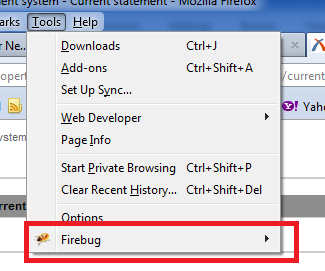
HTML for Web Development

# CSCI 4300

### Before starting this exercise:

1. Make sure you have a Java Development Kit (JDK) with version 1.6 or higher. If you are not sure and can’t find your JDK, download the latest “standard edition” (SE) JDK from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
2. Download the Eclipse IDE for Java EE Developers from <http://www.eclipse.org/downloads/>. If you already have the Eclipse IDE from a previous course but don’t have the Java EE version, go ahead and download the Java EE version. It will save a lot of hassle.

During the Eclipse installation process, you’ll be asked to specify a workspace. I recommend placing your workspace folder inside your Documents folder, which will make your files easier to find. At any rate, be sure to remember where your workspace is!

1. Make sure you have the Firefox Web browser. These notes use Firefox because it has some excellent Web developer tools.
2. In Firefox, check to see if the Firebug tool is already on your Tools menu. use the **Tools/Web Developer/Get More Tools** menu item to get the Firebug debugger. You may need to restart Firefox.

### Overview:

### *HTTP* and *HTML* are two key standards for the Web*.* Here is what they mean:

The ***Hypertext Transport Protocol (HTTP)*** is a computer networking protocol, like TCP-IP, which lets computers talk to each other. In a nutshell,

* TCP-IP is the base language which all computers on the Internet use to communicate;
* HTTP is the special language used by *servers* on the World-Wide Web.

We call these special languages ***protocols***. Since HTTP and TCP-IP are used by client and server programs to talk to each other, you don’t need to be able to understand these languages to write basic Web pages. Your operating system knows how to “talk” TCP-IP, and your Web browser program knows how to “talk” HTTP. To make more sophisticated, interactive Web applications, you will need to learn about the workings of HTTP.

All of the Web pages you see, with the exception of any pages on your own system, are hanging out on somebody’s server. If you want to put your Web page masterpiece (the one you are about to create in this lab) out on the Internet for everyone to admire, you have to find an HTTP server for it. This server will hopefully be a campus machine on which you have a free account, or it could be a commercial server. If you have a paid account on an Internet Service Provider (ISP), your monthly fee probably includes space to put your Web pages.

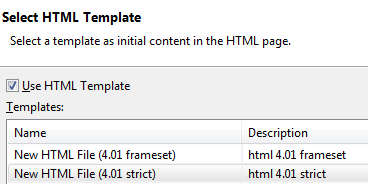
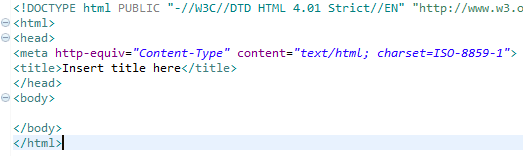
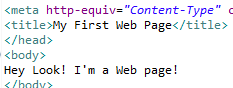
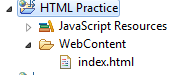
To make a long story short (too late!), only advanced Web programmers and server administrators need to know anything about HTTP.

The ***Hypertext Markup Language (HTML)*** is a different story. HTML is the actual language in which Web pages are written. So if you want to write Web pages, it helps to learn HTML. Fortunately, HTML is a lot easier to learn than programming languages such as BASIC or Java. It’s more fun, because you can see the visual output immediately.

# The HTML Authoring Exercise

1. Launch Eclipse. From the **File** menu, choose **New/Static Web Project** or **New/Other/Web/Static Web Project.** In this context, static means that the Web content of this project will be stored in files that you write, as opposed to dynamically generated by Java classes running on the Web Server. Name the project **HTML Practice.**

When you press the Finish button, Eclipse will ask if you want to open the Web perspective. In Eclipse lingo, a perspective is a set of window panes that are normally found helpful or a particular task (editing static Web content, in this case). Go ahead and accept the suggestion. If you ever have difficulty finding a particular window pane, use Window/Show View to re-open it.

1. Right-click the HTML Practice node and choose **New/HTML page.** Name the page **index.html** – this is a standard name for the “welcome page” that first appears when your project is run on a server. Click.
2. In the next screen you will be asked to choose an HTML template. Each template is based on a particular Document Type or version of HTML. In this course, we will write pages that strictly follow the grammar rules laid down by the World-Wide-Web Consortium (W3C). Therefore, please choose the **HTML 4.01 Strict** template. Click  to create the template HTML page.
3. What you see now will save you a great deal of hassle in the future. 
4. Let’s take a moment to identify some of the useful content that has been auto-generated by Eclipse:
   1. The <!DOCTYPE> declaration identifies the particular HTML version being used;
   2. The <META tag identifies this content as an HTML file and names the character set. “ISO-8859-1” is a later version of the good old ASCII character set, otherwise known as “vanilla text” or “typewriter text” (not that any of your generation have ever seen a typewriter). Basically, this is plain text characters without any formatting.[[1]](#footnote-1)
   3. The <title> of the page will appear in the title bar when the page is viewed in a browser.
   4. The <body> of the page will appear in the main browser window. Right now, the body is empty.
5. Change the title of the page to “My first Web page,” as shown at right.
6. Add content to the body of the page, “Hey Look! I’m a Web page!”
7. Now let’s look at the page with Eclipse’s built-in browser. First, save your changes by clicking the SaveAll icon . In the Project Explorer view, open the HTML practice node so that you can see **index.html** in the WebContent folder. Right-click index.html and choose Open With/Web Browser. You should see “Hey Look! I’m a Web Page!” in the browser window and the title, “My first Web page” on the Eclipse tab.
8. Launch Firefox and open this page. (If you don’t see a File menu, click the Firefox tab and look under **New Tab**. Navigate to your workspace folder (you did remember where this is, right?) and find index.html in the WebContent folder. It should look more or less the same as in the Eclipse built-in browser.

# Validating your Web Page

**Validation** is a mechanical process to verify that your page follows the formally defined rules of HTML grammar. These rules can be picky, just as when someone makes you use precisely correct English grammar. We use the validation check for two reasons:

1. Checking the grammar can reveal subtle errors in our HTML source code;
2. HTML code that complies with the formal grammar is much less likely to suffer **browser incompatibility** problems: pages that display fine in one browser, but look terrible in a different browser.

To use the validator, open a new browser tab from the File menu. Navigate to [http://validator.w3.org](http://validator.w3.org/), the validator page. Use the **File Upload** interface shown here. Use the **Browse** button to find your file, and then click the **Check** button to validate the file.

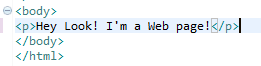
Your first attempt will yield an error message similar to the one shown at right. If you read the accompanying verbiage, you will (maybe) see that the validator doesn’t recognize your DOCTYPE declaration.

This appears to be a glitch in Eclipse ☹. You can fix it by editing your DOCTYPE declaration to remove the word STRICT, as shown here. Save your work and reload in to the validator page. The “Unable to determine parse mode” error should now disappear.

You should now see the following error (maybe not on line 8):



What is the validator saying? Well, it clearly says you are not writing valid code! But what is wrong?

Read the error message closely. Validator suggests that you might have put text directly into the body, and that is exactly what we did. The suggested remedy is to put the text into a **container** element, such as a paragraph element.

Let’s try it! Go back to Eclipse and add a <p> paragraph tag as shown. Now go to the end of the line and type “</”. Eclipse recognizes the start of a closing tag, and autocompletes the </p> closing tag. Way cool, eh?

Save the page and reload Validator again. This time you should see the success message shown below. Yippee!



Click on the Mozilla tab that contains your page and click the Reload button. The display should stay exactly the same.

Now use an <h1> tag to add a header to the page, as shown here. Watch for Eclipse to autocomplete the closing tag! Revalidate the page.

# The Firebug debugger

In Firefox, find the Firebug button  and click it. You should now see a window in which you can explore the HTML structure of the document. Way cool, eh?

Now close the Firebug window . Select . Right-click and choose  (not ). Firebug re-opens and is focused on the particular **element** you chose. The relationship between tags and elements is a bit subtle: tags in the source code create elements in the document. The elements are more abstract entities, like Java objects. More about this later in this exercise.)

# Images and Attributes

Web pages are ***compound documents***, meaning that the page content might be contained in multiple files. Images are one example of this: the actual image content is contained in a separate file in one of the standard image file formats such as .PNG, .BMP, .GIF, or .JPG.

Add the code shown at right to your **index.jsp** page to create an image element. The notations *src=…* and *alt=…* are ***attributes***. Attribute values are always defined within the opening HTML tag using ***name-value semantics***, as shown here. In this case the names of the two attributes are *src* and *alt*.

As you can probably guess, the *src* attribute gives the name of the file that contains the actual image data. To find the meaning of the *alt* attribute, open a Web browser window to **blooberry.com** and select the HTML reference. Look up the IMG element and find the meaning of the ALT attribute under “Specific Attributes.” The description may be a little obscure, so save **index.jsp** and view it in your browser. The browser is unable to display the actual image (because the file is not there!) so you will see the alt message instead.

To fix the problem, download the **FredFlintstone.jpg** file from the Exercises folder of eLC, and store it in your Web Contents folder in the Eclipse workspace folder for this project. Reload the page – you should now see Fred in all his glory!

**Height and width attributes:** Right-click on the Fred Flintstone image and choose **View Image Info**. You should see a message similar to the one at right. The message mentions **caching** an image; this is the browser’s trick of storing the image file in local storage so that when you reload a page, the image can be retrieved from the cache instead of being downloaded from the Web.

In this case, the image was not cached because it is already stored on local storage. To see what is in the Firefox cache, type **about:cache** in the URL bar. Click **List Cache entries** to see the files that Firefox has cached – they will be media files downloaded from the Web.

Go back to the HTML source code for this page in Eclipse, and add the width and height attributes as shown here. This is considered good practice, because if your page is downloaded over a slow Internet connection, the browser won’t know the image size until the image has finished downloading. With the width and height attributes specified, the browser can lay out a 202 x 433 pixel rectangle and arrange the rest of the page content around it, while waiting for the image to download.

# Intellectual Property and Intellectual Honesty

Any time we use page content that we did not create, such as Fred’s image, there are legal and ethical issues to consider:

***Intellectual property*** is a legal issue: are we violating the property rights of the copyright owner? In general, non-commercial academic productions are covered by the Fair Use Doctrine, which allows limited exemptions from copyright laws for scholarly purposes.

***Intellectual honesty*** is known around here as “Academic Honesty,” but goes further than that. Intellectual honesty means not passing off someone else’s creative work as your own. Just as you always give attribution to quotes in your term papers, you should give attribution to the provider of your Web page content. (If that provider in turn ripped it off from somebody else, that’s too bad, but I won’t require you to chase down the original copyright holder. In a commercial application, you definitely should.)

In summary, please add a note of attribution for anything you copy. This can be visible (my preference) or hidden as a code comment. In this way we are not only being honest about our own work, but honoring the creation of others. Here is an attribution for the Fred Flintstone image, included in an **HTML comment**:

# The DIV element

Try validating index.html again. You should get a slew of errors, starting with “character data is not allowed here.” The first suggested possible error in the validator is “putting text directly in the body of the document without wrapping it in a container element,” which is precisely what we’ve done.

The error message also gives a list of possible start tags that might solve the problem. We’ll use the DIV element. Look up DIV at [www.blooberry.com](http://www.blooberry.com), and follow the link to the description of **Block Formatting**. You need not read this page in its entirety (although that would be worthwhile). Just notice that block formatted elements should have a line break before and after.

Edit your page to add a <div> tag before the <img> tag and a </div> tag after. The page should now validate. However, the DIV element also will force the image to occupy its own horizontal division of the page. Later on, we’ll show how to lay out an image side-by-side with text and to wrap text around an image.

# Lists and Tables

HTML lists and tables are called **container** elements:

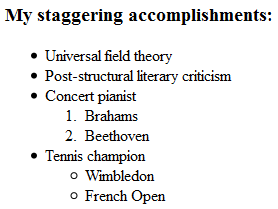
* A **list element** contains **list item elements**, which contain the actual list item text;
* A **table element** contains **table row** **elements**, which contain **table data elements**, which contain the actual data.

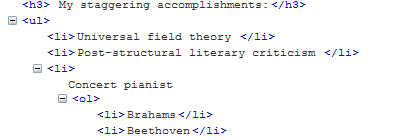
Lists can be ordered lists (numbered lists) or unordered lists (bullet lists). Later we will show how to add styling to make an ordered list display its items as 1, 2, 3, or A, B, C, or I, II, III as well as other numbering types.

Let’s start by creating a couple of lists. In your project, create a new HTML file named vonManderpootz.html, which will chronicle a character from the 1930’s science fiction stories of Stanley Weinbaum. Here is the result and the starter HTML code:

|  |  |
| --- | --- |
|  | * Note how the <li> -- </li> pair, which defines a list element, lies within the <ol> tag and its matching </ol> tag (not shown here. This is what we mean when we say that the OL element *contains* the LI elements. * Don’t forget to close this first ordered list with an </OL> tag before starting the second <h3> header and the second list. * The second list is an unordered list, so use a <ul> tag instead of an <ol> tag. |

When you have finished generating this list, use the validation service at validator.w3.org to make sure that it validates as HTML 4.01 Strict.

**Nested lists**: Now add sub-lists to the last two unordered list items as shown here. Each sub-list is a part of a list item; for example, the list of composers is a part of the list item, “concert pianist.” Therefore, the entire list of composers, including the <ol> and </ol> tags, must come between the <li> and </li> tags of the “concert pianist” list item. As you can see here, the sub-list should come after the text content of this list item. Try it out!

When you have completed the two sub-lists, make sure they validate. Then return to von Manderpootz.html and launch **Firebug** by clicking the bug icon  at the bottom right corner of your Firefox window. Click on the HTML tab and open the second list, which appears as a <ul> element. You should see that the third and fourth list items can be opened to reveal their sub-lists, as shown here. Do you see how the nested tags in your HTML code correspond to the tree structure shown in Firebug?

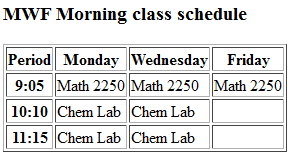
**Tables**: create another page called **morningSchedules.html** (using HTML 4.01 Strict, as always!) We will create an HTML table to represent a hypothetical student’s MWF morning schedule, as shown here. Tables have a more complex containment structure: tables contain table rows (TR elements), which contain table data (TD elements). Complete the starter code to reproduce the table shown below. You will not see the borders between table cells yet! Make sure your code validates before proceeding.

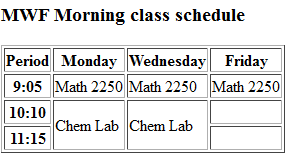
|  |  |
| --- | --- |
|  |  |

**Attributes**: How can we get the borders? Open a separate Web browser window to [www.blooberry.com](http://www.blooberry.com) and select the HTML reference. Navigate to the TABLE element and look under “Specific attributes” for the Border attribute. See if you can find these two key facts about the Border attribute:

* It is valid HMTL 4.01 Strict;
* Its value should be an integer denoting the border width in pixels.

Attribute values are always defined within the opening HTML tag using ***name-value semantics***, as shown here. In this case the attribute *name* is “border,” and the attribute *value* is “1.” Although this particular attribute value is an integer, it is conventional to always enclose attribute values in quotes.[[2]](#footnote-2)

**Table headers**: in your blooberry browser window, look up the TH element. Try changing the TD elements in the first row and the first column of your table to TH elements. Does this look better?

**Merging table cells**: Look up the TD element in your Blooberry reference and find the *rowspan* attribute. Using this attribute, define the two “Chem Lab” elements in the third row to take up two rows each. Remember that attributes are defined inside the opening tag, using name-value syntax. When this is done, the “Chem Lab” elements in the fourth row are no longer needed, because the third row elements now fill their slots in the table. The result should look like the example at right.

By the way, there is also a “colspan” attribute for use when a table cell spans multiple columns.

## The Document Object Model

Use the Firebug HTML tab to open the <table> element. You will see an element, <tbody>, that does not correspond to any tag in the HTML source code. TBODY represents the the body of the table, as opposed to specialized header and footer rows. We don’t identify our header rows as such in this simple table, but if you write tables that span multiple pages you might want to look into the THEAD element.

TBODY is part of the **Document Object Model (DOM)**, which is how Web browsers represent a document as collection of objects. As you have seen, tag pairs such as <li> … </li> create **elements** that you can view in Firebug; these elements are part of the DOM. But as you have just seen, the DOM contains more than these elements. We’ll have more to say about the DOM later on in this course.

**Tables for side-by-side layout**: Tables are commonly used to lay elements out side-by-side. Return to your **vonManderpootz.html** page and create a table with one row, two TD elements, and no border. Move the lists and their headers into the first TD, and put an image tag in the second TD. Set the **src** attribute to “MadScientist.jpg” and the **alt** attribute to “A shining example of Humanity”. Also add an <h3> header element, “Dr. Felix von Manderpootz,” before the <img> tag.

Download the “MadScientist.jpg” file from eLC and store it into your WebContent folder. If you have done everything correctly, the result should look like the image at right.

Giving attribution: as mentioned above, it is important to be respectful of intellectual property and also to honor the creative work of those who have gone before us. Add the following paragraph below your page:

## Input Elements

HTML input elements allow the user to specify input variables that will be used by Web applications. (We will show how the Web apps access these values later on.) Input variables are specified using name-value syntax again: the variable name is specified in the HTML code, and the value is whatever the user enters.

Return to index.html and add this code at the bottom of the page. Save and view the results; you should see empty text boxes for the user to enter their first and last name. Note that the name attribute is not visible to the user, so we must add a prompt.

It’s possible to specify initial values for the text boxes. Try adding value attributes as shown at right. When you save and view the page, you will see the values in the text boxes. However, you can still type in text to override these values. Single quotes and double quotes are equivalent in this context.

Revalidate index.html again. You should get a slew of errors, starting with “character data is not allowed here. Once again, we have “[put] text directly in the body of the document without wrapping it in a container element.” Do you remember how we fixed that problem with the Fred Flintstone image? This time, use a <P> element as your container. Revalidate the page to confirm.

# About HTML 5

HTML 5 is the latest version of HTML, but it has not yet been accepted as a W3C standard and is not fully supported by all browsers. To learn some of the cool features of HTML5, visit <http://www.w3schools.com/html/html5_intro.asp>. I especially like the new input elements! We won’t be using HTML 5 in this course, however.

1. For this reason, I recommend against writing HTML files using a word processing application or Mac text editor. These programs are fond of fancy characters that are not in the ISO-8859-1 character set and may appear as ugly symbols on other people’s browsers, even if they look great on your machine. [↑](#footnote-ref-1)
2. These borders look a bit clunky. For a nicer looking division between table cells, you can set the background color of the table element itself, then space the table cells so that the background shows through. This requires CSS, which you will learn in the next exercise. [↑](#footnote-ref-2)