



Part 2 -- Cascading Style Sheets

A cascading style sheet allows you to add style --or what is often called *graphic design*-- to the content of your XHTML document. Cascading style sheets allow you to specify the following characteristics of elements in a document:

- **Fonts** (also called *typefaces*) including size, italicization, or boldness
- **Colors:**
 - **"Foreground" colors**, which mean the colors of text elements
 - **"Background" colors**, which mean the colors of the backgrounds "behind" text or other elements
 - **Border colors** of lines drawn around rectangular elements
- **Positions** of elements on a page, including:
 - **Absolute positions** of elements
 - **Relative positions** of elements
- **Sizes (widths and heights)** of elements:
 - **The width and height** of elements
 - **The width of padding** around elements
 - **The width of margins** around elements
- **Background Images**, both tiled and non-repeating images.

2.1. The Box Model

To understand CSS, you have to first learn about the *box model*. Every element in an XHTML document, whether text, an image, or some other element, can be enclosed by a rectangle or *box*. The rectangle itself has properties, such as line width, padding, margin, border color, and background color, as shown in **fig. 1**.

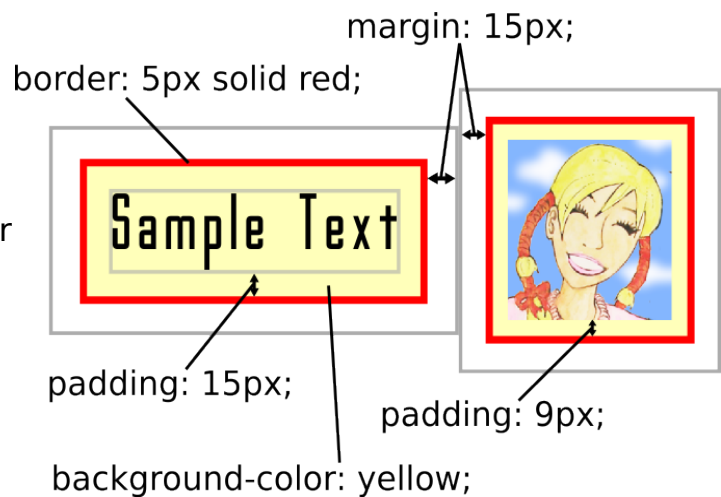


Fig. 1. The CSS box model.

2.1.1. Borders

A line can be drawn around a rectangle by specifying a *border*, such as "`border: 5px solid red;`" as shown in **fig. 1**. Here "`px`" means "pixels". **Notice that there is no space between the number and the abbreviation "`px`".** If you don't want a border, you would specify "`border: none;`". When you begin to design a CSS layout, we recommend having a visible border so that you can understand how the margin and padding attributes interact. You can later change the border to *none* if that is what you need for your final design. **Fig. 2** shows a several other border styles.

`border: 2px solid gray;`



`border: 2px dashed orange;`



`border: 2px dotted blue;`

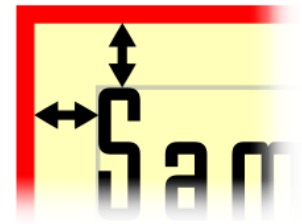


Fig. 2. Several border styles.

2.1.2. Padding

Padding refers to the distance *on the inside* from the border of the rectangle to the edge of the html element itself.

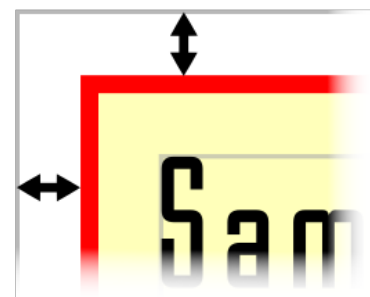
In the case of an image, the padding is simply the distance from the border of the rectangle inwards to the edge of the image itself. In the picture of Hexxa shown in **fig. 1**, the padding of 9 pixels makes the yellow background color visible, as if the picture were framed with a yellow mat. If you did not want to have a mat around the picture, you would simply specify "`padding:0;`".



In the case of a line of text or a paragraph, you must imagine that the text is enclosed by a rectangle which stretches as high as the tallest letters and as low as the longest descenders. In **fig. 1**, you can see that this rectangle (*shown in gray*) stretches as high as the tall letters **S**, lower case **L**, and **T**, and as low as the descender (tail) on the lower case **P**. The red border is then "pushed away" from the text by the padding distance.

2.1.3. Margins

Margins refer to a separation distance *on the outside* of the rectangle. Margins determine how close one element can approach another element. For example, as shown in the figure above, both the sample text and the image have margins of 15 pixels around them. Therefore, the right edge of the sample text's border will be separated from the left edge of the image's border by exactly 30 pixels. Think of margins as *pushing elements away from each other* by a specified distance.

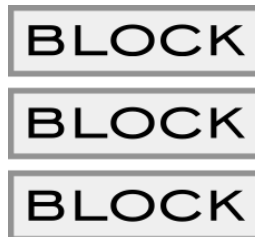


2.2. Inline versus Block Display

All HTML elements are assigned a *display* property of either *inline* or *block*. Inline elements line up horizontally, like this:



Block elements stack one on top of the other as blocks do, like this:



For example, paragraphs are *block* elements because paragraphs of text stack up vertically. In contrast, individual words in a paragraph are *inline*, because words line up horizontally in a row.

2.3. Making a menu from anchor tags

Let's take a look at the `<a>` anchor tag and how its *display* property affects the appearance on a web page. The `<a>` tag is *inline* by default. So, without any CSS, the following hypertext links:

```
<a href="#link_1">Link One</a>  
<a href="#link_2">Link Two</a>  
<a href="#link_3">Link Three</a>
```

... will line up horizontally like this on a web page:

[Link One](#) [Link Two](#) [Link Three](#)

... because `<a>` tags are *inline* by default. But we can change that by putting the following definition into our CSS style sheet:

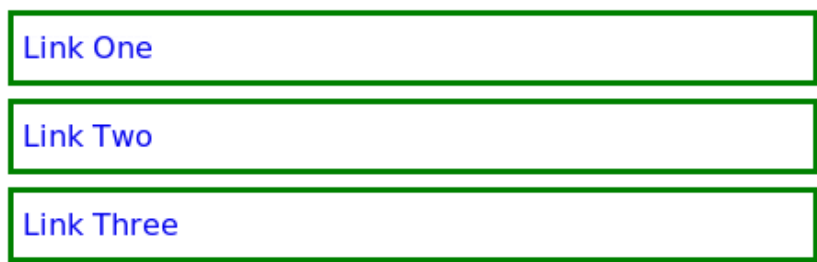
```
a{  
  display:block;  
}
```

The links will now look like a vertical navigation menu:

[Link One](#)
[Link Two](#)
[Link Three](#)

Now we can start to style our links. First, we can get rid of the default "text-decoration" responsible for the underline. Then, we can add a border, some padding, and a margin:

```
a{
  display:block;
  text-decoration:none;
  border:2px solid green;
  padding:5px;
  margin:2px;
}
```



Here the rectangular boxes fill the full width of the browser, probably not what we want. We can fix this by specifying a fixed width. Finally, we can also change the "color" and "background-color" and make the "font-weight" bold so that the letters stand out more:

```
a{
  display:block;
  text-decoration:none;
  border:2px solid green;
  padding:5px;
  margin:2px;
  width:150px;
  color:white;
  background-color:orange;
  font-weight:bold;
}
```



With just a few lines of CSS code, we have now changed plain "<a>" anchor elements into a professional-looking menu. But we are not quite done yet. To finish off the links, let's reverse the foreground and background colors when the user hovers over the link by creating a new entry for the "a:hover" properties:

```
a:hover{
  color:orange;
  background-color:white;
}
```




Now we have a truly professional result! The key to making it all work was to first change the anchor tag's default *display* property from *inline* to *block*.

2.4. Putting a picture frame around an image

Suppose we have the following snippet of XHTML consisting of a paragraph with a nested image:

```
<p>  
  This is a picture of Hexxa:  
    
  I took this picture at the fair.  
</p>
```

According to XHTML, the `` image tag is inline by default. This means that the image will simply line up in a horizontal row with the text in the paragraph:

This is a picture of Hexxa:  I took this picture at the fair.

If the picture is very small, this may be the result that we want. But many times the picture is large and a block display would be more suitable. So, just as we did with the anchor tag, let's make images display in a block format:

```
img{  
  display:block;  
}
```

This is a picture of Hexxa:



I took this picture at the fair.

Now let's define padding, a margin, and a border just as we did with the anchor tag. But this time, instead of defining a single color for the border, let's make the the right and bottom border a darker color so that we achieve a three-dimensional shadowed effect as if the picture were in a picture frame. To complete the effect, we need to make the border fairly wide:

```
img{  
  display:block;  
  padding:25px;  
  margin:20px;  
  border-left: 15px solid red;  
  border-top: 15px solid red;  
  border-right: 15px solid maroon;  
  border-bottom: 15px solid maroon;  
  background-color: teal;  
}
```

This is a picture of Hexxa:



I took this picture at the fair.

2.5. Spans, Divs, and Classes

Now that you understand the difference between *inline* and *block* display properties, let's take a look at *spans*, *divs*, and *classes*:

- A **span** is just an **inline** element that you define yourself.
- A **div** is just a **block** element that you define yourself.
- A **class** is a **named collection of CSS properties** used for a span, a div, or any other XHTML element.

We have already used CSS to define properties for built-in XHTML elements such as `<a>` anchors and `` images. **Div** and **span** allow us to create our own elements when no built-in element meets our needs. In order for this to work, **divs** and **spans** have an attribute called *class*. A *class* is like a category. All we have to do to make a class is give the class a name, prefix the name with a period, and define the CSS attributes for the class.

2.5.1. Spans

Let's look at spans first. Suppose that you want to have a party. Instead of sending invitations by snail mail, you decide to email your friends and tell them to check out the invitation on a web page. When you make the web invitation, you want to use a font that is appropriate for a party invitation. Styling a common serif or sans-serif font by simply making it **bold** (using the `` tag) or *italic* (using the `<i>` tag) isn't enough. What you really want is to create your own category of text. Just as "italic" and "bold" are well-known types (or categories) of text, what you want is a new category called "party".

Let's say you and your friends already have some wild and crazy fonts installed on your computers. Here's one called "Cocktail Bubbly":



A^o B^ob C^oc D^od E^oe F^of G^og

Here is our CSS definition for the "party" class:

```
.party{  
    font-family: "Cocktail Bubbly", sans-serif;  
    font-size: 50px;  
    color: blue;  
}
```

First, notice that the definition of the class "party" **starts with a period**, "." . All class definitions begin with periods.

Secondly, notice the "font-family" line. This line says to use the "Cocktail Bubbly" font, but if that is not available (because not everyone has this font), then fall

back to using the default sans-serif font.

Here's a snippet of XHTML code from the invitation:

```
<p>  
  <b>Please</b> come to my <span class="party">Cool Party</span>  
  to <i>celebrate</i> the <span class="party">New Years!</span>  
</p>
```

And the result seen in the browser is:

Please come to my *Cool Party* to *celebrate* the *New Years!*

2.5.2. Divs and Ids

As mentioned, a **div** is just a block element --that is, a *rectangular area*-- on the web page that you define yourself. **Divs** are commonly used to divide web pages into parts, such as a title area, a navigation menu, a content area, a news area, and so on. To keep things simple in this tutorial, we will divide our web page into only three parts: a *title* area, a navigation *menu*, and a *content* area:

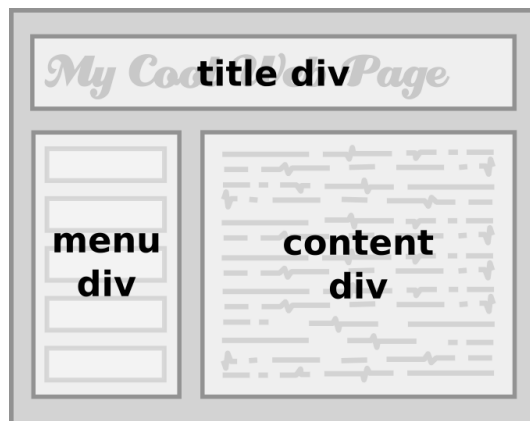


Fig. 3. Schematic layout of an XHTML document with three **divs**.

If you've followed along this far, you may be already thinking to yourself that all that is needed is three *classes*, one called "title", one called "menu", and another called "content". And you are very close to being right!

Actually, instead of classes, we are going to use IDs. An ID is a special kind of CSS class:

- An **ID** is a CSS class that **occurs only once**.
- In contrast, a regular CSS **class** can **occur many times**.

Previously we defined a class called "party" and we used it twice --once to style the words "Cool Party", and a second time to style the words "New Years". We

could use the `` element over and over again, even hundreds of times, in our web document (We could also use `<p class="party">` or even `<div class="party">` to apply the "party" attributes to a whole paragraph or a whole div block).

In contrast, we know that we only need to have a single instance of "title", "menu" and "content" on our web page:

- When you know that an element will occur only **once** on a page, use an **ID**. Otherwise, use a class.
- ID definitions start with a hash sign, "#" instead of a period.

In our XHTML page, the title, menu, and content divs are written as follows (**fig. 4**):

```
<div id="title">
  <h1>My Cool Web Page</h1>
</div>
<div id="menu">
  <p>... menu goes in here ...</p>
</div>
<div id="content">
  <p>... content goes in here ...</p>
</div>
```

Fig. 4. XHTML snippet for a CSS layout with title, menu, and content areas.

Fig. 5 summarizes what we've learned about classes and divs. In the XHTML file, we use the "class" attribute to specify classes, but we use the "id" attribute to specify ids.

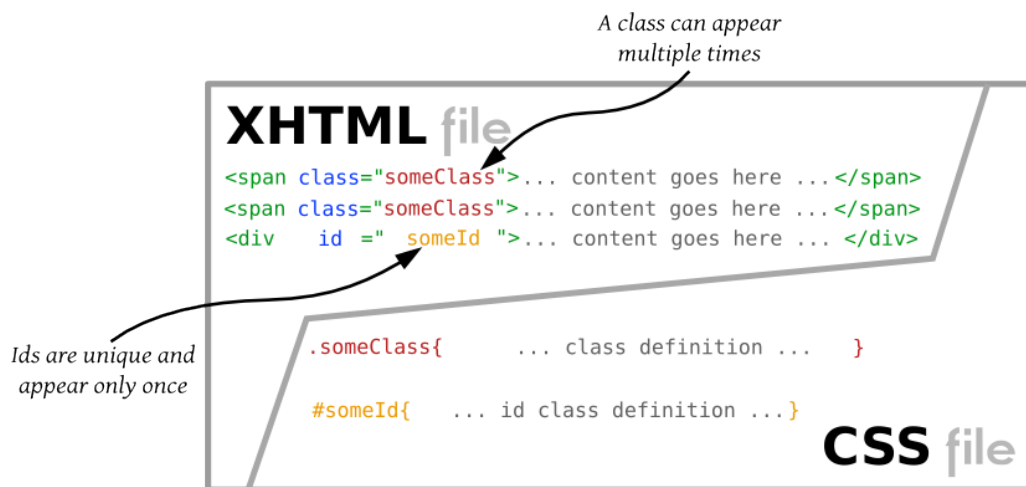


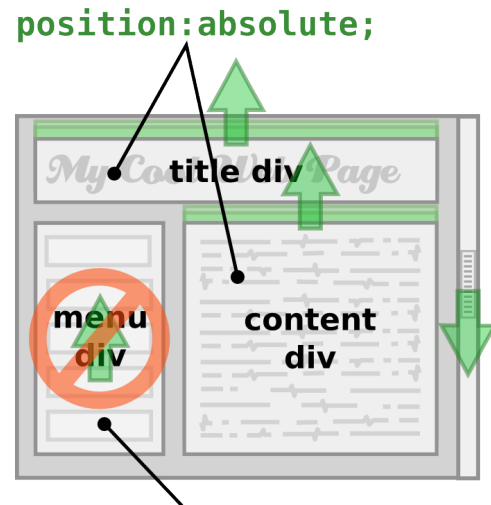
Fig. 5. Classes vs. IDs. Classes can appear multiple times, while IDs are special classes that appear only once. Pay attention to the differences in syntax between classes and IDs.

In the CSS file, class definitions begin with a period "." while ID definitions begin with a hash mark, "#". The only other thing to remember is that IDs are unique, which means they should occur only once in your XHTML file.

Now let's define our title, menu, and content divs in order to get the basic layout shown in **Fig. 6**.

First, we define the positions of the *title* and *content* divs as *absolute*, but the *menu* div as *fixed*:

```
#title{
    position: absolute;
}
#menu{
    position: fixed;
}
#content{
    position: absolute;
}
```



position: fixed;

Fig. 6: Absolute vs. fixed positioning. Absolutely positioned elements can scroll up or down while fixed elements do not move at all.

In CSS, *absolute* elements will scroll when we drag the scroll bar or use the mouse wheel, while *fixed* elements will remain fixed and won't move at all. We want the menu to remain fixed because we don't want it to scroll off the screen when we scroll the content.

We define the positions, widths, and heights of the three divs so that they will not overlap. We set the **top**, **left**, and **right** attributes of the **title** to be **10** pixels away from the edge of the body. Then we set the height of the title area to be **80** pixels high:

Next, we do the **menu**. Why do we set the top of the menu to **104** pixels? Well, we want the menu to be **10** pixels away from the title. The title is **80** pixels tall, plus the **10** pixel gap at the top, plus another 10 pixel gap that we want between the title and the menu. So, **10+80+10=100**. But --since we want perfect results-- we also have to add in the thickness of the line around the title. That's **2** more pixels for the top line of the title, and **2** more for the bottom line, so the total is **10+80+10+2+2=104** pixels:

```
#title{
    position: absolute;
    border: 2px solid red;
    top: 10px;
    left: 10px;
    right: 10px;
    height: 80px;
    padding: 0;
}
#menu{
    position: fixed;
    border: 2px dashed green;
    top: 104px;
    left: 10px;
    width: 130px;
    bottom: 10px;
    padding: 10px;
}
```

The settings for the content area are similar to the menu. The reason that "left" is set to **174** pixels comes from the fact that we don't want to overlap with the menu, so:

$$\begin{aligned} & \mathbf{10} \text{ px left} + \mathbf{2} \text{ px border} + \mathbf{10} \text{ px padding} + \mathbf{130} \text{ px width} + \dots \\ & \mathbf{10} \text{ px padding} + \mathbf{2} \text{ px border} + \mathbf{10} \text{ px right} = \mathbf{174px} \end{aligned}$$

If you think these calculations are tedious -- you're right! Most of the time, we can just "guesstimate" and see whether the result looks OK in the browser. It's faster! Here then is the definition for the content div, and the result when viewed in Firefox (**fig. 7**):

```
#content{
  position:absolute;
  border:2px dotted blue;
  top:104px;
  left:174px;
  right:10px;
  bottom:10px;
  padding:10px;
}
```

Try resizing the browser. You will notice that the divs we defined expand and contract perfectly as we enlarge or shrink the browser window. This is called a *liquid layout*. Liquid layouts are one of the benefits of CSS over traditional design methods.

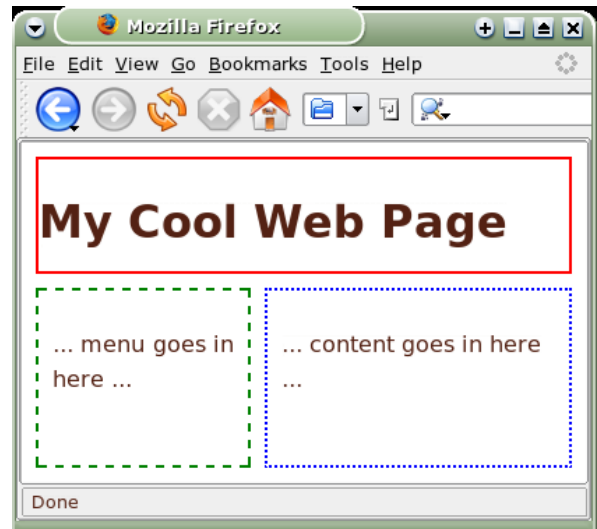


Fig. 7. The result in Firefox.

2.6. Conclusion

This introduction should be enough to get you started. The rest is up to you!

Just nest the `<a>` anchor tags from section **2.3** above inside your menu div, add some content, and play around with changing the colors and fonts in the CSS, and you will be well on your way to creating professional web sites.

Be sure to refer to the attached summary of CSS properties, read our tutorial on web colors, and download the sample XHTML and CSS files.

Good Luck!

*HeFa
& Decima*