Lab 1 Report

DEMONSTRATED TO: AMIT PAREKH   ON: 04/02/2022

CAMERON DOUGLAS – H00295794

Lab Findings

In this section I will briefly outline my findings from each lab exercise, along with relevant discussions on how I achieved these. Discussion about the extra exercises I added will be done at the end of this section.

Exercise 1:

This exercise introduced d3 and how to set up the scripts which will be used throughout the rest of this lab. The result was printing the current d3 version (7.3.0 at the time of writing) to the screen

Exercise 2:

Using the DOM (Document Object Model) I was able to select paragraphs by the <p> tag and use d3 to modify the style attribute.

Exercise 3:

Using a for loop, I was able to append <div> elements to the body of the page and then change their text and colour using d3

Exercise 4:

Once again using the DOM and d3, I was able to select the first <div> element and change its colour and contents.

Exercise 5:

Instead of creating multiple variables and then appending one to the other, I was able to use chain syntax to more easily add and update DOM elements to produce the same results as when using multiple variables, in a more succinct and readable manner.

Exercise 6:

First, I modified the otherdata array, adding an element 'color' to each value. Then using an anonymous function, I weas able to print the colour and then change the colour of the <div> elements to the colour specified by the data.

Exercise 7:

By adding an if statement within the anonymous function, I was able to stipulate when certain elements were coloured red and when they were coloured yellow.

Exercise 8:

After updating the array, I was able to use an if statement to check the typeof the data values within the anonymous function, this allowed me to change the colour of each element depending on whether it is a character or a number. To display the data, I use the enter() method which will append the requisite number of an element based on the length of the array passed in the data() attribute. As there are no <span> elements selectAll("span") will run enter() and append span elements and set their contents to the content of the data array, until there are no elements left in the data array.

Exercise 9:

To import the data from a CSV, I use the d3.csv function. This function returns a promise so I can use the .then() function to ensure that execution occurs in the correct order. Calling d3.csv without using the .then() function will add the job to the async stack and then carry out the rest of the code as Javascript is only run on one thread. Because d3.csv is a large function and the thread cannot be put into a waiting state, JS will try and execute other code before it however this is not what we want.

--Note-- Any time from hereon in, if data is passed via CSV, I will be using the .then() function as described here so will not discuss it again.

Within the .then() function, the rest of the code is fairly trivial, for each instance in the dataset, I check whether the name contains "Mr." or "Mrs." and whether the sex is "male" or "female". I then add the results to a <div> to display them. These values were Mr: 240, Mrs: 72, Male: 266, Female: 152. The discrepancy between the sexes and the titles will be caused by some individuals having different titles for the same gender, such as Master or Miss.

Exercise 10:

Passes in the data from CSV as before, then, using the .enter() function, add <p> elements to the body containing the counts of patients from 1-30, 31-40, 41-60 and 61-100. These values were 0, 7, 155 and 135.

Exercise 11:

I appended four lines to the SVG element, arranging them using the x and y attributes to form a square. The giving them each a different stroke attribute so that each side of the square is a different colour.

Exercise 12:

Importing the data from a CSV again, I was able to check the .shape value of each element in the CSV and then use a switch case statement to appropriately add each element to the SVG.

Exercise 13:

I create the scene in a much similar way to before, only this time I use the enter() function, and a placeholder array to keep track of the number of elements currently in the scene (this will be useful when it comes to exiting and will ensure I only ever enter one of each shape).

I then create two buttons, 'enter' and 'exit', these buttons call my updateEnter() and updateExit() functions respectively. updateEnter() enters a new ellipse object, which the updateExit() function can then remove.

Exercise 14:

In this exercise, I obtain the data and put it in an array as described in exercise 10, I then pass this data to the bar chart generator provided.

Exercise 15:

When adding the rect objects to the SVG, I use an if statement to check the value of the bar, if it is > 150, the bar is red, between 100 and 15 it is orange, and less than 100 it is green.

Exercise 16:

I modify the existing code to enter() rect objects instead of circles from the same dataset as the circles.

Exercise 17:

Modified the data to include an if statement to check the value and change the colour based on the values.

Exercise 18:

Import a CSV as described before and then use this data to draw the bar chart as above.

Exercise 19:

Put the code from exercise 18 into a function with a parameter *link* which allows me to pass in different CSV files to produce two different graphs on the same page.

Exercise 20:

Added two more axes and used .attr(“transform”, “translate(x,y)”) to move them into the correct locations to form a square set of axes, I then changed the colour of the left and top axes by giving them an class attribute so that I can change those specific elements in a CSS style sheet.

Exercise 21:

I created the bar chart as done before in exercise 18, I then create two new scale variables to scale the axes from 0 – max(data) and from 0 – data.length for the x and y axis respectively, this means that the x axis will perfectly scale from zero to the same length as the bar chart bars, and the y axis will have a tick on the centre of each bar.

Exercise 22:

I put all of the code in a function called plotGraph with a parameter of data, so that I am able to call this function and draw multiple sets of data on the same graph.

Exercise 23:

I created a CSV to load the data in from, doing so as described before, and then pass this data to the plotGraph function defined in ex22 to draw the graph of the data.

Exercise 24:

I modify the plotGraph function slightly to create the SVG outside of the function so that when I call the function, I can append the path to the same SVG to have the data display in the same graph, rather than in two separate graphs. I also introduce a new parameter *color* so that I can define what colour I want each line to be.

Exercise 25:

I modify the plotGraph function further to select all “dot” elements, and append a circle for every data point, using the data and anonymous functions to return the x and y values for each one.

Exercise 26:

I define a variable triangleSymb which uses d3.symbol to create a triangle object (as SVG does not have a built in triangle like it does with circles). I then append these symbols to the line and use the transform function to give them the correct x and y coordinates. I also added a new parameter to the plotGraph function to determine which marker each line should use.

Exercise 27:

Using a counter variable, I am able to add text to the dot elements every 20 dots. I then set the text value to be the (x,y) coordinates of the point at which the text is being added.

Exercise 28:

I create two linear colour scales, one which scales with x values, and one which scales with y values. I then added a parameter to the barChart function to determine which colouring scheme to use for each graph. This results in one graph having colours dependant on the x axis (these go from blue to red, small values are blue and large values are red), the other graph has colours dependant on the y axis (these go from green to yellow, with green as the top bar and yellow as the bottom bar).

Exercise 29:

Similarly to the previous exercise, I create two colour scales, scaling with the two axes. I then apply them using an anonymous function to each marker (getting a colour from the scale dependant on its x or y value). The sine wave scales along the y axis, with blue at the top and green at the bottom, the cos wave scales along the x axis, with red on the left and yellow on the right.

Exercise 30 and 31:

I first added new values to the array and sort them from smallest to largest, I then define a colour gradient as before to colour the segments based on their size. After generating the arcs using the code given, I select them all and append a text element to each arc and set the text to be the value from the data array. Then using the arc.centroid() function, I can set the x and y coordinates of each text element to be the centre of it's respective arc.

Exercise 32:

I import the SVG image from a link and append it to the SVG using the SVG attribute xlink:href, I then set the width and height of the image to be the width and height of the SVG and then send it to the back using the .lower() function. I then draw the pie chart on top of the SVG image so the image appears as its background.

When developing this exercise, I had problems with the image appearing over the top of the pie chart, so I thought it would be interesting to make a small addition, where when you hover over the SVG image, it calls the .raise() function, so the background image comes to the foreground, and returns to the background when the mouse is moved away.

Extra Exercises:

Exercise 7b:

Extending from exercise 7, I added two sliders to the page which would control the colour of the elements, the first slider changes the colour to blue from the smallest value down, and the second changes the colour to blue from the largest value up (so when using the top slider, the display shows blue numbers then red numbers and the bottom slider shows red numbers then blue numbers).

By using HTML and the JS DOM, I am able to create and select the sliders and get the value from them, I can then pass this value to my d3 functions where I can change the colours based on the values obtained from each slider.

Exercise 16b:

I added an enter and exit button, which allows me to add and remove the circles from exercise 16 in order. I do this by using a placeholder array which keeps track of the current number of elements on screen. It starts as an empty array, when the enter button is pressed, it adds one element to the array which allows the first circle to be entered into the SVG using the enter() function. When the exit button is pressed, an item will be removed from the array, meaning there is one more element on screen than in the array, allowing that element (which will be the most recently added element) the be exited from the SVG using the exit() function.

Exercise 21b:

In this extension, I add a text input field and a button to allow the dynamic creation of the bar chart. Numbers are input to the text field one at a time (pressing the submit button each time), and when the submit button is pressed, it draws the bar chart with the current elements in the array.

This is done by calling the whenPressed function when the button is pressed. This function gets the value from the input source and pushes it to the dataset. It then removes the current data set displayer (a textual display of the values in the array) and then enters a new one with the updated values. It then calls the updateChart function.

The updateChart function simply exits the old chart from the SVG and calls drawChart to draw a new chart with the new dataset. This chart is then drawn in the same way as in exercise 21.

Exercise 21c:

Using the same chart generation as in ex21b, only this time there are sliders to control the bars which are green and those which are red (in the previous example, I was using the fixed colouring from exercise 17). Any bar which has a value less than the value indicated by the green slider will be green, and any bar with a value higher than the value indicated by the red slider will be red.

I do this by updating a variable based on the values from the sliders, and then call updateChart from the oninput() functions of each slider (these are called whenever the sliders are clicked). The updateChart function now checks the variables when selecting a colour (rather than the fixed values used before).

Exercise 22b:

Using a slider, I can control the multiplier of the sine function (originally set to 6.2 in the code given to us) by modifying this number I can change the frequency of the peaks and display them. This is done by obtaining a value from the slider in the oninput() function, then recalculating the dataset using this new value, and then calling the updatePath() function. The updatePath() function, removes the old SVG and then calls the plotGraph() function with the new dataset to draw a new one in its place.

Exercise 22c:

This time I have two sine waves which can be manipulated by sliders in the same manner as exercise 22b. Once these values have been updated, a new dataset is created. Each datapoint in this new dataset is the addition of the (x,y) coordinates from the other two sine waves. This gives a new graph which shows how the two waves would interfere with each other.