<DIV> tags and positioning with CSS

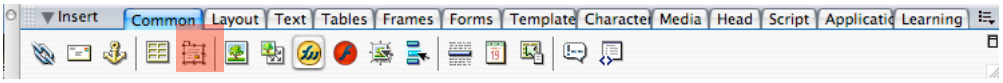
- I've posted a number of links to really helpful CSS tutorials under the InterestingLinks section of the class wiki. You'd be well served by having a look when you get the chance.
- Also: Here's a 1 page overview of the CSS box model (which determines how all CSS elements are laid out) that will help you understand positioning: http://student.ttuhsc.edu/RBoettger/lesson1.html
- This is NOT an exhaustive overview of CSS positioning techniques, but it will get you started. The best way to learn is to play
- By now, you've realized that we can't just position elements (what's an element? Well, basically, it's a tag, or anything that shows up on a web page) anywhere we like on html pages. And you've listened to me say time and again that we should not be using tables to do element layout because they're not really for element layout (what are they for? Tabular data!). That's where tags start to come in.
- What is a tag? To begin with, it's short for *division.* A is a generic *block element.* What does that mean? Well, a block element is any element on an html page that starts on its own line. You see them all the time: <p>, <h1>, <table>, and <ol> are all examples of block level elements. These are in contrast to *inline elements,* which typically do not appear on a new line—the , <a>, and tags are all examples of inline tags
- You can think of the tag as an invisible box: it's designed to hold any other element but doesn't have a default look of its own. In class we've discussed how tags will default to a certain "look" within a browser. While we can give a tag a look, its default is to appear as nothing. So, if you were to include the following tags in the Dreamweaver code space— —you'd see nothing, but if you added content to those tags in the Dreamweaver code space— 1 —you'd see the number 1. We could then use additional CSS styling (as we've been doing) to style the tag.
- So, we can't use tables and we can use tags. There are 4 types of positioning (remember, that means positioning elements anywhere on an html page) that we can use for CSS:
    - *Static positioning:* "A web browser takes an HTML file, parses it into its elements, applies style from a style sheet (or if there is none, from a default style sheet) and then "flows" the elements onto the page. The position that an element, say a paragraph, has in this flow is its *static* position. We'll need to remember

static positioning when we get to relative positioning below" (copied from the westciv CSS site). Static positioning is the default position for all elements on an html page—you don't have to define elements as "static" unless you have a specific reason to do so. We'll discuss it more below.

- *Absolute positioning:* "In a nutshell, it lets a developer say where the top left hand corner of an element should be located **with respect to its parent element**. It is not with respect to the window. Don't get absolute positioning confused with relative positioning, which we'll look at below" (copied from the westciv CSS site).
- *Fixed positioning:* "Fixed position is closely related to absolute position. When an element is absolutely positioned, it's positioned with respect to the element which contains it. When the page is scrolled, the element also scrolls. Wouldn't it be nice if we could position elements with respect to the top of the *window*, so that regardless of how the page is scrolled, the elements remain fixed? You guessed it, fixed positioning provides precisely this feature. With fixed positioning and the <object> element in HTML 4, we can do away with frames forever, yet gain all the advantages, with none of the disadvantages of the FRAME construct" (copied from the westciv CSS site). **NB: We're not going to cover fixed positioning in this exercise. We'll talk about it later.**
- *Relative positioning*: Positioning relative to what? A lot of people jump to the conclusion that relative positioning is when you specify a position with respect to the parent element, and absolute positioning is when the position is specified with respect to the top left hand corner of the page. But as we have just seen, this is **not** how it works at all. In essence, relative positioning places an element with respect to where it would *statically* be positioned. When you relatively position an element, as a developer you are saying to a browser: "hey, take this paragraph, and put it 10 pixels down and 10 pixels to the right of where it would normally be." (copied from the westciv CSS site).

The Exercise

- We're going to do this via hand coding. Yes, yes, I know—it's terrible. But it's really important that you get a sense of what the code looks like before you start using the tools that Dreamweaver has built in to build divs
- That being said, the tool that you would use in Dreamweaver to insert tags (in the Design area—not the code view area, right?) is the "Draw Layer" button. It looks like this—  —and is found in the Insert window. If you don't see it when you have Dreamweaver open, go to the Window menu up top and click on "Insert." The window looks like this—you want the area in red:



- You can click and drag using this tool in the Design area to draw an *absolutely positioned* tag. You'll notice that the code for the tag in the Code area has a number of parameters (as you've already determined the divs position and size by clicking and dragging). You can change those parameters either by making sure that the entire is selected (a hand will appear when you move the cursor over the div: click once on the little handle and the whole will be selected) and using the Properties bar (this is the one we use all the time; if you don't see it, click on "Properties" beneath the Windows menu). Otherwise, you can change the parameters by hand in the Code view.
- AGAIN: we're going to hand code in this exercise
- *Absolute positioning:* An absolute positioned div is positioned on a page wherever the top and left declarations tell it to be. You can imagine it as floating above the rest of the content on an html page.
- Create the following internal style (remember, internal styles go within the <head> tag and always start with the <style> tag). We could do all of this using our regular method in Dreamweaver (as usual, using the CSS Styles window), but we're *hand coding,* remember?

```
#red {
width: 75px;
height: 200px;
position: absolute;
top: 0;
left: 0;
background-color: red;
}
```

- The first thing to notice is the "#red." This is an ID? What is an ID? Well, remember how we created classes in Dreamweaver? Classes are a way to create special cases for a particular tag (remember, we created a special class for one <th> tag so that all of the other <th> tags would remain the same). Classes always start with a "." For example, a class called "special" would look like ".special" in the style sheet. You can use a class as many times as you want on a page. So, if you have thirty <th> tags, 4 of which need to be the "special" class, then you can apply it 4 times. ID's, on the other hand can only be used per page, and they are usually reserved for very specific elements. In this case, we are only going to have 1 called "red." We might have 10 divs on a page, but we can only have 1 called "red."
- Then, within the <body> tag (as we want the div to show up on the page, after all), create a new using the following code:
- Preview the page in a browser and see what you get. You'll notice that is flush left and flush top, just as you positioned absolutely using the CSS declaration above (the "top" and "left" indicate distance in pixels from the top and left). Try

playing around with those numbers in the Code view in Dreamweaver and then preview the page again to see what you get

- Now, set the top and left values for the div tag back to 0. Then, in the Code area, add the following <h1> tag on the line *before* the tag: <h1>This is the title line for this web page.</h1>. Preview the page. The <h1> tag is showing up where it should show up, as is the tag because you told it exactly where to show up through the style. Remember, block-level elements (as both <h1> and tags are) usually start on their own line and you don't get this kind of overlap. But absolute positioning takes the element out of its normal flow—remember to think of it as floating above the page
- Now try switching the position of the <h1> tag and the tag in the Code view. Note that when you preview it you won't see any change. Again, you've positioned that absolutely.
- Okay, now we're going to try some *relative positioning.* Change your internal style to look like the following code:

  ```
  #red {
  width: 75px;
  height: 200px;
  position: relative;
  background-color: red;
  }
  ```

- Leave the <h1> tag where it is (below the tag) in the Code view area and preview the page
- Now, given the relative positioning (and remember, relative positioning means "position this element relative to where it would be positioned if you were just reading the code from top to bottom in the Code area"), you'll note that the two block-level elements are no longer overlapping. Go ahead and switch the positions of the <h1> and tags in the Code view and then preview again. Relative positioning means *relative* to where the element would appear in the normal flow of text. So, given the CSS declaration that you've used, the call "red" will appear relative to it's original position
- Now add the "top" and "left" declarations back into your code, which should now look like this:

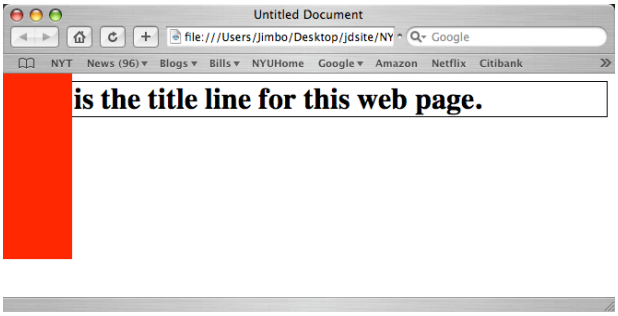
  ```
  #red {
  width: 75px;
  height: 200px;
  position: relative;
  top: 50;
  left: 50;
  background-color: red;
  }
  ```

- When you preview this, the will now appear 50 pixels from the top and 50 pixels from the left relative to where it would appear if it was just a part of the regular html rendering
- Next: style the <h1> internally so that it has a solid, black border and has no margin (*margin* is the space around the element, while *padding* is the space inside of it). The declaration (remember, up in the <style> that lives inside the <head>) should look like this:

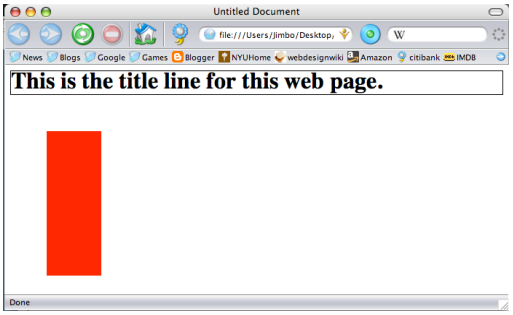
```
h1 {
border: 1px solid #000;
margin: 0;
}
```

- Now, try positioning the div tags absolutely and relatively again (using the different codes from above). They should render something like the following in a browser, depending on the values that you've used in the CSS declaration:

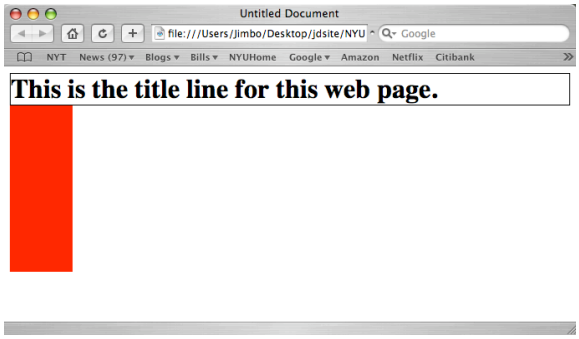
Absolute positioning



Relative positioning



- The absolutely positioned takes it position in relation to the element that contains it. Remember, elements are tags. So, the element (or tag) that the div lives inside of here is the <body> tag.
- The relatively positioned is positioned relative to the <h1> element (tag) that it follows. The relatively positioned is *in the flow* of the page (unlike the absolutely positioned which, as we said, is essentially floating above the page) and is affected by the other elements that surround it
- Change the top and left values in the relatively defined back to 0 and see what it looks like in preview. It should resemble the following, as the is now 0 pixels from the top and 0 pixels from the left of where it would appear normally:



- Finally, we're going to look at *static positioning*. Change your CSS declaration to look like the following:

```
#red {
width: 75px;
height: 200px;
position: static;
top: 0;
left: 0;
background-color: red;
}

h1 {
border: 1px solid #000;
margin: 0;
}
```

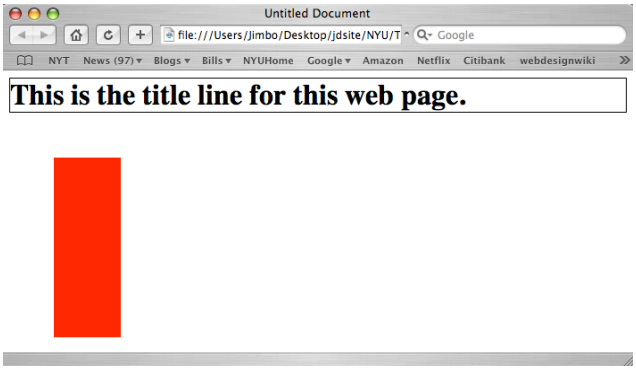
- Go ahead and preview the div in a browser. It should look like the relatively positioned from the previous step (with the 0 values)
- Kay, now set the top and left values to 100 pixels and see what happens. Nothing. Why? Because static positioning doesn't respond to the top and left properties. Now, however, delete the top and left properties and replace them with the "margin-top" and "margin-left" properties. The CSS declaration should look like this:

```
#red {
width: 75px;
height: 200px;
position: static;
margin-top: 50;
margin-left: 50;
background-color: red;
}

h1 {
border: 1px solid #000;
```

margin: 0;
}

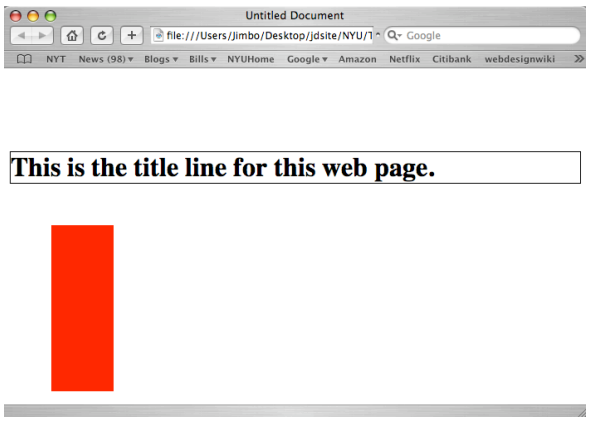
- Now preview the and you'll see that the div is modified by the margins (areas surrounding the element, according to the box model) and affected by other elements on the page. It'll look something like this:



- Let's see how a change to the <h1> tag that precedes the div tag in the Code view will affect the tag. Change the <h1> CSS tag declaration to look like the following:

h1 {
border: 1px solid #000;
margin: 0;
margin-top: 100;
}

- The result should resemble the following screenshot. Notice how the tag is affected by the styling associated with the <h1> tag—the margin on the <h1> has caused the to move that much lower. If this were absolutely positioned, the margin on the <h1> wouldn't matter because absolutely positioned <divs> are unaffected by surrounding tags:



- Okay, we'll wrap the tutorial up with this. We'll talk more about "stacking orders" (or, how divs can be "stacked" on top of one another, like a deck of cards)

and floating on Thursday. Again, I can’t recommend enough that you look at those links on the wiki. :-)