required.</title>
</head>
```

Note that the import rule must come before any selector (it may come after other import rules however).
Colors and fancy backgrounds are nice for Web pages, but they are often a nuisance when a page is printed. There has been a common convention on the Web to provide a 'printer friendly' version for pages that are information-rich and likely to be printed. Once the media-specific style sheets are created, you must attach them to the source document and specify which style sheet is to be use for each **media**. There are two ways to do it:

- When linking to an external style sheet, set the attribute **media** of the link element to the corresponding kind of media.

  ```
  <link rel="stylesheet" type="text/css" href="cabbage.css" media="screen" />
  <link rel="stylesheet" type="text/css" href="cabbage_print.css" media="print" />
  ```

- When using the import rule inside the style element, just specify the kind of media immediately after the import rule.

```
<style type="text/css">
  @import url(cabbage.css) screen;
  @improt url(cabbage_print.css) print;
</style>
```

Embedded style sheets. An embedded style sheet is place in the document using the **style** element. The style element must be placed in the head of the document and it must contain a **type** attribute that identifies the content of the **style** element as 'text/css'.
Inline styles. You can apply properties and values to a single element using the **style** attribute in the element itself (to add multiple properties just separate them with semicolons).

```
<h1 style="color: red; margin-top: 2em">Introduction</h1>
```

Inline styles should be avoided, unless it is absolutely necessary to override styles form an embedded or external style sheet. Inline styles are problematic in that they intersperse presentation into the structural markup.
When you write a font-related rule using the **p** element as a selector, the rule applies to all of the paragraphs in the document as well as the inline text elements that they contain. This is called *inheritance.* Notice that some style properties inherit whereas others do not. In general, properties related to the styling of the text (font, size, color, style, etc) are passed down. Properties such as borders, margins, backgrounds and so on that affect the boxed area around the element tend not to be passed down. This makes perfect sense: when you put a border around a paragraph, you generally don't want that border to appear around the inline elements that are contained in that paragraph.
CSS starts with a C. This *C* stands for *Cascading*, which means that several style sheets may be applied to the same document. So conflicts between rules are bound to appear. Generally speaking, the closer a style sheet is to the content, the more weight it is given. When two rules in a single style sheet conflict, the type selector is used to determine the winner. The more specific a selector, the more weight it is given to override conflicting declarations. Finally, if there are conflicts within style rules of identical weight, whichever comes last in the list wins. Additionally, style information can come from various sources (external style sheets, embedded style sheets and inline styles), items lower in the following list will override items above them:

- Browse default settings.
- User style settings (set in a browser as a 'reader style sheet').
- Linked external style sheet (added with the **link** element).
- Imported style sheets (added with the **@import** keyword).
- Embedded style sheets (added with the **style** element).

- Inline style information (added with the **style** attribute in an openning tag).
- Any style rule marked **!important** by the author.
- Any style rule marked **!improtant** by the reader.

If you want a rule not to be overriden by a subsequent style rule just mark it as **!important**:

```
p { color: blue !important; }
```

2.1.1 Summary - How to include style sheets

You can include a style sheet:

- External style sheet: **link** tag and **@import** statement inside an embedded style sheet.
- Embedded style sheet: **style** tag.
- Inline style: set the **style** attribute of the element you wish to format.

2.2 Font Formatting

In typography, a *typeface* is a set of one or more fonts, in one or more sizes, designed with stylistic unity, each comprising a coordinated set of *glyphs*. A typeface usually comprises an alphabet of letters, numerals, and punctuation marks; it may also include ideograms and symbols, or consist entirely of them, for example, mathematical or map-making symbols.
The term *typeface* is frequently conflated with *font*; the two terms had more clearly differentiated meanings before the advent of desktop publishing. The distinction between font and typeface is that a font designates a specific member of a type family such as roman, boldface, or italic type, while typeface designates a consistent visual appearance or style which can be a ‘family’ or related set of fonts. For example, a given typeface such as Arial may include roman, bold, and italic fonts.
As the range of *typeface* designs increased and requirements of publishers broadened over the centuries, *fonts* of specific *weight* (blackness or lightness) and *stylistic variants* (most commonly regular or roman as distinct to italic, as well as condensed) have led to *font families*, collections of closely related typeface designs that can include hundreds of styles. A *font family* is typically a group of related fonts which vary only in weight, orientation, width, etc., but not design. For example, Times is a font family, whereas Times Roman, Times Italic and Times Bold are individual fonts making up the Times family. Font families typically include several fonts, though some, such as Helvetica, may consist of dozens of fonts.
Digital fonts store the image of each character either as a *bitmap* in a *bitmap font*, or by mathematical description of lines and curves in an *outline font*, also

called a *vector font*. When an outline font is used, a *rasterizing routine* (in the application software, operating system or printer) renders the character outlines, interpreting the vector instructions to decide which pixels should be black and which ones white. Rasterization is straightforward at high resolutions such as those used by laser printers and in high-end publishing systems. For computer screens, where each individual pixel can mean the difference between legible and illegible characters, some digital fonts use hinting algorithms to make readable bitmaps at small sizes.
Most scripts share the notion of a *baseline*: an imaginary horizontal line on which characters rest. In some scripts, parts of glyphs lie below the baseline. The *descent* spans the distance between the baseline and the lowest descending glyph in a typeface, and the part of a glyph that descends below the baseline has the name *descender*. Conversely, the *ascent* spans the distance between the baseline and the top of the glyph that reaches farthest from the baseline. The ascent and descent may or may not include distance added by accents or diacritical marks.

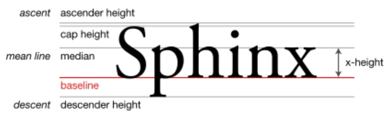

Figure 1: Typeface metrics

Typefaces with the same metrics (i.e., with the same glyph dimensions) are said to be *metric-compatible*, that is, they can be substituted for one another in a document without changing the document's text flow. Several typefaces have been created to be metric-compatible with widely used proprietary typefaces to allow the editing of documents set in such typefaces in digital typesetting environments where these typefaces are not available. For instance, the open source Liberation fonts have been designed as metric-compatible substitutes for widely used Microsoft fonts.
The nature of the Web makes specifying type tricky, if not downrigth frustrating because you have no way of knowing which fonts are loaded on the user's machine (you cannot be sure that everyone will see the text in the font you have chosen). And because the default font size varies by browser and user preferences, you cannot be sure how large or small the type will appear as well.
The font related properties are the following:

- **font-family**
- **font-size**
- **font-weight**
- **font-style**
- **font-variant**

- **font**

All **font-family** names, with the exception of generic font families, must be capitalized. Use commas to separate multiple font names. You should specify multiple font-family names because you do not know which fonts are installed in the user's pc. So you just specify several. Another approach is to specify a *generic font family* and the browser chooses one font among the available fonts within that font family. The generic font families are:

- **serif**
  Examples: **Times**, **Times New Roman**, **Georgia**
  Serif typefaces have decorative serifs, or slab-like appendages, on the ends of certain letter strokes.

- **sans serif**
  Examples: **Arial**, **Arial Black**, **Verdana**, **Trebuchet MS**, **Helvetica**, **Geneva**
  Sans-serif typefaces have straight letter strokes that do not end in serifs. They are generally considered easier to read on computer monitors. It is commonly believed that, in contrast to the case for printed material, sans serif fonts are easier than serif fonts to read on the low-resolution computer screen.

- **monospace**
  Examples: **Courier**, **Courier New**, **Andale Mono**
  In monospace typefaces all characters take up the same amount of space on a line. Monospaced typefaces function better for some purposes because their glyphs line up in neat, regular columns. So, program editors tend to use this kind of font. The **pre** HTML tag most commonly specify monospaced fonts.

- **cursive**
  Examples: **Comic Sans**
  Cursive or script typefaces simulate handwriting or calligraphy. They do not lend themselves to quantities of body text, as people find them harder to read than many serif and sans-serif typefaces; they are typically used for logos or invitations.

- **fantasy**
  Ornamental (also known as novelty or sometimes display) typefaces are used exclusively for decorative purposes, and are not suitable for body text. They have the most distinctive designs of all fonts, and may even incorporate pictures of objects, animals, etc. into the character designs.

The best practice for specifying fonts for web pages is to start with your first choice, provide some similar alternatives, then end with a generic font family that at least gets users in the right stylistic ballpark:

```
body { font-family: Verdana, Arial, Helvetica, sans-serif; }
```

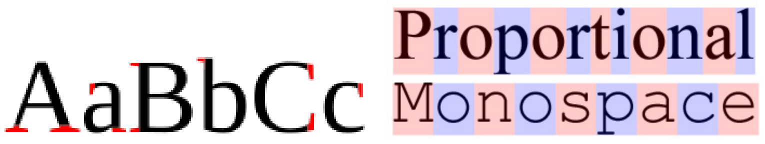

Figure 2: font types

To specify the **font-size** one procceeds as follows:

```
h1 { font-size: 150%; }
```

You can specify the font-size in several ways:

- A specific size using one of the CSS units: **em** (unit of measurement equal to the current font size), **px** (considered relative because it varies with display resolution).
- A percentage value, sized up or down from the element's default or inherited size.
- Using one of the absolute keywords (**xx-small**, **x-small**, **small**, **medium**, **large**, **x-large**, **xx-large**). On most current browsers **medium** corresponds to the default font-size.
- Using a relative keyword (**larger** or **smaller**).

In contemporary design the acceptable ways to specify the font-size are **em** measurements, percentage values and keywords. These are preferred because they allow the users to resize text using the text-zoom feature on their browsers. The benefict of the keywords is that browsers will never let text sized in keywords to render smaller than 9 pixels, so they protect against illegible text. On the downside, the size keywords are imprecise and unpredictable. One popular method for font sizing text is to make the text smaller globally using the body element with a percentage value and then size all elements up using em measures.
If you want a text to appear in bold, use the **font-weight** property to adjust the boldness of the type. Because most fonts on the Web have only two weights, norma (or roman) and bold, the only font-weight value you will use in most cases is **bold**.
The **font-style** property affects the posture of the text, that is, whetherthe letter shapes are vertical (normal) or slanted (italic and oblique). Property values: **normal**, **italic**, **oblique**.
The **font-variant** allows designers to specify **small-caps** font for text elements.
Of course there is a **font** property shortcut, the order of the values is as follows:

```
{ font : style weight variant size/line-height font-family }
```

At minimum, the **font** property must include a **font-size** value and a **font-family** value specified in that order.

2.3 Text and line adjustments

The **line-height** property defines the minimum distance from baseline to baseline in text. It is said to specify a minimum distance because if you put a tall image on a line, the height of the line will expand to accomodate it. Line heights can be specified in one of the CSS length units but once again the relative **em** is the best option.
The **text-indent** property indents the text by a specified amount. The **text-align** property allows you align your text in several different ways: **left**, **right**, **center**, **justify**. If you want to put a line under, over, or through text, or if you would like to turn the underline off under links, then the **text-decoration** property is the one that you seek. You can set the **text-decoration** property the following values: **none**, **underline**, **line-through**, **blink** and **inherit**. The most popular use of the **text-decoration** property is turning off the underlines that appear automatically under linked text:

```
  a { text-decoration: none; }
```

To capitalize the text you may use the property **text-transform** which can have the following values **none**, **capitalize**, **lowercase**, **uppercase**.
The final two text properties here presented are **letter-spacing** and **word-spacing** which are used to insert space between letters and words (respectively). The spacing must be specified in one of HTML measurement units.

2.3.1 Summary - Text formatting properties

- **line-height**: minimum distance from baseline to baseline.
- **text-indent**
- **text-align**: **left**, **right**, **center**, **justify**
- **text-decoration**: **underline**, **line-through**, **blink**, **inherit**, **none**
- **text-transform**: **capitalize**, **lowercase**, **uppercase**
- **word-spacing**
- **letter-spacing**

2.4 Color

There are two main ways to specify colors in style sheets: with a predefined color name or with a numeric value that describes a particular RGB color.

```
color: red;
color: olive;
color: #FF0000;
color: #808000;            // hexadecimal format
color: rgb(200, 178, 230);  // decimal format
color: rgb(78%, 70%, 90%);  // percentage
```

CSS properties for spcifying colors:

- **color:** specify the foreground color of an element. The foreground of an element consists of its text and border.
- **background-color:** apply background-colors to any element. A background color fills the canvas behind the element that includes the content area, and any padding added around the content, extending behind the border out to its outer edge. Background colors do not inherit, but because the default background setting for all elements is **transparent**, the parent's background color shows throw its descendants.

```
.glossary {
            color: #7C3306;
            background-color: #F2F288;
          }
```

2.5 Selectors

- Element type selector:

  ```
  p { property: value; }
  ```

- Grouped selectors:

  ```
  p, h1, h2 {property: value; }
  ```

- Id selector:

  ```
  #sidebar {property: value; }
  ```

- Class selector:

  ```
  .warning {property: value;}
  ```

- Universal selector:

  ```
  * {property: value;}
  ```

There is a special kind of selectors, called *pseudoclass selectors*, which are particularly useful to handle anchors:

```
a:link              // Applies a style to unclicked links
a:visisted          // Applies a style to links that have already been clicked
a:hover             // Applies a style when the mouse pointer is over the link
a:active            // Applies a style while the mouse button is pressed
```

Pseudoclass selectors are indicated by the colon (:) character.
Other pseudclass selectors:

- **:focus** - targets elements that have focus, such as a form element that is highlighted and accepting user input.

Aside from pseudo class selectors, we also have *pseudoelement* selectors. Pseudoelement selectors are also indicated with a colon (:) symbol. We list several ones:

- **:first-child** - targets an element that is the first occurring child of a parent element such as the the first **p** in a **div**.
- **:first-line**
- **:first-letter**

There are also attribute selectors:

- Simple attribute selectors (**element[attribute]**): targets elements with a particular attribute regardless of its value;
- Exact attribute value (**element[attribute="exact value"]**): selects elements with a specific value for the attribute;
- Partial attribute value (**element[attribute ="value"]**): allows you to specify one part of an attribute value;
- Hyphen-separated attribute value (**element[attribute—="value"]**): targets hypen-separated values.

Examples:

```
img[title] {border: 3px solid;}
img[title="first grade"] {border: 3px solid;}
img[title~="grade"] {border: 3px solid;} // Looks for the word grade in the title
*[hreflang|="es"] // selects any element that specifies a variation on the spanish languag
```

3 Background

- **background-image:** used to add a background image to an element. Default values: **none**, **inherit**. Like background colors, tiling background images fill the area behind the content area, the extra padding and extend to the outer edge of the border. If you provide a **background-color** and a **background-image** to an element, the image will be placed on top of the color.
- **background-repeat:** By default images tile up and down, left and right. You can change this however. Values: **repeat**, **repeat-x**, **repeat-y**, **no-repeat** and **inherit**.

- **background-position:** specifies the position of the original image in the background. In general, you specify both horizontal and vertical values that describe where to place the origin image, but there are a variety of ways to do it:
  - **Keyword positioning:** The keywords **left**, **right**, **top**, **bottom**, **center** are used to specify the position of the image. They are generally used in pairs, if you provide only one the missing keyword is assumed to be center.
  - **Lenght measurements:** You can also specify the position by its distance from the top-left corner of the element using pixel measurements. When providing lenght values the horizontal measurement always goes first.
  - **Percentages:** As with keywords, if you only provide one percentage, the other is assumed to be 50%.
- **background-attachment**: With this property you have to choose if the background image scrolls or is fixed. Values: **scroll**, **fixed**, **inherit**.
- **background**: you can use the shorthand background property. The order of the properties is the following: background-color, background-image, background-repeat, background-position and background-attachment.

```
body {
   background-color: black;
   background-image: url(alro.jpg);
   background-repeat: no-repeat;
   background-position: center center;
   background-attachment: scroll;
}

body {background: black url(alro.jpg) no-repeat center center scroll;}
```

3.1 The Box model

According to the box model, every element in a document generates a box to which properties such as width, height, padding, borders and margins can be applied. Thus, for each element (be it a block-level element or an inline element), the browser generates a box which contains it:

- *content area:* at the core of the element box is the content itself.
- *inner edges:* the edges that surround the content area.
- *padding:* the padding is the area between the content area and an optional border.
- *border:* the border is a line that surrounds the element and its padding.

- *margin:* the margin is an optional amount of space added on the outside of the border.
- *outer edge:* the outside edges of the margin are make up the outer edges of the element box.

Use the **width** and **height** properties to specify the width and heigth of the content area. You can only specify the width and heigth of block-level elements and non-text inline elements such as images. By defautl, the width and heigth of a block-level element is automatically calculated by the browser. It will be as wide as the browser window or other containing block element, and as high as necessary to fit the content. Note that the width and heigth values do not include padding, margins and border.
When an element is set to a size that is to small for its contents, it is possible to specify what to do with the content that doesn't fit, using the **overflow** property. Possible values: **visible**, **hidden**, **scroll**, **auto** and **inherit**. The meaning of each one is obvious (the **auto** value allows the browser to decide).
Padding is the space between the content area and the border (or the place the border would be if one isn't specified). It's a good idea to add a little padding to elements when using a background color or a border. You can add padding to the individual sides of any element (block-level or inline). There is also a shorthand **padding** property that lets you add padding on all sides at once:

- **padding-top**
- **padding-right**
- **padding-bottom**
- **padding-left**

When using the shorthand **padding** property keep in mind the TRouBLe mnemonic.

```
{ padding: top right bottom left; }
```

If you specify only three values, they are interpreted in the following way:

```
{ padding: top right/left bottom; }
```

If you specfiy two values:

```
{ padding: top/bottom right/left; }
```

A border is simply a line drawn around the content area and its (optional) padding. You can choose from eight border styles and make them any width and color you like. You can apply the border all around the element or just a particular side or sides.
The style is the most important of the border properties because, according to the CSS specification, if there is no border style specified, the border not exist:

- **border-top-style**
- **border-right-style**
- **border-bottom-style**
- **border-left-style**

Naturally, you can use the shorthand property **border-style**.
To specify the width of the border:

- **border-top-width**
- **border-right-width**
- **border-bottom-width**
- **border-left-width**

You can also use the shorthand property **border-width**.
Border colors are specified in the same way: using the side-specific properties or the **border-color** shorthand property:

- **border-top-color**
- **border-right-color**
- **border-bottom-color**
- **border-left-color**

To specify the style, width and color of the border at the same time use the shorthand property **border**:

```
{border: border-style border-width border-color;}
```

The properties **border-top**, **border-right**, **border-bottom** and **border-left** work in the same way.
The last remaining component of an element is its margin, which is an optional amount of sapce that you add on the outside of the border. Margins keep elements from bumping into one another or the edge of the browser window. The specific and shorthand margin properties work much like the padding introduced above:

- **margin-top**
- **margin-right**
- **margin-bottom**
- **margin-left**

The shorthand property is **margin-width**.
The most significant hehavior to be aware of is that the top and bottom margins of neighboring elements collapse. This means that instead of accumulating, adjacent margins overlap, and only the largest value will be used.
You can specify top and bottom margins to inline elements but they will not appear. Conversely, if you specify left and right margins to inline elements, they will appear. For replaced inline elements (such as images) all the margins appear.
You can specify negative margins. If you do this, the neighboring element will overlap the element which has the negative margin.
Not all XML languages assign default display behaviors for the languages they contain. For this reason, the **display** property was created to allow authors to specify how elements should behave in layouts.

- **inline**
- **block**
- **none** - removes the content and the space it would have occupied is closed up. Note that if you you set the property **visibility** to hidden, it will prevent the browser from showing the element but corresponding space in the layout will not be closed up.
- **inherit**

To put it more formally, the display property defines the type of element box an element generates in the layout. The W3C discourages the random reassignment of display roles for HTML elements. However, in some situations it is commonly done. For instance, it is common practice to make **li** elements display as inline elements to make an horizontal navigation bar.

4 Floating and Positioning

In the CSS layout model, text elements are laid out from top to bottom in the order in which they appear in the source, and from left to right. Block elements stack up on top of one another an fill the available width of the browser window or other containing element. Inline elements and text characters line up next to one another to fill the block elements. When the windown or containing element is resized, the block elements expand or contract to the new width, and the inline content reflows to fit it.
Simply stated, the **float** property moves an element as far as possible to the left or right, allowing the following content to wrap around it. Possible values: **left**, **right**, **none** and **inherit**.

```
img {
  float: right;
  margin: 10px;
}
```

Some remarks are in order:

- A floated element is like an island in a stream. They are not in the flow, but the stream has to flow around them.
- Floats stay in the content area of the containing element.
- Margins are maintained.

You can also float non-replaced inline elements. However, when doing this, you must always provide a width for the floated text element. Additionally, keep in mind that floated inline elements behave as block elements.
You must provide a width for floated block elements. If you do not provide a width value, the width of the floated block will be set to auto, which fills the available width of the browser window or other containing element.
Floated elements will be placed as far left or right (as specified) and as high up as space allows. However, every floated element will stay below any element that precede it in the flow.
Note that the other elements, those that are not floated, are placed on the flow of the document and they surround the floated elements, but they do not obey any constraint. Therefore, a non-floated element that occurs after a floated one can be displayed above it. To avoid this behavior, we can set the **clear** property of the non-floated element. Applying the **clear** property to an element prevents it from appearing next to a floated element and forces it to start against the next available clear space below the float. The possible values are **left**, **right**, **both**, **none** and **inherit**. The left value starts the element below any elements that have been floated to the left. Similarly, the right vaule starts the element below any elements that have been floated to the right.

```
img {
  float: left;
  margin-right: 10px;
}

p {clear: both;}
```

CSS provides several methods for postioning elements on the page. They can be positioned relative to where they would normally appear in the flow, or removed from the flow altogether and placed at a particular spot on the page. You can also position an element relative to the browser window (technically known as the **viewport**) and it will stay put while the rest of the page scrolls.
The **position** property indicates that an element is to be poisitioned, and specifies which positioning method should be used.

- **static** - this is the normal positioning scheme in which elements are positioned as they occur in the normal document flow.
- **relative** - Relative positioning moves the box relative to its original position in the flow. The distinctive behavior of relative positionaing is that

the space the element would have occupied in the normal flow is preserved. Overlap happens.

- **absolute** - Absolutely positioned elementes are removed from the document flow entirely and positioned relative to the nearest positioned containing element. If the positioned element is not contained within another positioned element, then it will be placed relative to the document. But if the element has an ancestor that has its position set to **relative**, **absolute**, or **fixed**, the element will be positioned relative to the edges of that element. Unlike relative positioned elements, the space they would have occupied is closed up. In fact, they have no influence at all on the layout of surrounding elements. Absolutely positioned elements always behave as block level elements (for example, the margins on all sides are maintained).

- **fixed** - The distinguished characteristic of fixed positioning is that the element stays in one position in the window even when the document scrolls. Fixed elements are removed from the document flow and positioned relative to the browser window (or viewport) rather than another element in the document.

The **position** property indicates the type of position of a given element, whether the properties **top**, **right**, **bottom**, **left** are used to actually specify the position of the element. The values provided for each of the offset properties defines the distance the element should be moved away from that respective edge. A positive value for top results in the element box moving down. Similarly, a positive value for left would move the positioned element to the right. The offset values apply to the outer edges of the element box.
Absolutely positioned elements may overlap other elements, so it follows that multiple positioned elements have the potential to stack up on one another. By default, elements stack up in the order in which they appear, but you can change the **z-index** property. The value of the **z-index** property is a number, the higher the number, the higher the element will appear in the stack.

4.1 Page Layout Strategies

Over time, three general page layout approaches have emerged:

- **Liquid layout:** liquid pages resize along with the browser window. There is no attempt to control the width of the content or line breaks, the text is permitted to reflow as required.
  - Advantages: you don't have to design for a specific monitor resolution. You avoid potentially akward empty space.
  - Disadvantages: on large monitors, line lengths can geet very long and uncomfortable to read (rule of the thumb: 12 words, 60-75 characters). You can use the **max-width** property to tackle this problem.

They are less predictable. You also need to decide where the fixed-width layout will be positioned in the browser window. By default, it stays on the left edge of the browser, with the extra space to the right of it.

Create liquid layouts by specifying widths in percentages.

```
div#main {
   width: 70%;
   margin-right: 5%;
   float: left;
   background: yellow;
}

 div#extras {
   width: 25%;
   margin-left: 5%;
   float: left;
   background: orange;
}
```

- **Fixed layout:** fixed pages put the content in a page area that stays a specific pixel width regardless of the browser dimensions. First, you need to pick a page width usually based on common monitor resolutions (one of the most common: 800*600).

  – Advantages. It is predictable. It offers better control over line length.

  – Disadvantages. Content on the right edge will be hidden if the browser window is smaller than the edge. Text elements will reflow if the user resizes the font size, so the layout can vary. Line lengths can grow awkwardly short at very large text sizes.

  Fixed-width layouts are created by specifying width values in pixel units. Typically, the content of the entire page is put into a div that can be set to a specific pixel width. This div may also be centered in the browser window. Widths of column elements, and even margins and padding, are also specified in pixels.

```
#wrapper {
   width: 750px;
   position: absolute;
   margin-left: auto;
   margin-right: auto;
   border: 1px solid black;
   padding: 0px;
}
```

```
#extras {
   position: absolute;
   top: 0px;
   left: 0px;
   width: 200px;
   background-color: orange;
}

#main {
   margin-left: 225px;
   background-color: yellow;
}
```

- **Elastic layout:** elastic pages have areas that get larger or smaller when the text is resized. Elastic layouts are creates by specifying widths in em units.

4.1.1 Multicolumn Layouts

The most popular way to create a column is to float an element to one side and let the remaining content wrap around it. A wide margin is used to keep the area around the floated column clear (we recall that the default value of the postion property is static). One of the advantages of floats is that it is easier to start page elements such as a footer below the columned area of the page. The drawback of float-based layouts is that they are depedent on the order in which the elements appear in the source.
Two-column with footer - Method: float - Layout: liquid

```
#header {
  background: #CCC;
  padding: 15px; }

#main {
  background-color: aqua;
  float: left;
  width: 60%;
  margin-right: 5%;
  margin-left: 5%; }

#footer {
  clear: left;
  padding: 15px;
  background: #CCC; }

#extras {
 margin-right: 5%}
```

```
body {
 font-family: verdana, sans-serif;
 margin: 0;
 padding: 0; }

li {
  list-style-type: none,
  margin: 10px 0;
}
```

Three-column with footer - Method: float - Layout: fixed

```
#container {
  width: 750px;
  border: solid 1px;
}

#header {
  background: #CCC;
  padding: 15px; }

#links {
  background-color: fuchsia;
  float: left;
  width: 175px; }

#main {
  background-color: aqua;
  float: left;
  width: 400px; }

#news {
  background-color: green;
  float: left;
  width: 175px;
}

#footer {
  clear: both;
  padding: 15px;
  background: #CCC; }

body {
 font-family: verdana, sans-serif;
 font-size: small;
 margin: 0;
 padding: 0; }
```

Absolute positioning can also be used to create a multicolumn page. The advantage is that the order of the the source document is not as critical as it is in the float method, because element boxes can be positioned anywhere. The disadvantage is that you run the risk of elements overlapping and content being obscured. When working with absolute positioning, remember that every element you position is removed from the normal flow. If content you expect to be at the bottom of the page is sitting at the very top, it's because you positioned all the elements above it that were holding it down.

Two-column with narrow footer - Method: positioned - Layout: liquid

```
#header {
   height: 70px;
   padding: 15px;
   background: #CCC;  }

#main {
   margin-right: 30%;
   margin-left: 5%;
}

#extras {
   position: absolute;
   top: 100px;
   right: 0px;
   width: 25%;
   background: green;
   padding: 10px;}

#footer {
   margin-right: 30%;
   margin-left: 5%;
   padding: 15px;
   background: #666;}
```

Three-column with narrow footer - Method: positioned - Layout: liquid

```
#header {
   height: 70px;
   padding: 15px;
   background: #CCC;  }

#main {
   margin-right: 25%;
   margin-left: 25%;
}

#links {
   position: absolute;
   top: 100px;
   left: 0px;
   width: 22%;
   background: fuchsia;}

#news {
   position: absolute;
   top: 100px;
   right: 0px;
   width: 22%;
   background: green;}

#footer {
   margin-right: 25%;
   margin-left: 25%;
   padding: 15px;
   background: #666;}
```

Three-column page - Method: positioned - Layout: fixed

```
#container {
  position: relative;
  width: 750px;
}

#header {
  height: 70px;
  padding: 15px; /* total height 100=70+15+15 */
  background: #CCC; }

#main {
  position: absolute;
  top: 100px;
  left: 150px;
  width: 428px; /* 428+2*1+10*2=450 */
  border-left: solid 1px black;
  border-rigth: solid 1px black;
  padding: 0px 10px;
  background-color: aqua; }

#links {
  position: absolute;
  top: 100px;
  left: 0px;
  width: 134px; /* 134+8*2=150 */
  padding: 0px 8px;
  background: fuchsia; }

#news {
  position: absolute;
  top: 100px;
  right: 0px;
  width: 134px; /* 134+8*2=150 */
  padding: 0px 8px;
  background: green; }

body {
 font-family: verdana, sans-serif;
 font-size: small;
 margin: 0;
 padding: 0; }
```

Adding color to columns is an effective way to further emphasize the division of information and bring a little color to the page. However, adding background colors to the column elements may raise some technical issues that have to be tackled. Namely, the color stops at the end of the element box and does not extend to the bottom of the page. When using a fixed layout this problem can be solved using a background image:

```
#container {
  width: 750px;
  border: solid 1px;
  background-image: url(2col_bckgd.gif);
  background-repeat: repeat-y;
}
```

In CSS, the proper way to center a fixed-width element, is to specify a width for the div element that holds all the page's contents, then set the left and right margins to **auto**.

```
#container {
  position: relative;
  width: 750px;
  margin-right: auto;
  margin-left: auto;
}
```

An alternative method uses negative margins:

```
#container {
  position: absolute;
  width: 750px;
  left: 50%;
  margin-left: -325px;
}
```

4.2 Style properties for tables

CSS provides two methods for displaying borders between table cells: spearated or collapsed. When borders are separated, a border is drawn on all four sides of each cell and you can specify the space between the borders. The **border-collapse** prorperty allows authors to choose between these rendering techniques. The possible values are **separate**, **collapse** and **inherit**.
Table render with separated borders by default. You can specify the amount of space you'd like to appear between cells using the **border-spacing** property. The values for **border-spacing** are two length measurements: distance between columns (first) and distance between rows (second). If you only provide one value it will be used horizontally and vertically. The default value is 0, so when no value is specified the rows become twice as big.

```
table {
  border-collapse: separate;
  border-spacing: 15px 3px;
  border: none; /* no border around the table itself */
}

td {
  border: 2px solid purple; /* borders around the cells */
}
```

When the collapsed border model is chosen, only one border appears between the cells.

```
table {
  border-collapse: collapse;
  border: none;
}

td {
  border: 2px solid purple;
}
```

For tables with separated borders, you can decide whether you want empty cells to display their backgrounds and borders using the **empty-cells** property. The possible values are: **show**, **hide** and **inherit**.

```
table {
  border-collapse: separate;
  border-spacing: 15px 3px;
  empty-cells: hide;
  border: none;
}

td {border: 1px solid purple;}
```

4.3 Styling Lists

Using the **list-style-type** property to select the type of marker that appears before each list item. There are several types of icons available:

- **none**
- **disc**
- **circ**
- **square**
- **decimal**
- ...

The default is **disc**. Bullet size changes with the font size.
By default, the marker hangs outside the content area for the list item, displaying as a hanging indent. The **list-style-position** allows you to pull the bullet inside the content area so it runs into the list content. Therefore, the possible values are: **inside**, **outside** and **inherit**.

```
li {background-color: #F99;}
ul#outside {list-style-position: outside;}
ul#inside {list-style-position: inside;}
```

In order to specify you own bullets, use the **list-style-image**. As imagined, the default values are **none** and an URL (use the function **url**).

```
ul {
  list-style-image: url(/images/happy.gif);
  list-style-type: disc;
  list-style-position: outside; }
```

These three list properties can be set together using the shorthand property **list-style**. The values of these property may provided in any order.

```
ul { list-style: url(/images/happy.gif) disc outside; }
```

There are two main techniques for styling a list as a navigation bar:

- **Inline list items.** This technique uses the **display** property to make the list items behave as inline elements instead of block elements. As a result, they line up next to one another, instead of each starting on a new line.

- **Floated list items.** To apply this technique follow the steps below:
  - Turn off the bullet (**list-stlye: none;**).
  - Float each list item (**float: left;**).
  - Make the anchor elements display as block elements so dimensions may be set (**display: block;**).
  - Format the links with various styles.

```
Inline List Items:

ul#nav {
  list-style-type: none;
  margin: 0px;
  padding: 0px;
}

ul#nav li {
  display: inline;
}

ul#nav li a {
   padding: 5px 20px;
   margin: 0px 2px;
   border: 1px solid #FC6A08;
   text-decoration: none;
   text-align: center;
   color: black;
}

Floated List Items:

ul#nav {
  list-style-type: none;
  margin: 0px;
  padding: 0px;
}

ul#nav li {
  float: left;
  margin: 0 2px;
  padding: 0;
}

ul#nav li a {
   display: block;
   width: 100px;
   height: 28px;
   line-height: 28px;
   background: #E3A7CA url(tab.gif) no-repeat;
   text-decoration: none;
   text-transform: uppercase;
   text-align: center;
}
```

4.4 Design Tips

Limit the number of colors you use on a page. Nothing creates visual chaos faster than too many colors. Choose one dominant color and one highlight color and use a couple of shades of each one. But do not use too many colors.

When specifying a foreground and a background color make sure there is a contrast. People tend to prefer reading dark text on very light backgrounds online.

If you choose to use an image as the background of your page, choose a simple image that won't interfere with the legibility of the text over it. Always provide a **background-color** value that matches the color of the **background-image**. If the background-image fails to display, at least the overall design of the page will be similar. This is particularly important if the text color is illegible against the browser's default white background. Obviously, keep the size of background images as small as possible.