

## Assignment #4

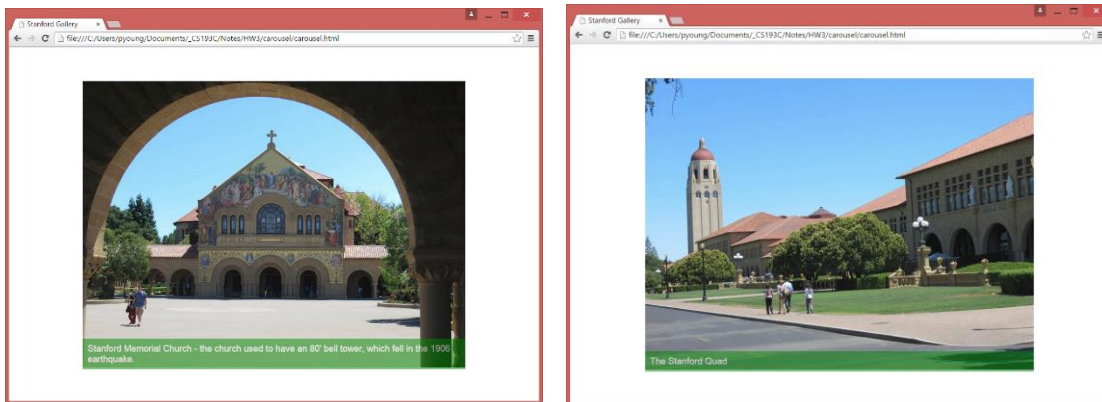
# JavaScript

### CS193C Summer 2020, Young

We now get a chance to explore some of JavaScript's more advanced features. All problems should work in Firefox and Chrome. This assignment is due Tuesday August 4<sup>th</sup> at 1:30pm.

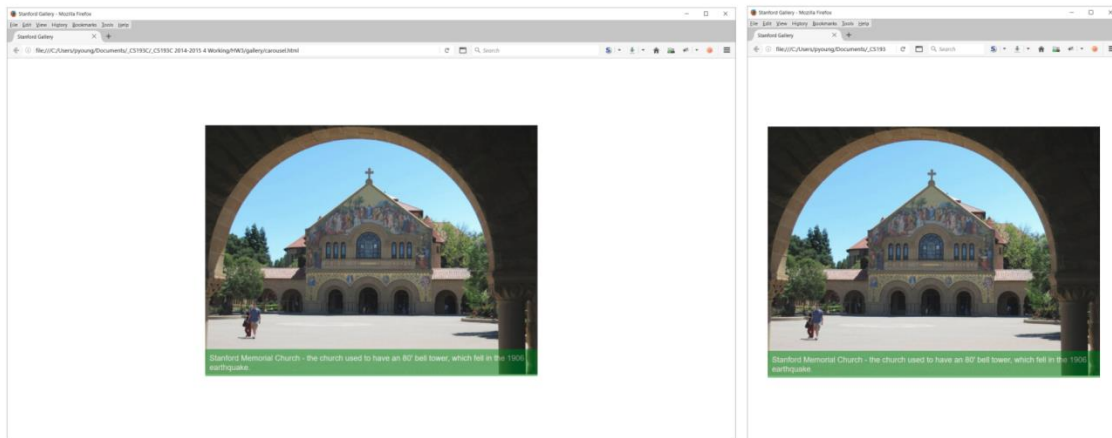
#### Photo Gallery

For this next problem we will create a simple photo gallery which allows the user to rotate through a number of images. Here's some samples of the webpage in action:



Clicking on the right-hand side of the image will change the picture and the caption to the next image on our list. When we reach the last image on our list, we will begin again with our first image. Clicking on the left-hand side of the image will take the user to the previous image on our list—when we get to the first image, we will wrap to the last image.

The image will always appear in the exact center of the window. When the user resizes the window the image will be moved to the center of the window. The image will not change size if the user increases the size of the window. Instead it will always remain centered at its native 800x600 size. You may hard code in the image size. You may assume that the window always remains larger than 800x600, so you do not need to worry about the case where the window is smaller than the image.



The caption will always appear at the bottom of the image. Note that the caption is in white with a green background #008800 and notice that the image shows through the caption box because opacity of the caption style property is set to 0.5.

We will be using absolute placement to get things working. Remember that in spite of the name "absolute", placement of an item using "absolute" placement is actually relative to the item it is contained within. This if a div A contains a div B, and both A and B are absolutely positioned, the left, top, right, or bottom settings of B are actually relative to div A. If we move div A, div B will also move.

While there are several ways to create a gallery with the behavior we want, probably the most common is to overlay empty divs on top of the image. We will work with four different divs:

- We have a parent div which I will refer to as the photoSection. This div contains everything associated with the image. This includes the actual <img> tag and the other divs described below. Place this absolutely initially in the middle of the window when the page loads. Change its position when the user resizes the window. See below for more information on how to do this.
- We have a caption div. The caption div will be a child of the photoSection. If you absolute position the caption div, it will actually be placed relative to the parent div in which it is contained. While typically we position an item using left and top, we can also set its position using bottom. If you set bottom to 0px, the caption div will be aligned at the bottom of the photoSection.
- We will have left and right overlay divs on top of the <img> and the caption div. You can also place these absolutely. Again as these are contained within photoSection, they will move with photoSection even though they are absolutely positioned. Set left to 0px for the left overlay and right to 0px for the right overlay.
- Place onclick handlers on the left and right overlays which change the image and caption either backward or forward as appropriate.

### Handling Resizing of the Window

For the gallery problem, **please handle resizing using JavaScript**. The main purpose of this part of the assignment is to give you a bit of practice programming CSS positioned elements from JavaScript before tackling the much harder Map problem, and solving the resize using CSS would not give you any practice.

You can cause a function to execute when the window is resized by adding a listener waiting for the window's resize event. For example:

```
window.addEventListener("resize", handleResize, false);
```

will cause the web browser to call the handleResize function when the window is resized.

To get the size of the window use the following:

```
window.innerWidth and window.innerHeight
```

Once you get the new window size, use the techniques we learned in class (and found in the Dynamic Contents handout) to change the location of the photoSection. Don't forget when you assign a new top, left, height, or width to an element's style properties you must include a "px" at the end of the string.

```
elem.style.width = 5;           // wrong, wrong, wrong

elem.style.width = "5px";      // right
```

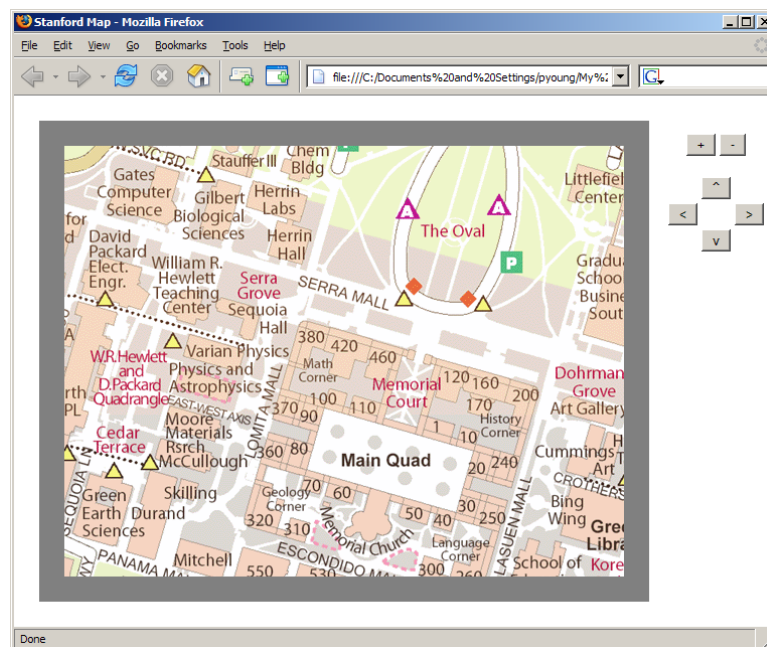
## Provided for You

For this part of the assignment I provide you with 6 jpeg images (each 800x600 pixels) and an HTML file. The HTML file has the correct div structures, but does not do any of the placement. You'll need to do that on your own. I also setup a JavaScript array with the information you'll need for the images (including the captions).

There is one important item that is missing from our implementation that you should add, should you decide to use this on a real website. There is no indication of how this webpage works. In HCI (Human-Computer Interaction) parlance there is no "affordance" indicating how someone might manipulate this webpage. If you use this on an actual website you might consider adding permanently visible right and left buttons (these could go on the caption, below the entire photoSection, or just outside of the photoSection on the right and left). An even fancier way to do this is to add right and left indicators on top of the left and right previous/next image overlays. You can take advantage of the onmouseover and onmouseout events so that these overlay indicators are only visible when the mouse is moved on top of their corresponding left or right overlay div.


## Stanford Maps

We will now create a Stanford-specific version of Google Maps. Make sure you've completed the gallery part of the assignment first. It's much easier and is designed to give you some practice with techniques you'll need for Stanford Maps. Here is a screenshot:



Your webpage should support the following functionality:

- The map should be enclosed within a map frame (as shown above). The size, color, and style of the frame is up to you.

- Allow the user to click and drag on the map to change the map area which is currently visible within the map frame.
- Set the cursor to the move cursor  when the user begins a drag operation. Return the cursor back to the standard arrow cursor when the drag operation is complete. You can set the cursor using the cursor style property.<sup>1</sup>
- When the user double-clicks on the map move the point double-clicked to the center of the view area.
- The map frame should resize as the window is resized. As the window is enlarged, the map area should enlarge, as the window size is reduced the map area should be reduced. The map frame should maintain fixed margins on all sides. These margins should be maintained as the window is resized. The exact margin sizes are up to you—choose something which you find aesthetically pleasing.
- The map should support zooming in and out. As the map zooms, the point in the center of the view area should stay fixed. In other words, if the map is centered on the Gates Computer Science building, the Gates building should stay in the center regardless of the zoom level. We'll discuss how zooming works in more detail below.
- Provide controls to scroll left, right, up, and down. Left or right scrolling should cause the window to scroll 1/2 of the total width currently visible. Scrolling up or down should cause the window to scroll 1/2 of the total height currently visible.
- To keep things simple, if the user scrolls off the edges of the map it's okay to go ahead and let them keep scrolling even if it means you can no longer see the map.

## Creating the Map and Map Frame

The map itself is represented by a GIF file. Create a `<div>` for the map frame. Put an `<img>` tag for the map within the map frame's div. Set the map frame's overflow style property to hidden. The overflow property controls what happens when the contents of an element do not fit within the containing element. Setting the overflow property to hidden tells the web browser to hide any sections of the contents which overflow the map frame.

## Resizing the Map Frame

You may either use the same technique used for the gallery to resize, or if you want to you can use CSS (hint if you're using CSS and need to get the current location of the frame, use `getBoundingClientRect`). You may assume that the user never resizes the window so small that there is not space for the map, map frame, and navigation buttons.

## Dragging the Map

You should be able support dragging the map around via careful use of `mousedown`, `mousemove`, and `mouseup` events. However, a couple of points will be helpful.

These next two paragraphs are very **important**. If you ignore these you may be in for great frustration. In your `mousedown` handler if you want to move the map, you must call `preventDefault()` on the event object. Otherwise the standard default behavior supporting a drag and drop to a separate window will run. You do not want this to happen.

---

<sup>1</sup> You'll only have complete control the cursor while it's over your map or map frame. When the mouse moves over other HTML elements, they will temporarily override the cursor based on their own style settings, even if you set the cursor on the `<body>` tag. If you experiment with Google Maps, you'll discover this is the same behavior they have.

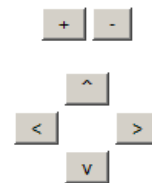
Place your event handlers on the document, not map (or whatever specific element you are trying to drag). Placing the handler on the actual element can cause problems if the user moves the mouse faster than you can move the element (if the mouse moves outside of the element you will lose mousemove and mouseup events). In theory you can place the handlers on the <body> tag. However, in my experience you're better off putting it on the document itself. You can do this programmatically like this:

```
document.addEventListener("mousemove",handleMouseMove);
document.addEventListener("mousedown",handleMouseDown);
document.addEventListener("mouseup",handleMouseUp);
document.addEventListener("dblclick",handleDbClick);

window.addEventListener("resize",resizeFrame);
```

### Navigation Rosette

As previously noted, you need to provide controls for zooming in and out and for scrolling in each of the four cardinal directions (up, down, left, right). The easiest way to provide controls is simply creating push buttons using <input> tags inside an HTML <form>. You can see a closeup of my controls at right.



If you want to get fancier, you can replace the buttons with images (or use the <button> tag to combine a pushbutton with an image). If you're feeling more ambitious you can move the navigation buttons onto the map (actually this is not that difficult). You can also create a zoom slider control as on Google Maps.

### Zooming

We support zooming by having four separate GIFs—"map-xl.gif", "map-l.gif", "map-m.gif", and "map-s.gif". We swap back and forth between the GIFs as the user zooms in and out. You'll need to do some careful arithmetic combined with setting the position of the GIFs to make sure that the same map point stays in the middle of the viewing area as the user zooms. Don't forget to preload the map images for smooth operation when the user first starts zooming.

If you want to get some extra practice, here are some things you can try adding:

- Add a text field and allow the user to enter in building names. Center on the building or if you want to get fancy add a flag or map pin.
- The default application can instantly jump from one location to another in response to double clicks or scroll requests. A more gradual scrolling may help the user by allowing them to see the relationship between the original location and the new. Support smooth scrolling when the user double clicks or clicks on a scroll button.

### Credits

Our map is from the Stanford Maps and Records Department. The original Map was from:

<http://www-facilities.stanford.edu/maps/download/TransportationMap.pdf>