Variables and Operators Lab

When we're finished with this lab we'll have lines on our graph that will show the overall trend of our survey results as opposed to each individual data point. We hope that the 'excellent' and 'good' responses rise over the month and the 'poor' and 'terrible' responses drop.

Refactoring for encapsulation

You may have put drawTrends, clearCanvas, and the other functions inside of the DOMContentLoaded event handler. This isn't wrong but it isn't best practice because your DOMContentLoaded event might be tough for other developers to understand. If you've done that, let's fix it.

1. Cut all of your functions out of DOMContentLoaded and paste them at the end of your JavaScript file.
2. Run and test. You probably have some new errors that have to do with a variable like ctx not being defined. If so, this is because ctx is defined with function scope inside DOMContentLoaded's function and can't be seen outside of that.
3. Make all variables recognized. You'll do this by moving the declaration of those variables up and out until they're seen everywhere. Just put them at the top of the file with *var, let,* or *const* in front of them.
4. Keep working until you stop seeing undefined variable errors in the console.

Creating some number variables

5. Go to the top of your JavaScript file and add these variables: leftSideOfGraph, widthOfGraph, topOfGraph, heightOfGraph. Their names are pretty self-explanatory. For instance, leftSideOfGraph should be the location on the canvas where the vertical axis is. Give them initial values -- estimate what you think they should be.
6. Let's test your guess! In DOMContentLoaded, draw a test rectangle kind of like this:

```
context.fillRect(leftSideOfGraph,topOfGraph,widthOfGraph,heightOfGraph);
```

7. Adjust their values so your rectangle fits perfectly inside your axes.
8. Now leave the variables but remove the rectangle. It was only there for testing the size of your graph.
9. Add a couple more variables at the top: rightSideOfGraph and bottomOfGraph. Calculate these numbers using leftSideOfGraph, widthOfGraph, topOfGraph, and heightOfGraph.
10. Redraw your axes using these new variables instead of your hardcoded numbers.
11. Rewrite your labels using these values instead of your hardcoded numbers. You'll want to do a little math like adding to bottomOfGraph to push your x-axis label down and subtracting from leftSideOfGraph to push your y-axis label left.

Coding in this way allows you to reposition or rescale your graph (within reason) by adjusting these variables. In fact, let's test that out.

12. Change the values slightly and re-load your page. You should see that your graph still draws but it will be a different size or position based on the change you made to those values. Do this several times to prove to yourself that these new variables are being used properly.

Creating some string variables

Similarly we can adjust colors globally.

13. Add some new variables at the top that represent colors; backgroundColor, labelsColor, backgroundLinesColor, axesColor and of course set them equal to some colors you like.
14. Use those variables in place of any colors you have hardcoded previously.
15. Now let's make an object:

```
var color = {
  excellent: "green",
  good: "blue",
  okay: "purple",
  poor: "orange",
  terrible: "red"
};
```
16. Last set of variables; we will use these in just a few minutes to draw the trend lines.

```
var xTic=(rightSideOfGraph-leftSideOfGraph)/(29); //pixels per day
var yTic = (bottomOfGraph - topOfGraph) / (100); //pixels per score
```

We'll use these to draw the trend lines.

Drawing the overall trend

Let's draw some actual lines on the graph. We'll start out simple; we'll just draw a line from the first data point to the last.

17. Edit drawTrends(). Make it look kind of like this:

```
function drawTrends() {
  document.getElementById("graphType").innerText = "Trends";
  clearCanvas();
  //Excellent results
  //TODO: Start the Path
  //TODO: Set the stroke style to the excellentColor
  ctx.moveTo(getXPointForDay(0),
             getYPointForScore(raw_survey_results[0].excellent));
  ctx.lineTo(getXPointForDay(29),
             getYPointForScore(raw_survey_results[29].excellent));
  ctx.stroke();
  //TODO: Close the Path
}
```

18. Lucky you! See all those "TODO" comments? Put in JavaScript code to make that work. (Note: this will not work until you write the functions in the next steps).
19. Now create the getXPointForDay() and getYPointForScore() functions. Something like this will do the trick:

```
function getXPointForDay(day=0) {
  return leftSideOfGraph + (day * xTic);
}
function getYPointForScore(score=0) {
  return bottomOfGraph - (score * yTic);
}
```

20. Run and test. You should see one line in your excellentColor.
21. Modify drawTrends to handle all five of the levels.
22. Run and test. You should expect to see five different colored lines on your graph.

Once you see them, you can be finished!

Bonus! Eliminating duplicate code

Notice that the code in drawTrends() is the same lines repeated. If you have time, let's pull the duplicated code into a function and call that multiple times.

23. Identify the lines that are duplicated. Put them in a function called drawLine that receives a startDay, endDay, and a level (like "good", "okay", "poor", etc).
24. Call that function instead of drawing the lines manually.