FIT100: Fluency with Information Technology

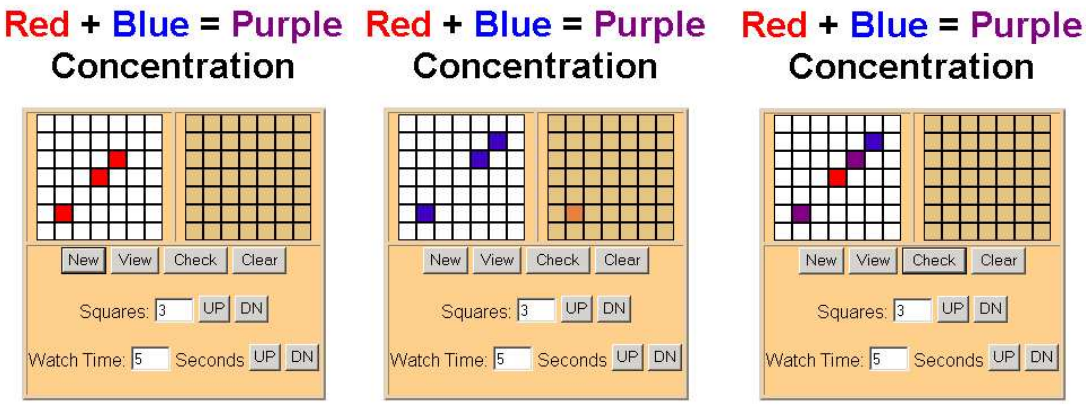
Project 2

Winter 2004

The goal of Project 2 is to introduce JavaScript, a programming language that allows the creation of active Web pages. It is possible to create Web pages with animations, pages that perform computations, and pages that have "forms" type inputs, like surveys. Project 2 will demonstrate several of these features.

> In all parts of Project 2 you must comment the HTML and JavaScript that you write or change. The comments do not have to be long, but they must say what you've done in a way that is coherent to a human. *Uncommented HTML and JavaScript will not be graded.*

The ultimate goal of the project is to produce the Purple Concentration application.



(a) Random Problem (b) User Guesses (c) Solution Tested

This is an ambitious project, so we do it in 3 parts. We begin first by a practice exercise in personalizing the Memory Bank application. Then in two more parts, we build the Purple Concentration application. Each part will have a separate turn in, and there will be a version of part B available at the start of part C.

The application has two grids. The one on the left is the Display grid and the one on the right is the Key grid. The application works as follows. The user clicks "New" to start a new concentration challenge. A pattern of *n* randomly chosen red squares is placed in the Display grid, (a). The user can control *n*, which is initially 3. This pattern remains displayed for *s* seconds, and then it disappears. The user can control *s*, which is initially

5. The user then keys in the same pattern in the Key grid, on the right. With each click on the key grid, (b) the corresponding square in the Display grid is set to blue. (In Figure (b), the cursor is over the square shown in brown.) When the user thinks the pattern is right, he or she clicks the Check button. The matching positions are shown in purple, and the missed squares are shown in red (c). If the user forgets the pattern and wants to see it again, clicking on "View" shows it again for *s* seconds. Finally, the marked squares can all be cleared using the "Clear" button. (The pattern is remembered.)

Project 2A Exercise

The purpose of part 2A is for you to get experience modifying and running JavaScript programs in order to become familiar the JavaScript programming process. So, the task is to customize the Memory Bank program from Chapter 20 pp. 559-572.

Read these pages before proceeding further.

The source for that program can be found at www.aw.com/snyder/ and copied. (Remember, you need to grab the GIF for the bullet mark, too.) Or, you can type in the code from Appendix D, pp. 690-693; but if you do, type it very carefully to avoid errors.

Your specific tasks are as follows (refer to the Memory Bank page).

A1. **Change the Look**. Make some modification(s) to the "look" of the Memory Bank page. You can completely change it, or only slightly modify it by revising its background

color, font, font color, etc. to your liking. The result should be attractive. Also, change the level 1 heading to include your name. For example, I would change it to "Larry's Memory Bank."

A2. **Add a link to the Web page**. Since JavaScript uses HTML, the links to remote pages are just HTML links, which you are already familiar with. Change the links in the Memory Bank program so that they point to destinations of interest to you.

(a) To the **Reference** links, add a link of personal interest, or a link to `www.google.com/advanced_search?hl=en` (or both).
(b) Change the **Classes** entries to match your classes, i.e. make some link suitable for each of your classes this term.
(c) Add one more **category** of links of your choice, say, links to **Friends**, links to **News**, or a **Hot-List** of favorite sites.

A3. **Good as Gold**. Add a new computation line to the Memory Bank page. Your computation can be one of your own choosing, or it can be the following computation to compute how much a weight in gold is worth, or both. (On January 28, gold was trading for $409.90 per troy ounce, of which there are 12 per pound.) Therefore, the "weight in gold" computation is

*Worth = (Weight*12)*409.9*

Using the analogy of the Celsius to Fahrenheit computation that already exists on the Memory Bank page,

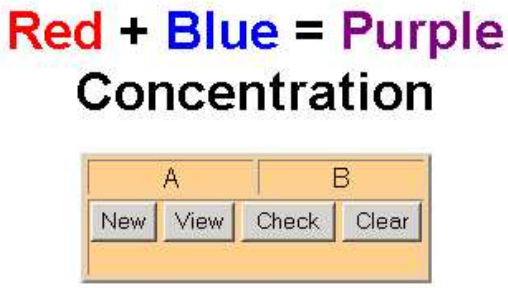
(a) Create a new row in Memory Bank with the proper title (left side table data).
(b) For the right hand side table data, construct the text input window. Begin with the text "Weight LBS", and then place the input window, giving it a unique name and a size of 3.
(c) Construct the text input window (for output). Place the text "Worth As Gold $", and then place the window, giving it a unique name, and a size of 7.
(d) Construct a function—place it with the other functions—named `goldworth()`, and having one parameter. The function `goldworth()` returns the dollar value of a given weight (in lbs) of gold.
(e) Define the `onChange` event handler for the input control of (b) to assign the result of the `goldworth()` function applied to the input value from (b) to the output window of (c). (Notice how this is done in Celsius to Fahrenheit line.)
(f) Include the following "last modified" code on your page at the bottom. (If the position where you place this code is already inside JavaScript tags, you do not need those shown.):

```
<script language = "JavaScript">
var _modified;
document.write("Last Modified: ");
modified = document.lastModified;
```

```
document.write(modified);
</script>
```

Hint: Part A3 is extremely easy if you have studied and understood the Celsius to Fahrenheit computation.

A4. **Create the Purple Concentration Page**. Having practiced input controls for the "Good as Gold" computation, build a page for the Purple Concentration application. Give it a heading as shown in the figure. Then, set up a table with a beige background (#FFCC88), a two-unit border, and two rows. The first row has two data entries (use 'A' and 'B' for the data until we set up the pattern and key grids), and the second row that has only one data entry that spans two columns. (Check the Memory Bank's definition to see how the border and spanning two columns are done.) Place four command buttons—New, View, Check and Clear—in the second row in preparation for the next part. The page will be very primitive, but it will be filled in later. (Notice how everything is centered.)



A5. Include the following "last modified" code on your page at the bottom.

```
<script language = "JavaScript">
var _modified;
document.write("Last Modified: ");
modified = document.lastModified;
document.write(modified);
</script>
```

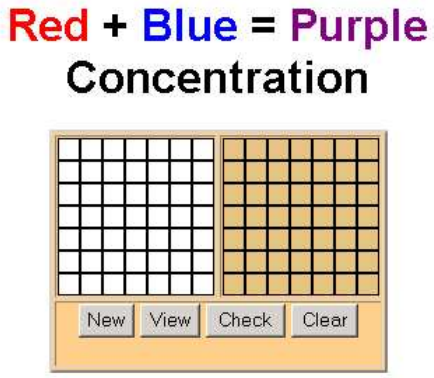
A6. **Turn In**. Publish your new Memory Bank page in your Web space under a unique name. (Your Memory Bank page is for your use, not for public use, so choose a name that is memorable to you.) Remember to include the `bullet.gif`. Also, publish your Purple Concentration Page in your Web space, also under a unique name.

(a) Have your pages published by 11:30 PM, Tuesday February 10, 2004.
(b) Do not modify your page until after it is graded.
(c) Print the HTML/JavaScript source for both pages, add your name and section letters, and turn it in in class on Wednesday, February 11, 2004.

In all parts of Project 2 you must comment the HTML and JavaScript that you write or change. The comments do not have to be long, but they must say what you've done in a way that is coherent to a human. *Uncommented HTML and JavaScript will not be graded.*

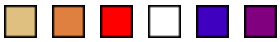
Project 2B Grid Arrays

The purpose of Part 2B is to set up the Display and Key grids for the Purple Concentration application. The "grids" are actually tiny GIF images, and as the mouse moves over the Key grid, they change color. The resulting page has the form



This part depends on your knowing "Prefetching Images" and "Redrawing Images," (pp. 596-599) of Chapter 21, and "Key Sensing Task" (pp. 620-623) of Chapter 22. *The following explanation assumes you're familiar with that reading.*

B1. **Grab the GIFs**. Retrieve the six `.gif` files to your desktop from the Resources page at the class Web site. You will get a brown, orange, red, white, blue and purple square:



You will only use some of these GIFs for Part B, but you will use them all by the end of the project. So, get them now.

B2. **Prefetch the GIFs**. Step B1 only got the GIFs to your computer so you can work on the project. The Web page (from A4) must also get the files for its use. This is called prefetching the files, and is done so that the time to transmit them over the Web doesn't cause delay in the application.

As explained in Prefetching Images in the textbook, it is necessary to declare an array to hold the fetched images. You need an array with as many elements as images, that is, six. The array elements must be initialized to be `new Image`. And then, the GIFs must be assigned to the array elements. You will perform the following steps:

a) Specify a pair of script tags so you have a place to write your JavaScript program.
b) Using **var** declare an array variable with a unique name of your choice; the array variable will contain the images, and so will need six elements; also, because iteration variables will be needed, declare two of them now, such as `i` and `j`.
c) Write a loop – use the World Famous Iteration and use `i` as the iteration variable – to initialize the array elements to `new Image`.
d) Assign the six GIF images to the `.src` field of the array. Notice that you cannot use a loop for this because the GIFs all have different names. Assign the images to the elements in the order shown above: `BrownBox.gif`, `OrangeBox.gif`, `RedBox.gif`,`WhiteBox.gif`, `BlueBox.gif`, `PurpleBox.gif`.

B3. **Place The First Row of Display Grid**. Recall that to place the `WhiteBox.gif` on a Web page, JavaScript can be used to place the proper image tag in the source file with the command

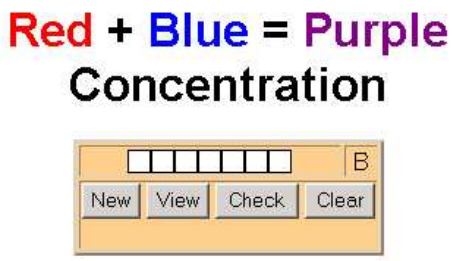
```
document.write(<img src="WhiteBox.gif">)
```

To create the Display grid we will need to iterate that command seven times to make a row, and to iterate that row operation seven times to create the whole 7×7 grid. (Check p. 616 for an example of making a row of images.) Creating a row of the Display grid is the purpose of this step.

The Display grid goes in the "A" position of the present Web page (A4), which is inside a table. In that table data element, replace the "A" with a pair of script tags, and insert JavaScript code to accomplish the following operations.

a) Write a WFI that iterates seven times using the iteration variable of B2b.
b) On each iteration of the loop, place an image tag for the `WhiteBox.gif` into the source file using the `document.write()` operation.

Check the page. It should look like this.

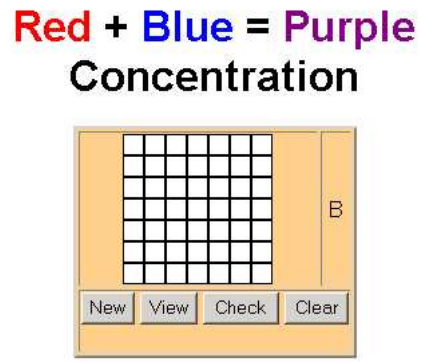


c) After the loop, place a `<br>` tag into the source file using another `document.write()`. This moves the HTML cursor to the next line.

B4. **Create the Display Grid**. With one row completed it is a simple matter to create the entire grid—just iterate the iteration of B3! By repeating the row-creation operation seven times we get the 7×7 grid.

To iterate the iteration, write a WFI *around* the loop of B3. That is, using the other iteration variable declared in B2b, define a WFI whose body is the iteration and the
 tag of B3.

Check the result. It will look like this.



B5. **Create the Key Grid**. The Key grid is created just like the Display grid was created, except that it replaces the "B" table element.

a) Perform the operations specified in steps B3 and B4 to create the Key grid with the `BrownBox.gif` image. This can be done by copy/paste/edit, but it is great experience to rewrite the loops by hand.

Check the result. The figure will look like the figure at the start of Part B, above.

B6. **Add Event Handlers to the Key Grid**. When the mouse moves over the Key grid the box beneath the mouse pointer is to turn from brown to orange. When the mouse moves off of the key, it turns back to brown. If the mouse is clicked the key turns from orange to red. Making these changes is the goal of this step.

For the grid of GIFs to become mouse keys, we must add event handlers to each of the images placed in B5. (See pages 620-623 in the textbook.) The event handlers will tell the browser what it is to do when the mouse is over a GIF. Three events will be useful for the Purple Concentration:

`MouseOver` is the event of a mouse being over a GIF,
`MouseOut` is the event of a mouse, having previously been over a GIF, to move somewhere else
`Click` is the event of a mouse button being clicked on the GIF.

We will use event handlers named `touch()` for the `onMouseOver` event, `untouch()` for the `onMouseOut` event, and `change()` for the `onClick` event.

In order for the events to cause the boxes to change color (be over-written with a different colored box), we will need to know where in the grid the box is. A box's position is given by its row and column position in the 2 dimensional grid. So, each of the three procedures will have a parameter for the row position and a second parameter for the column position: `touch(row, col)`, `untouch(row, col)`, `change(row, col)`. The `row` is the number of rows down from the top, 0-origin indexing, and `col` is the number of columns from left, 0-origin. Thus, the box at the upper left hand corner of each of the grids is row=0, col=0, and the lower right hand corner is row=6, col=6. This is convenient, because it means that row is just the iteration variable of the WFI that created the row (outer loop), and col is just the iteration variable of the WFI that created the box (inner loop).

The event handlers are included in the `<img src=…>` tags. (See how this is done with a single position parameter on p. 623 of the textbook. Notice that the event handler calls are constructed using concatenation.) These tags were written in B5. So return to that loop and revise the `document.write` to include the `onMouseOver` event by calling the `touch(<`*row*`>, <`*col*`>)` procedure, where <*row*> is the iteration variable of the outer loop and <*col*> is the iteration variable of the inner loop. At the same time, include the `onMouseOut` event calling the `untouch(<`*row*`>, <`*col*`>)` procedure and the `onClick` event calling the `change(<`*row*`>, <`*col*`>)` procedure, with the same parameter settings.

Check that the Web page still works. (It won't look any different, because the functions `touch()`,`untouch()`and `change()` haven't been written, but it is wise to check at this point since it is easy to mess up the construction of the event handler calls.)

B7. **Activate the Key Grid**. To make the Key grid to be active, we need to write the functions `touch(), untouch()` and `change()`. What are they supposed to do? The `touch(row, col)` handler should turn the brown box at `row`, `col` to orange, and `untouch(row, col)` should do the opposite and the `change(row, col)` function should turn the Key grid square to red, indicating it was clicked, and turn the corresponding Display grid to blue. To over-write the boxes, it is necessary to refer to them where they are stored in the `document.images` part of the Web page description. (This is explained under Redrawing Images on p. 598 of the textbook.)

There are three complications in referencing the grid images in `document.images`.

First, the images are stored in the order in which the browser placed them on the Web page, which means that the Display grid images are listed first, and the Key grid images follow. Because there are 49 Display grid images, they must be indexed as 0 through 48. Because there are 49 Key grid images and they come next, they must be indexed 49 through 97.

The second complication is that we have the `row` and `col` numbers of the box, but we don't have its index numbers. But we can find them because there are `row*7 + col`

boxes ahead of any box in its grid (check this!), which is it's index. And there are 49 additional boxes from the Display grid ahead of the boxes in the Key grid. So, to change a box in the Key grid we must over-write the box indexed by

```
49 + row*7 + col
```

stored in the `.src` field of `document.images`.

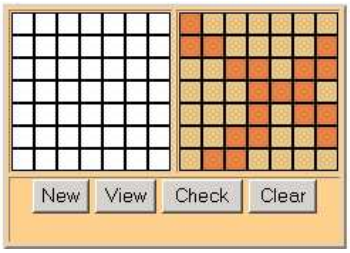
The third complication is that the over-write should use the images prefetched, but they cannot be referred to by their file names, so they're referred to by their index in the array into which they were prefetched. (Recall that you chose a unique name for this in B2b.) If the colors were stored in the order given above, then the orange box needed for `touch()` was prefetched into <*your unique name*>`[1].src` and the brown box needed for `untouch()` was prefetched into <*your unique name*>`[0].src`, etc.

After the table definition on the Purple Concentration, place a pair of script tags, and within them define the three functions, `touch(row, col)`, `untouch(row, col)` and `change(row, col)`.

a) `touch()` changes `document.images[49+row*7+col].src` from (the current) brown box to an orange box, that is `OrangeBox.gif`, which is stored in <*yourArrayName*>[1] in step B2.
b) `untouch()` changes `document.images[49+row*7+col].src` from (the current) orange or red box to brown, that is `BrownBox.gif`, which you stored in <*yourArrayName*>[0] in step B2.
c) `change()` changes `document.images[49+row*7+col].src` from (the current) orange box to red, that is `RedBox.gif`, which you stored in <*yourArrayName*>[2] in step B2. It also changes `document.images[row*7+col].src` from a (probably) white box to blue, that is, `BlueBox.gif`, which you stored in <*yourArrayName*>[4] in step B2.

The functions do not have to return any value. Their whole operation is to make the assignment to update `document.images` from the prefetched GIFs.

Check the result. With only the `onMouseOver` event defined it is possible to turn the boxes to orange but not turn them off. So sweeping the mouse across the Key grid in an "X" results in an image like this



When the `onMouseOut` event is defined, the orange will change back to brown as the mouse moves away.

B8. **Turn In**. Publish your new page in your Web space under a unique name.

a) Have your page published by 11:30 PM, Thursday, February 19, 2004.
b) Do not modify your page until after it is graded.
c) Print the HTML/JavaScript source, add your name, address and section letters, and turn it in in class February 20, 2004.

> In all parts of Project 2 you must comment the HTML and JavaScript that you write or change. The comments do not have to be long, but they must say what you've done in a way that is coherent to a human. *Uncommented HTML and JavaScript will not be graded.*

Project 2C Completion of Purple Concentration

The goal of this part is to complete the application begun in Part B. If you completed Part B, use your solution, but if not, get a working version of part B here.

The main task is to generate the random red square pattern and save it in an array so that the result can be checked. To create the random pattern will require the use of the random number generator as shown at the bottom on Figure 21.1 in the textbook.

C1: **Set Clear Event Handler**. The Clear button is to "white out" the Display grid. The function that will perform that operation (written next) will be called `whiteout()`.

a) Set the `onClick` event handler for the Clear button to be a call to the `whiteout()` function.

C2. **Write the `whiteout()` Function**. The `whiteout()` event handler clears all of the positions on the Display Grid. That is, it sets the first 49 images in `document.images` array to `WhiteBox.gif`, which was the value it was initialized to in Part 2B and is (probably) stored in the index 3 position of the image array, that is, <*yourArrayName*>`[3].src`.

a) Write the `whiteout()` event handler function as a loop from 0 to 48 assigning the `.src` field of `document.images` the white box. (Notice that you can also use a loop within a loop, each with seven iterations, as if visiting rows and columns; if `i` is the outer loop iteration variable and `j` is the inner loop iteration variable, the positions of `document.images` being updated are `i*7+j`.)
b) Place the `whiteout()` function with the others previously written.

C3. **Create a Random Pattern**. Include the `randNum()` function of Chapter 21 among the functions at in the Purple Concentration application.

Declare variables `squares`, initialized to 3, and two ten-element arrays, `probr` and `probc`. These two arrays will hold the random positions of the pattern, `probr` containing its row position, and `probc` containing its column position. The variable `squares` is the number of squares in the pattern. So, initializing the random positions will be a function, `setUp()` with three steps:

a) Clear the array by making a call on `whiteout()`.
b) Set up a WFI loop iterating `squares` times, and on the ith iteration assign a random number over a range = 7 (0 through 6) to `probr[i]` and a random number over the same range to `probc[i]`.
c) Display the random pattern with a call to `show()`, which is a new function written in the next part.

Notice that one shortcoming of the application (which we will not worry about) is that two random positions could be identical, lowering the number of squares below the `squares` count.

Make `setUp()` the `onClick` event handler for the New button.

Check your application.

C4. **Write the `show()` Function**. The `show()` function copies the random pattern into the pattern grid. It simply requires a loop of `squares` iterations in which on the ith iteration the square with row position `probr[i]` and the column position `probc[i]` is set to red. This requires that the index of the proper element in `document.images` is computed with the two values, and then set to the prefetched image, which is, by now, a standard operation.

Next declare a variable, `timerID`, and add to the end of the `show()` function a call to clear the pattern grid after five seconds. The timer call, as explained on p. 402 of the textbook, has the form

```
timerID = setTimeout("whiteout()”, 5000);
```

which will clear the pattern grid after the indicated amount of time.

Make `show()` the `onClick` event handler function for the View button.

Check your application.

C5. **Allow the User to Pick**. There are really two steps. When the user clicks on a Key grid square, the corresponding Display grid square must be set to blue. This was done by the `change()` function described above. But, if the user clicks a second time, the square should return to white, i.e. it is an undo. Second, we must record the square the user clicked so we can check it later. Making these changes requires the following steps:

a) Declare a variable `setOrNo` as an array with 49 items, because any of the Disiplay squares can be set.
b) We will use a value of 0 to indicate the square has not been set and 1 to indicate that it has been. Initialize the `setOrNo` array to zero in the loop that you wrote in step C2. (This is a good logical place to initialize `setOrNo` because the squares are being set up to be white, and a 0 initialization will indicate that.)
c) Change the `change(row, col)` event handler function. Rather than displaying blue every time, use an if-statement to check to see if `setOrNo[row*7+col]` is equal to 0. If it is, set the square to blue, as usual, and set `setOrNo[row*7+col]` to 1. If not, it is already blue, so set it to white, and clear the bit at `setOrNo[row*7+col]` to zero.

Test out your solution and be sure that two clicks on a square set it back to white.

C6. **Write the `check()` Function**. To check how the user has done with the concentration challenge, we simply go through the pattern and see if the position is set by the user or not, as indicated in the `setOrNo`. If it is set, then the user got the position, and we should draw it purple. If the position is not set, it should be drawn in red to show it was part of the original pattern. The steps are

a) Create a WFI for `squares` iterations.
b) On the ith iteration, use an `if` statement to see if the element at index `probr[i]*7 + probc[i]` of `setOrNo` is set.
c) If the position is set (1), fill the corresponding position of the Display grid with the purple square, which has been prefetched in Part 2, and is (probably) stored in the image array at index position 5.
d) If the position is not set (0), fill the position with the red square.

Make `check()` the `onClick` event handler function for the Check button.

The application is complete! Congratulations. Try it out.

C7. [Extra Credit] **Add Controls**. The controls are input buttons and windows for setting the number of squares in the pattern and setting the amount of time they are displayed. (See the original version of the application at the beginning of this description.) These controls are not needed for operation of the application, but they make it more useful. For extra credit, add them!

First, extend the form in the second row of the table to include all of the controls shown.

Next, change the number of squares in the pattern. Doing so requires the event handler to add or subtract one from `squares`. Make the upper limit 10 (that's all the space there is in `probr` and `probc` arrays to save pattern positions), and the lower limit 1. It is suggested that you use a function, of the form,

```
squares = Math.min(10, squares+1)
```

which keeps `squares` from exceeding the limit. `Math.min` returns the smaller of its two arguments. For subtracting, there is also a `Math.max`.

Finally, control the time of the application. This will require the use of the `Math.min` and `Math.max` functions, and a new variable, `duration`, initialized in its declaration to 5000. The timer setting operation in C4 must have the 5000 parameter replaced with `duration` in order have the variable amount of time be set. The event handlers for increasing and decreasing the time must work in units of 1000, since time is measured in milliseconds.

Try out the full application!

C9. **Turn In**. Publish your new page in your Web space under a unique name.

a) Have your completed page published by 11:30 PM, Thursday, Feb. 26, 2004.
b) Do not modify your page until after it is graded.
c) Print the HTML/JavaScript source, add your name, address and section number, and turn it in in class February 27, 2004.