# NM2207

Session 11 Codealong

# Overview of what we will do today:

* Updating charts with events
* Updating bar chart axes with events
* Introduction to d3
* Introduction to code exploration with ChatGPT

Please upload the codealong (only Part 1, 2) to your nm2207 page by the end of Saturday of Week 11.

**Revision**

**Buttons and click events**

Remember buttons and events?

* We created buttons on our html page using <input type=”button” id = “id of button”>Click here</input>
* We added events to it in our javaScript file using document.getElementById(“id of button”).addEventListener().

Today, we are going to use it to update our charts.

**forEach**

The forEach can come in handy again, when we need to update multiple charts after an event.

The forEach method provides a cleaner and more concise syntax to loop through arrays. It takes a callback function as an argument, which is executed for each element in the array. The callback function can also receive the current element's index (i in this case) as a parameter.

Now we can use it manipulate the original dataset(s) we created for a chart.

origData.forEach((data) => {}) and then in the curly brackets, we add commands to do something to each dataset of the chart, such as update it or add new data to it.

The following example is taken from 01.changeData.

* What is the datatype of origData?



We see that origData is an array of arrays. It has 3 arrays within it.

Each of them (ie each of the individual arrays) can be accessed as origData[0], origData[1], and origData[2].

* What do you think is happening in the following code?

Text

Description automatically generated

* What is each “dataset”?

Ans: These commands are pushing new objects to an object array called datasets, of the form that a chart usually takes as input. The properties of the object are its label, an array called dataset, a backgroundColor and a borderColor. The array called dataset is one of the arrays from origData.

**New concepts**

**The Map function**

The map function acts on every element of an array, similar to forEach.

* What do you think is happening in the code below?
* Look up what the “map” function does or try it out, here: https://www.w3schools.com/jsref/jsref\_map.asp

Graphical user interface, text

Description automatically generated

This code uses the forEach() method to loop through each element of an array called origData.

For each element of origData, the code creates a new array called squaredData by applying the function number => number \* number to each element of the original dataset. This function squares each number in the original dataset.

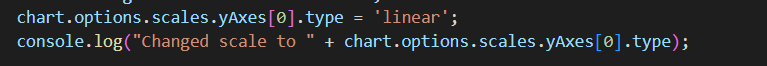
The let keyword is used to declare a new variable squaredData inside the loop that will store the squared dataset.

Finally, the console.log() method is used to print the squaredData array to the console for each dataset.

In summary, this code iterates through an array of datasets and squares each element in each dataset, printing the resulting squared dataset to the console.

**Bar chart axes**

We are also going to be modifying the yAxes of our charts this week, using code that looks something like this:



The first line sets the type of the first y-axis (index 0) in the chart to 'linear'. In Chart.js 2.7, the y-axis can have different types, such as 'linear', 'logarithmic', 'time', etc. By setting the type to 'linear', you're configuring the y-axis to display linearly-scaled values.

The second line logs a message to the browser console, indicating that the y-axis type has been changed. It concatenates the string "Changed scale to " with the current type of the y-axis (in this case, 'linear'), and logs the resulting message. For example, the message could be: "Changed scale to linear". This line is useful for debugging purposes, as it helps you track the changes in the y-axis type when the chart is updated.

**Datasets**

Here’s how you access the array of datasets being used in a chart:



**Part 1**

**Add a button and an event**

Lets start with the codealong, 01.changeData. Suppose we want to understand how squaring data changes its shape. How would we do that?

For starters, we would need to trigger an event when we want to change the data. When we change the data, we would also want to update our charts each time the button is clicked.

Let’s explore the code for 01.changeData.

First, in the index.html, add a button called “toggle” with the text “Square/Unsquare” as a warm-up.

Next, in the main.js, each time it is clicked, trigger a function called updateData, which prints a message in the console.

**Part 2**

Connect the button to the correct data update function.

We want the button to call the square function when the chart type is linear. And the orginalValue function when the chart type is logarithmic.

Remember to pass the chart as an argument in your function call.

Now, use the example code in Main concepts to fill up the originalValue and the square function, so that in the latter case, the square of the values of origData become the new dataset of the chart.

**Part 3**

Now, we want to update our chart with the new values we have calculated. Use the chart.config.data.datasets property to set it to the new “datasets” objects that are created in either function.

We also want to change the scale of the y axes so that it can correctly plot our data. Refer to “main concepts” earlier to see how that’s done.

Remember to chart.update() at the end of the function.

**Reflection:**

How do data transformations change the way we represent data?

How could squaring the data be used to (a) amplify small differences, and (b) shrink distances in our dataset?

**Part 4**

Now, lets explore the 02.mapEvent folder. This uses d3 to populate and update the map.

Lets explore the functions and calls we observe:

* **Line 2, d3.csv** : This seems to be the command that replaces our “fetch” command to create a promise. Instead of following up with “then”,the second argument to the function is what should be executed once the data is collected.
* You’ll notice that this function(err rows) is anonymous (it has no name) and it takes two arguments, err and rows. Where does its closing bracket end? That’s right, at the very end of the code.
* Trace the progress of the code. When is the filter\_and\_unpack function called?
* Add console.log statements to inspect what the value is of frames[i] after it is written, in line 16.
* Examine slider\_steps. It is the property of the slider which is updated in line 17. When is it finally used to initialize the “layout” property of the page?
* **Line 117, Plotly.newPlot:** This appears to have replaced the way we used to add a new chart to our webpage, by referencing a canvas element in the html. Notice what arguments are passed in this function call.Examine each of them. Here we see a “promise” function being used, so that the complete data is added to the charts only AFTER everything has been correctly rendered as a plot.

**Here's a detailed description of the code in main.js that I generated using Chat-GPT, and this is a demonstration of how Chat-GPT can be useful for your programming tasks.**

This code is an example of a Plotly visualization in JavaScript. It creates an interactive choropleth map of world life expectancy from 1952 to 2007, using data from Gapminder. The map is animated to show changes in life expectancy over time, and it includes a slider and play/pause buttons to control the animation.

Here's a step-by-step explanation of the code:

1. The **d3.csv** function reads the CSV data from the provided URL. The callback function handles the data manipulation and visualization.
2. The **filter\_and\_unpack** function filters rows of the data based on the specified year and then extracts the values of a specified key (column) in the filtered data.
3. The **frames** array stores the data for each year as separate frames for the animation. The **slider\_steps** array stores the configuration for the slider steps.
4. The loop iterates through the years, from 1952 to 2007, in increments of 5. For each year, it calls the **filter\_and\_unpack** function to get the life expectancy and location data for that year. These values are then used to create a new frame and a slider step.
5. The **data** array defines the initial choropleth map with the first frame's data, setting the location mode to 'world', the color scale (zmin and zmax), and other map properties.
6. The **layout** object specifies the overall layout of the plot, including the map's appearance (colors, borders, etc.), the title, and the configurations for the animation controls (play/pause buttons and slider).
7. The **Plotly.newPlot** function creates the plot in the specified HTML div ('myDiv') using the data and layout objects. After the plot is created, the **Plotly.addFrames** function adds the animation frames to the plot.

When the visualization is rendered in a browser, the user can interact with the map, play and pause the animation, and use the slider to jump to specific years.

*P.S.: It makes me feel a little bit inadequate, but I remind myself that it’s to help me do better as a teacher and a programmer, and it’s not fair to compare myself against something which has the whole world’s knowledge within it to look up at a glance!*

*P.P.S.: Personally, I find Chat-GPT very useful to understand and debug code. Nevertheless, it doesn’t let me be over-reliant (and neither should you be!) For my programming tasks (typically in R), I find that it’s wrong about 20% of the time. Also, when it messes up code, at least 10% of the time I have to give up and fix the code myself because it complicates rather than simplifies the code 😉*

**Reflection:**

How do data transformations change the way we represent data?

How could squaring the data be used to (a) amplify small differences, and (b) shrink distances in our dataset?