

Name: _____

ID: _____

Midterm

CS193C, Summer 2016, Young

As we've discussed, you have three hours to complete this exam and get it submitted. Please get it online by 9:30pm. Use the same submission procedure as for your homework assignments. Make sure your files are up at 9:30pm. If your files are more than a few minutes late, you can expect to get a zero.

This test is open book, open note. You may use the Internet to access reference material, but may **not** communicate with anyone in any way (electronic or otherwise) other than the CS193C teaching staff.

The Stanford Honor Code

1. The Honor Code is an undertaking of the students, individually and collectively:
 - a. that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
 - b. that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.
2. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.
3. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work

I accept the letter and the spirit of the Stanford University Honor Code. I swear that I have not given or received unpermitted aid on this exam, and I have taken an active part in stopping any violations of the Honor Code which I see on this exam.

Signed: _____

Instructions

- **Make absolutely sure you turn everything in on time. Do not turn in your exam late.**

This exam is due at **9:30pm.**

- We will be testing your exam solutions in either Firefox or Chrome, **your choice**. If you want us to test in a specific browser add a paragraph at the top of each webpage telling us which browser you want us to use.
- **We are interested in running code only.** If your code doesn't work, you should expect to get no points. Whatever you turn in needs to run. It doesn't need to run perfectly, so if the numbers generated aren't quite correct, or if things don't move the way the problem asks, you can certainly get partial credit. But we aren't giving partial credit for unfinished code that doesn't actually do anything in the web browser.
- **You may not use jQuery or any other library code.** If you are found using jQuery or other libraries on a problem, you will not receive credit on the problem.
- If we do not provide values for items you need, any reasonable value is acceptable. For example you may make up your own alt values for images or set widths and heights as desired when they are not specified in the problem statement.
- If desired you may use the convenience innerHTML property along with any other techniques which are supported by whichever web browser you've asked us to test on, even if they are not part of the official W3C standard.
- Make sure everything validates.
- You may not use Dreamweaver or other WYSIWYG editors.
- All code must be developed from scratch – this means you may not copy directly from code you find on the Internet. You may use CS193C lecture examples I have uploaded to Canvas directly.
- If in doubt, cite what you've done by explicitly putting a paragraph visible on your webpage and it won't be considered an honor code violation, although you may lose points. You can put this directly below the "grade in Chrome" or "grade in Firefox" paragraph.
- As noted in Handout 2 "Computer Science and the Stanford Honor Code" we reserve the right to use automatic plagiarism detection, comparing your solution to other students' solutions and to code on the Internet (the Judicial Affairs office has ruled that this is permissible).
- All problems are worth the same amount of points. Although some are definitely harder than others. Plan accordingly.
- *I'm not expecting most students to complete all the questions. So don't panic if you don't think you're going to finish the exam. This is expected.*
- **Keep a working backup copy of your code at all times. When you have successfully implemented a feature, create a backup copy of your files. You do not want to be rushing to add a new feature to a problem when time runs out and discover that you've broken your previously working webpage and don't know how to get it working again.**
- **Make sure you've double and triple checked your submission zip file before submitting it, it should contain all files needed to run your programs, including any files we've provided.**
- **Starter files for this midterm may be found at:**

<http://cs193c.stanford.edu/111/starter404.zip>

Mad Lib

A Mad Lib is a word puzzle in which the player is provided a sentence with some blank spaces in the middle of the sentence. These blank spaces are marked with their grammar type. For example:

The Noun told the Noun to Verb across the Noun !

The user fills in the blanks which can either lead to a story which makes sense or which is nonsense. In our example depending on our choices for the blanks, our sentence could end up being:

The teacher told the student to walk across the road!

Or

The rocket told the aardvark to eat across the moon!

See: https://en.wikipedia.org/wiki/Mad_Libs for more information.

We're going to create a webpage which allows the user to create mad libs. Here's what the webpage looks like with all the data filed in:

Mad Lib Generator

file:///C:/Users/pyoung/Documents/

Search

Mad Lib Generator

Entry

The Noun told the Noun to Verb across the Noun !

Make Mad Lib

Mad Lib

The told the to across the !

Show Result

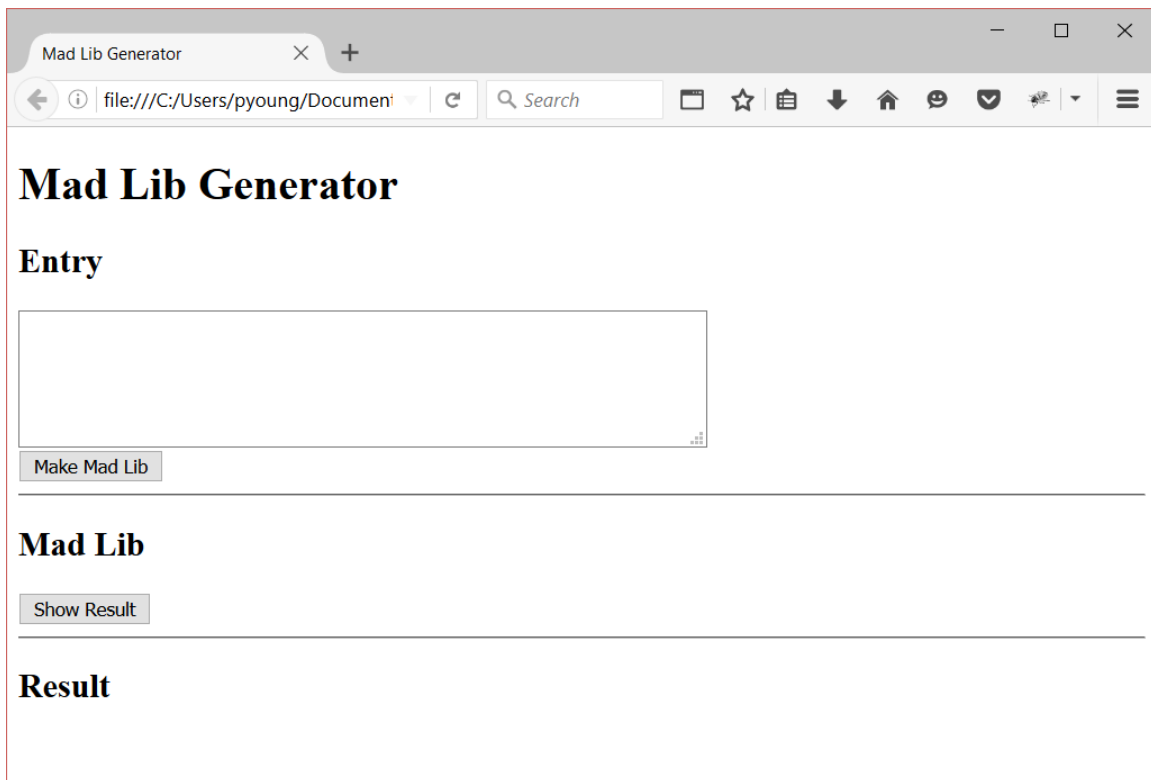
Result

The rocket told the aardvark to eat across the moon!

The webpage consists of three sections:

- An initial entry section in which the user enters the specification for their Mad Lib.
- The actual Mad Lib puzzle created from the user's specification.
- The resulting phrase using the text entered in the puzzle blanks.

Initially here's what your webpage should look like:



The screenshot shows a web browser window with the title "Mad Lib Generator". The address bar shows a file path: "file:///C:/Users/pyoung/Documenti". The browser's toolbar includes a search bar with the text "Search" and various icons. The main content area has a large heading "Mad Lib Generator". Below it is a section titled "Entry" with a large text input box. Under the input box is a button labeled "Make Mad Lib". Below this is a section titled "Mad Lib" with a button labeled "Show Result". At the bottom is a section titled "Result".

User interaction with your webpage will also proceed in several steps:

1) User Enters Mad Lib Puzzle

The user will enter in a puzzle in the entry area. The puzzle may contain zero or more blanks for the puzzle. These blanks will be designated using the underscore '_'. Between each pair of underscores is the type of word expected to fill the blank. Here's our sample input:

The _Noun_ told the _Noun_ to _Verb_ across the _Noun_!

Once the user's happy with their entry they hit the "Make Mad Lib" button which generates a Mad Lib which shows up in the next section.

2) Puzzle Created for User Entry

When the user clicks on the "Make Mad Lib" button your code will generate a new puzzle in the middle section allowing the user to enter in their word choices. Here's what the Mad Lib for our entry will look like before the user fills in the blanks:

Mad Lib

The told the to across the !

Notice each blank specifies what type of word should be entered into it. Use the placeholder attribute on your text fields to specify the type of word. [Google “HTML5 placeholder” if you need more information on what the placeholder attribute does.](#)

3) User Fills in Each of the Blanks

In this case the user is asked to provide three nouns and a verb. They fill in the four blanks in the form your program has generated:

Mad Lib

The told the to across the !

After the user has entered text into each of the blanks, they will click “Show Result” button. Note that your program is not checking what the user has entered in the text fields. If they put in a noun where your program asked for a verb, go ahead and let them do it. If they leave a field blank, that’s okay, their results will just be missing a word.

4) Results Generated and Displayed

When the user clicks on the “Show Result” button the last section of the webpage is filled in with results. Which is the text of the puzzle combined with what the user entered into each of the blanks:

Result

The rocket told the aardvark to eat across the moon!

Key Specifications

- Label the sections using h1 and h2 elements as shown.
- I’ve used the `<hr />` tag to put lines between each section to emphasize the parts of this problem. However, you do not need to have these lines there.
- You may assume that the user does not enter in any tabs or carriage returns in their Mad Lib specification. You may also assume they do not enter the ‘<’ or ‘>’ characters.
- You may assume that if an underscore ‘_’ is present it has a matching underscore. However, do make sure that your program works if the user does not enter any underscores at all.
- There are no restrictions on what the user enters between the underscores, except that they can’t use an underscore or a carriage return in the type. So if the user wants to specify a blank as:

`_Stanford Sports Teams_`

or

`_Amazin’ Fricken Great Stuff!!!#!_`

these are both okay. If the type specification is so long that it doesn’t fit in the text field and gets cut off by the web browser that’s okay too.

- As you can see from the screenshots, I’m not too concerned on this problem with your forms looking very neat. Just get the buttons in there where the TAs can use them for grading. You don’t need to line them up nicely. The text fields for the blanks however should be show in line with the puzzle text as shown in the screenshots.

- You don't need to modify the length of the text fields, just use the default size.
- If there are results from a previous run, don't forget to clear the Results section when the user clicks on the "Make Mad Lib" button to generate a new puzzle.

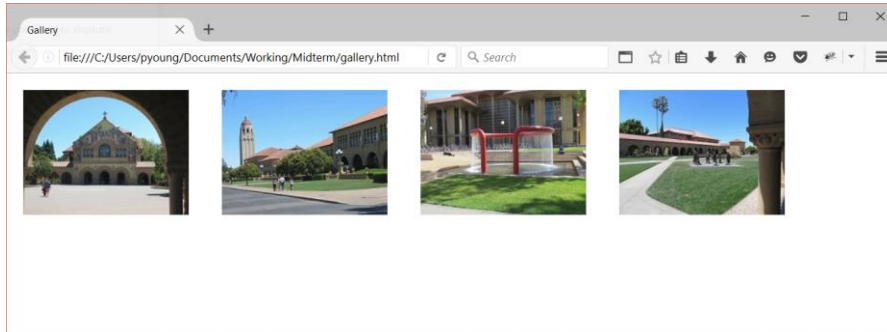
Please save the main HTML file under the name "mad-lib.html". If you have additional JavaScript files, feel free to name them whatever you want.

Implementation Comments

- If you're familiar with JavaScript strings feel free to use some of the more advanced methods on them. But if you don't, you should be able to do this problem easily using just the `charAt` method, the `!=` and `==` comparison operators, and the `+` concatenation operators combined with your basic looping operations. If your web browser stops responding. You probably have an infinite loop.

Gallery Reordering

For this problem, we will create a gallery of images and allow the user to reorder images in the gallery by dragging them around. Here's a screenshot:



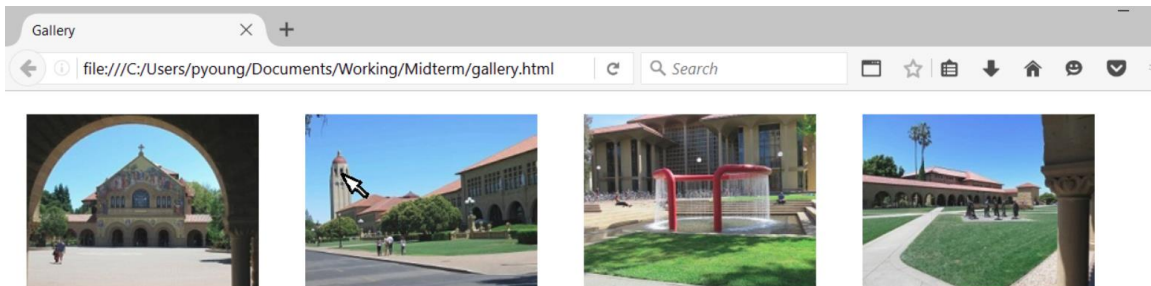
The images to use will be listed in an array provided in the midterm downloads called "photos.js". Load this file into your HTML file. Do not modify the format of this file. In other words, you must keep the array format and the name of the array --the TAs may replace the "photos.js" file with a different "photos.js" file with the same format and variable name, but different images and your program should continue to work correctly, so make sure that your program works if more items are added to or if items are removed from the array in the "photos.js" file. I've provided some additional images you can add to the array for testing.

The images should be placed in a horizontal line as shown. The images will be placed 20 pixels below the top of the window. The left most image will be placed 20 pixels from the left side of the window. Each additional image will be placed 40 pixels to the right of the preceding image. Each image will be 200x150 pixels. These margin amounts and image dimensions are fixed and you may hard code them into your program.

Images can be swapped by pressing the mouse button down on one of them and moving it and then releasing the button when it is on top of another image. This is a move operation, not a traditional drag operation and follows the same rules as moving the map in your homework assignment (feel free to copy code from your map solution).¹ When the user releases the mouse button, if the image being moved is over one of the other images, or within the 20 pixel margin of one of the images, it switches places with that image.

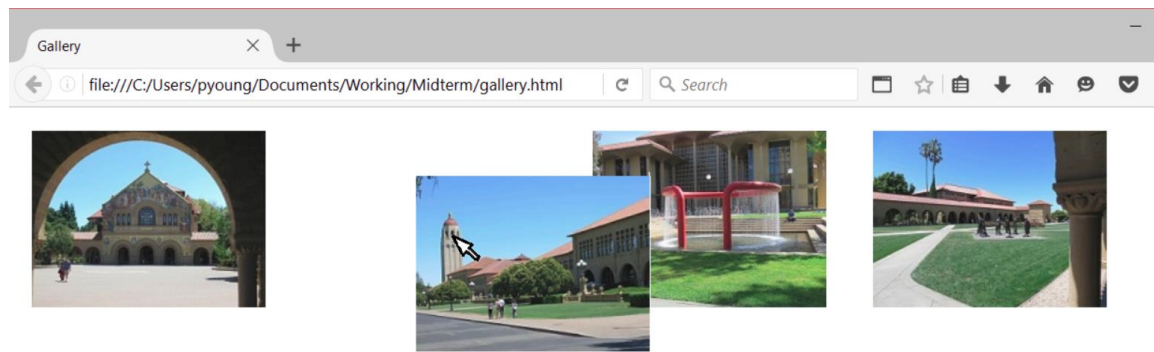
Example

Here I move my mouse cursor over the top of Hoover Tower and I press and hold the mouse button down:

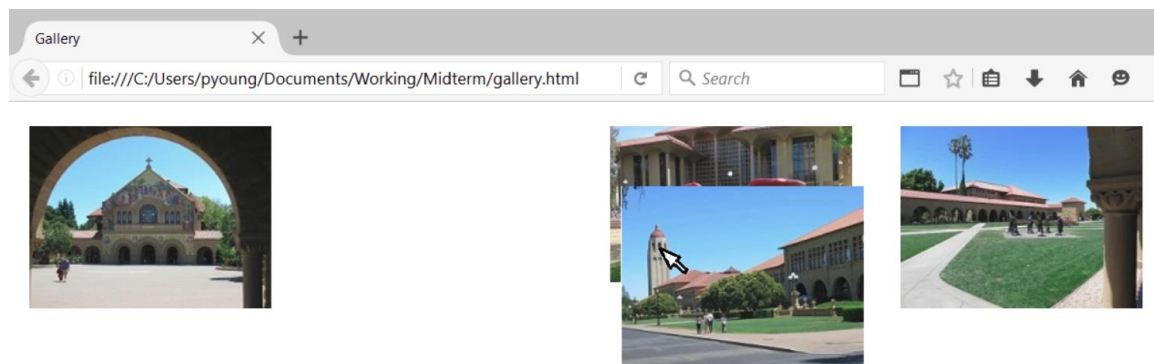


¹ As I've mentioned in class in a traditional drag operation, a translucent version of the item is moved, while the original item remains visible in the original location. Try for example dragging a file around on your computer's desktop. You'll see two copies of the file, the original copy and a translucent copy you are dragging. **For this problem we are not using a traditional dragging operation. Instead we are actually moving the item.** This should work exactly the same way you allowed the user to move around the map in the homework assignment.

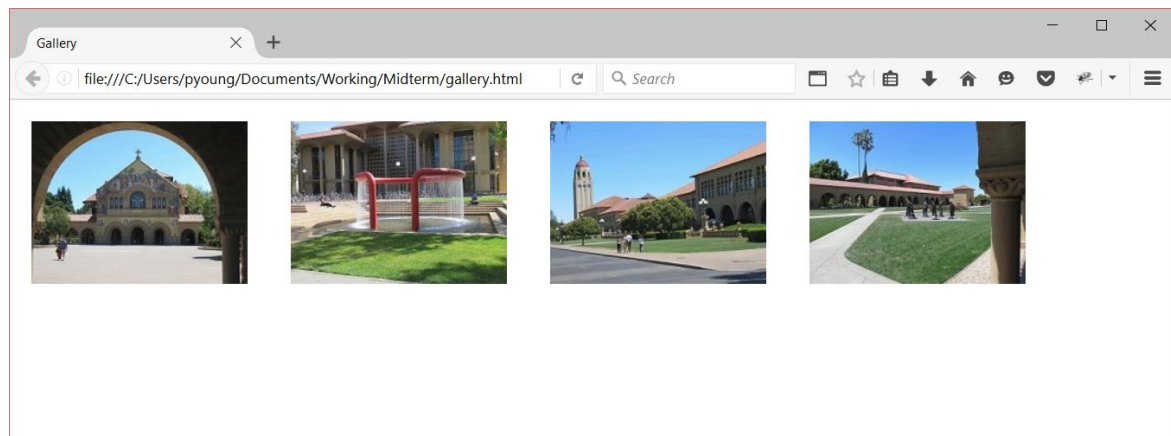
With the mouse button still pressed down, I move the image towards the right.



Depending on where I release the mouse button, either the Hoover Tower image will return to its original position, or it will be swapped with another image. In this case I continue the drag operation until the drag point is on top of the Hoop Fountain image:



When I release the mouse button the Hoover Tower replaces the Hoop Fountain image. The Hoop Fountain image switches to where Hoover Tower's image *was originally placed* before the move operation began resulting in this:



Please save the main HTML file under the name “gallery.html”. Include the original “photos.js” file with your submission. If you have additional JavaScript files, feel free to name them whatever you want.

Implementation Comments

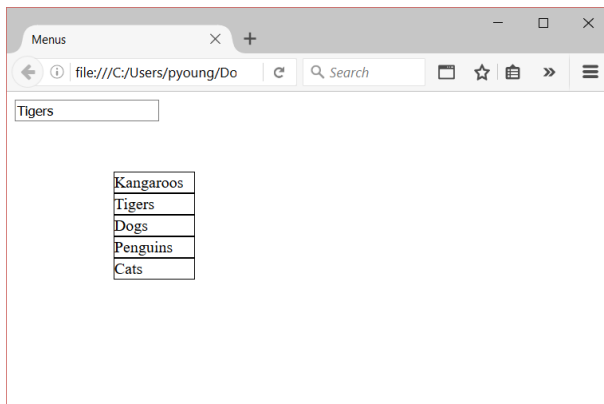
- Make sure that the image moves relative to the original drag point. For example, if I click on the top of Hoover Tower and start to move that image, the top of Hoover Tower should remain directly under the mouse until I complete the move operation.
- Use the point directly underneath the mouse when the mouse button is released to determine whether or not an image should be swapped. If the mouse point is on the image, or within 20 pixels of an image go ahead and swap it.
- If the mouse is released and you are either on the moving image’s original location or within 20 pixels of that original location go ahead and just snap the image back to its starting location.
- If the mouse is released and it is not within 20 pixels of any of the images, snap the image back to its starting location.
- If you’re dragging an image and it goes underneath instead of on top of another image, that’s okay for this exam. (The correct solution to this problem is to modify the z-order when clicking on an image.)
- You may assume that during grading the TA will always keep the mouse within the bounds of the window.
- You may assume that the window is always big enough to display all the images along with the 20 pixel surrounding margin.

Nested Popup Menu

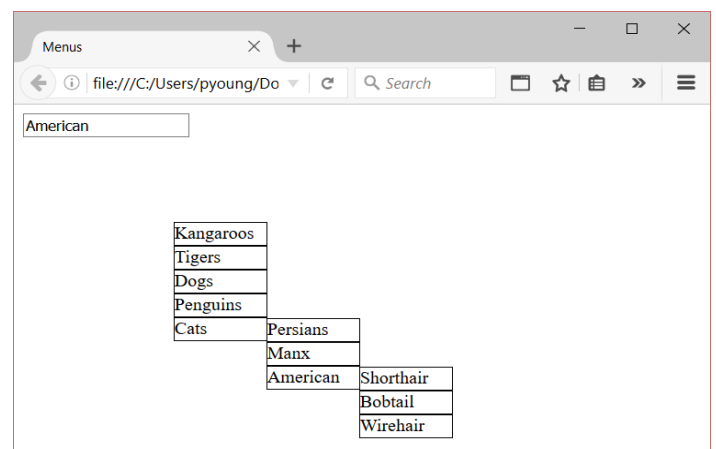
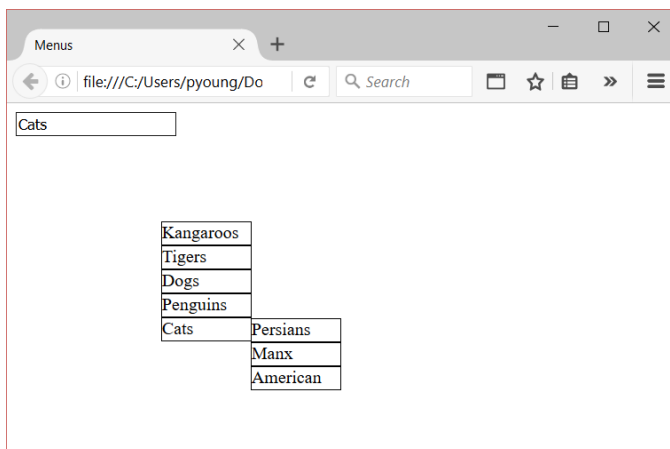
For this problem we will take a nested structure and create a popup menu from that structure. You will demonstrate that you can determine when the user clicks on parts of the popup menu by changing a text field which you will place in the top-left corner of the webpage. The actual structure used to define your popup menu can be found in the file “lists.js”. Load this file into your HTML file. Do not modify the structure of this file, you can assume that during grading the TAs will replace the “lists.js” file with their own copy which uses the same structure formats but has different contents. Your program should continue to work if the TAs add additional items including items with deeper nesting than the example provided.

I have provided two lists in the file. One of the lists has a nested structure, whereas the other list is just a flat list with no nesting. If you’re feeling confident, go ahead and solve this problem for the nested list structure. If you’re not feeling confident, I strongly recommend trying to solve a simpler version of this problem assuming no nesting. If you get that version completed, keep a backup copy of it and be prepared to turn in that version if you can’t get the more complex version working. Nesting is worth ~33% of the points for this problem. Remember, most students are not expected to complete this exam. The nested version of this program is fairly complex given your 3-hour time frame.

Here’s a screenshot showing the program in action. Initially the webpage is empty, except for a blank text field in the upper-left. I tell my program I want to display the menu by left-clicking anywhere on the screen. The menu then appears at that point. Once the menu has been deployed I click on the menu item labelled “Tigers” and in response the text field in the top left hand corner displays the word “Tigers”.



In the example below at left, I have clicked on the menu item “Cats”. If you’re doing the nested version of this program “Cats” has a nested set of submenus. Clicking on “Cats” changes the text field in the left corner to show “Cats” and also deploys the submenus. In the example below on the right, clicking on “American” deploys the next level of menus and changes the text field to “American”:



Each of the menu items should be 80x20 pixels. You may assume that the text for each menu item will always be small enough to fit in the 80x20 pixels space. The menu items have a 1-pixel black border.

When the menus are not displayed, left-clicking anywhere in the window will display the menu. The top-left corner of the menu will appear directly under where the mouse was clicked. The menu will remain up until explicitly dismissed (see below). If the user clicks on a menu item, the text associated with that menu item will appear in the text field in the top-left corner of the webpage. The menu will remain up, even after the menu item is clicked on, and the user can continue to click on menu items changing the text in the top-left.

If the user clicks on a menu item which has a submenu associated with it, the text associated with the menu item will appear in the text field as usual, however, in addition the submenu will be displayed lined up to the right of the menu item.

When a submenu is displayed, clicking on an item in that submenu will place the text of that submenu item into the text field. The menu and any submenus will remain displayed.

If the user clicks on a menu item in a higher level menu while a submenu is displayed the submenu will disappear. For example, if the “Shorthair / Bobtail / Wirehair” menu is displayed and I click on “Persians”, “Manx”, or “Americans” then the “Shorthair / Bobtail / Wirehair” submenu will disappear. If I click on any of the main menu items such as “Penguins” both the “Persians / Manx / Americans” and the “Shorthair / Bobtail / Wirehair” submenu will disappear. This is true even if the user clicks on the menu item that originally deployed the submenu. So clicking on “Cats” over and over again will deploy and retract, deploy and retract the “Persians”, “Manx”, or “Americans” submenu.

Submenus are always retracted when their parent menu is removed. So if both the both the “Persians / Manx / Americans” and the “Shorthair / Bobtail / Wirehair” submenus are displayed and I click on “Penguins” (which removes all the submenus) and then click on “Cats” only the “Persians / Manx / Americans” will appear.

If the menu structure is deployed and the user left clicks anywhere other than on the menu itself (or its submenus), the entire menu structure will disappear. The user can bring the menu back up by clicking again anywhere in the window. The menu should reappear under the new mouse point, and any submenus which had been previously deployed should be hidden.

Note we are only responding to left-mouse clicks. Ignore right-mouse clicks, center-mouse clicks, and the mouse wheel.

Please save the main HTML file under the name “menus.html”. Include the original “lists.js” file with your submission. If you have additional JavaScript files, feel free to name them whatever you want.

Implementation Comments

- If you’re solving the nested version of this program use the “list” variable to create your menu. If you’re not solving nesting, use the “listShort” variable to create your menu.
- You may assume that the grading TA will always click in a location which has enough space to display the entire menu including submenus. Don’t worry about what happens if there isn’t enough space to deploy the menu.