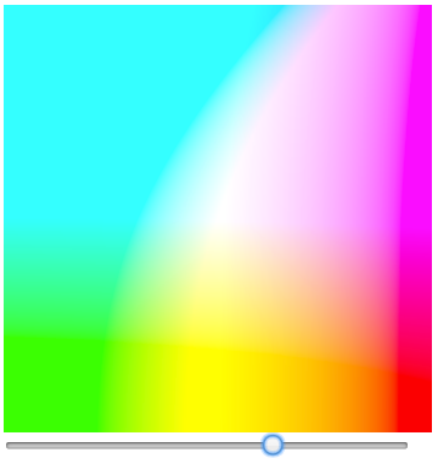
CS/INFO 3300; INFO 5100
Homework 7
Due 11:59pm **Tuesday** April 23

Goals: Implement force-directed layouts.

Your work should be in the form of an HTML file called index.html with one <p> element per problem. Wrap any SVG or CANVAS code for each problem in an element following the <p> element(s). For this homework we will be using d3.js. **In the <head> section of your file, please import d3 using this tag:**

```
<script src="https://d3js.org/d3.v5.min.js"></script>
```

Create a zip archive containing your HTML file plus **ALL associated data files** and upload it to CMS before the deadline. You will be penalized for missing data files or non-functional code. Check carefully for any issues with your file paths.



1. In Homework 6, you created a grid of circles to show colors across the HSL color space. In this problem, you will create a **smooth, square gradient for the LAB color space using a Canvas pixel array**. The LAB color space is a bit tricky. While L, luminosity, is fairly self-evident, A and B have specific perceptual definitions that are not as clear as Hue and Saturation might be conceptually. To get a sense for how A and B work, you will design an interactive tool for exploring different colors. **A and B will be mapped to the x- and y-axes of the canvas respectively**, and a **slider will allow the user to adjust the luminosity** of the canvas.

A. Below your P tag for this problem, place a **square Canvas element that is 320px in width and 320px in height**. In a <script> tag, write code to select the Canvas element and get a 2d drawing context. Using your drawing context, **generate an ImageData object** for the entire canvas and assign it to a variable named **image**. Assign the **.data attribute from the ImageData** object to a variable named **pixels**. (this ought to contain raw data for all of the pixels currently on the canvas, as seen in class).

B. Your square canvas will show **different values for A across the x-axis** and **different values for B across the y-axis**. For both A and B, allow **a range of values between -160 and 160**. Create a **d3.scaleLinear object** that maps **pixels in the image** (i.e. 0 to the width/height of the canvas) to this **-160 to 160 range**. Write a function, **rgbAtLocation(lum, x, y)**, which **returns a d3.rgb() object** containing the **correct color for that canvas location and luminosity value.** The parameters correspond to the **current luminosity value of the slider, x-axis pixel location in the canvas**, and **y-axis pixel location in the canvas**. Inside the function, first create a **d3.lab()** color object for the provided (lum, x, y) values (hint: use your scale to convert x pixels to A and y pixels to B). Finally, use **d3.rgb()** to convert the **lab color object to an rgb color object**.

C. Create a function, **fillCanvas(lum),** that uses the **pixel data** you gathered in step A, your **drawing context**, and **putImageData** to **fill the canvas with the correct LAB colors** for a given luminosity value. Use two loops to **visit every (x,y) location** in the canvas. For each (x,y) location, use your **rgbAtLocation** function to get a d3.rgb() object for that location. Using the d3.rgb() object, **adjust the value in the pixels array for each (x,y) location so that it contains the correct R, G, B, and A for the location. Alpha should always be 255**. Once you have looped through every pixel, **call putImageData to "paste" your pixels onto the canvas**.

Hints: Remember that pixels is **a 1-d array** containing a series of values for each (x,y) position – e.g. [$R_{(0,0)}$, $G_{(0,0)}$, $B_{(0,0)}$, $A_{(0,0)}$, $R_{(1,0)}$, $G_{(1,0)}$, $B_{(1,0)}$, $A_{(1,0)}$, $R_{(2,0)}$…]

For a (x,y) canvas position, its R value in the pixels array is at **pixels[4*(y*width + x)]**

Use the .r, .g, and .b properties of the d3.rgb() object to adjust the pixel array

If the canvas appears stripy or has lots of random colors, you are converting (x,y) position to position in the pixels array incorrectly

C. Add a slider **input** element so the user can choose a **luminosity value**. This slider should range in value from **0 to 150 with a step size of 1**. Use d3 to attach an event listener functions to the "**input**" event for the slider to call your **fillCanvas** function with the **current luminosity value of the slider**.

2. For this problem, you will visualize two provided datasets, **senate.109.rollcall.nodes.csv** and **senate.109.rollcall.edges.csv**. These datasets encode a graph of US Senators during the 109th congress. Edges have been drawn between senators who share similar voting patterns. Senators who almost always disagree will not be connected.
For this homework you are only required to show the 109th congress. In the "extras" folder I have included data for the 96th Congress if you want to try to make your own comparison. If you choose to do so, please do not include this code in your final homework submission. No extra credit will be offered.

A. Following your <p> element, create an **SVG element 800px in width and 400px in height**. Within a <script> tag, use d3 to create a **<g> element** within your SVG to contain your network diagram. Using either promises or await, **load both datasets** into memory. Finally, use **d3.scaleOrdinal** to build a color scale to show party affiliation. Set the domain and range of the scale manually so that **"Dem" maps to a blue color, "Rep" to red, and "Ind" to yellow**.

B. Construct a **d3.forceSimulation** model for your network diagram. You can use the data from **senate.109.rollcall.nodes.csv as nodes** in the model. Your model should include the following forces:

- A **linking force for edges** in the network. Use data from **senate.109.rollcall.edges.csv** to build your links. Source and target correspond to the **"icpsr"** property of nodes in this dataset, so **be sure to set .id() properly for this force.**

- A **many body repulsive force** between all nodes. Increase the strength of this force from its default of -30 to **a value of -60**.

- A **y-positioning force** that pulls all nodes towards the **middle (i.e. height / 2).** Set its strength to **0.1** so that it doesn't crush everything into a line.

- An **x-positioning force** that pulls nodes to **different x locations** based on their "party" property. This will help show divisions between political parties. Nodes where **"party" is "Dem" should be pulled towards width*0.25**; nodes where **"party" is "Rep" should be pulled towards width*0.75**; all **other nodes should be pulled towards width*0.5**. Set its strength to **0.1** so that it doesn't crush everything into a point.

C. Make a function, **render(),** that uses a data join to draw edges and a data join to draw nodes. **Draw the edges first** so that they do not appear to be on top of the nodes. You can choose the appearance of your edges, but **make sure that opacity=1** for performance reasons. **Draw circles for each node** and **set their color using the color scale you made in A**. Be sure to use enter() and merge() properly so that you only create nodes/edges once and update all of them each time render() is called. **Finally, add an .on("tick") call to your force simulation to call your render() function.**

Example: