School of Software and Electrical Engineering

COS30045 Data Visualisation

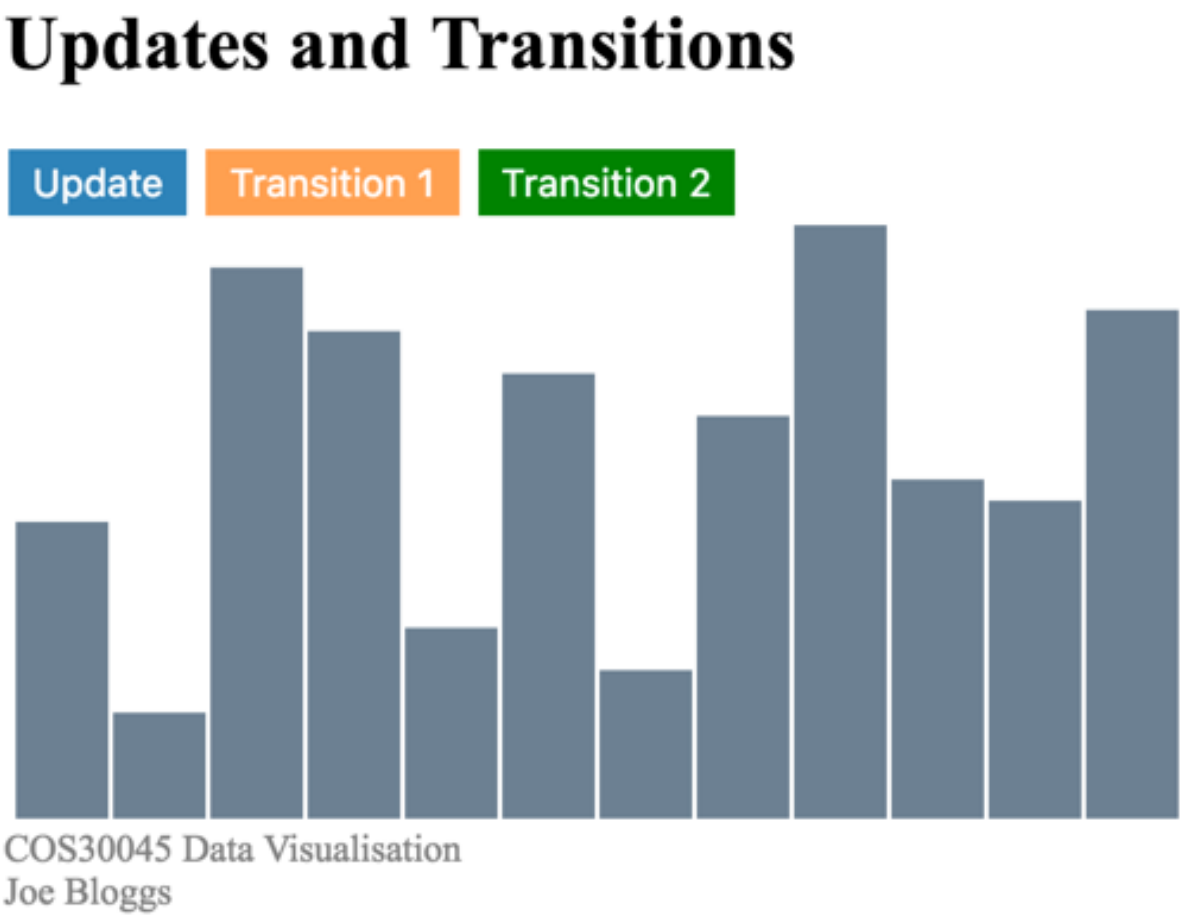
Exercise 5.2 D3 Transitions

ILO	Create web-based interactive visualisations using real-world data sets.
Aim:	Use D3 to generate smooth transitions between data updates.
Resources:	*Textbook:* Chapter 9 Updates, Transitions and Motion - Murray (2017) Interactive Data Visualisation (2nd Ed) on ProQuest
Demonstration	If you are required to demonstrate this exercise we will be looking for: - code that is appropriate for exercise, well formatted and commented - code that runs correctly and meets the requirements specified in this exercise - an explain programming features and concepts in the code - the ability to successfully edit code to change a specified feature of the program

Overview

At the end of the Updating the Data exercise we had a chart that updated with random values up to 25. However, the transition between updates was very sharp. Basically the old data set was removed and the new data put on top. In this exercise we will be working on making the transition between data sets smoother and more engaging by experimenting with a variety of transition animations. It should look a bit like this where the buttons trigger different kinds of transitions when the data is updated:

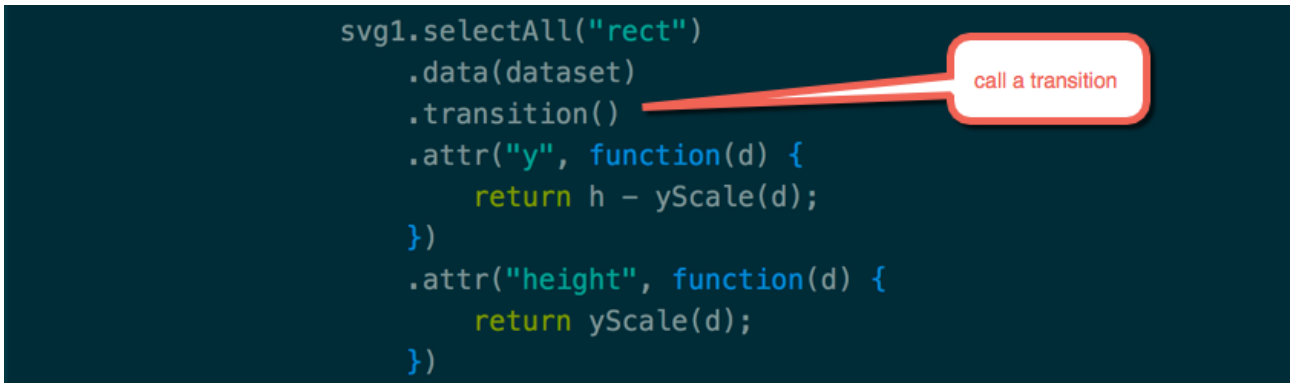
Step 1: Start with the code from your Updating Data Exercise



Requirements

- ☐ code produces a bar chart which allows the user to choose between at least two different transitions when updating data
- ☐ the duration of transition includes is the same no matter the number of data points and is delayed so the user is able to keep track of the transition

The first step in generating a smooth transition between data sets is as easy as one line of code:



Add `transition()` to your code from your Updating Data exercise. Save and watch what happens when you update. There is now a smoother transition between the data sets. D3 interpolates the difference between the start and end states and animates the values to change over time. The default time between the start state and end state is 250 ms.

Step 2: Change the duration of the transition

To control how long it takes for the transition to take place you can use `duration ()`. Extend the length of the transition to 2000 ms.

```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .duration(2000)
    .attr("y", function(d) {
        return h - yScale(d);
    })
    .attr("height", function(d) {
        return yScale(d);
    })
```

Now you should see the transition in 'slow motion'. Change the duration to 5000 ms and you will be able to see how the animation changes speed during the transition. It starts slow, then speeds up and then finishes slow. This is called easing.

Step 3: Experiment with different types of easing

There are a number of different easing functions in D3. The default is `d3.easeCubicInOut`. Experiment with a number of different easing functions. For example;

```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .delay(function(d, i) {
        return i * 100;
    })
    .duration(500)
    .transition()
    .duration(2000)
    .ease(d3.easeElasticOut)
    .attr("y", function(d) {
        return h - yScale(d);
    })
    .attr("height", function(d) {
        return yScale(d);
    })
```

`d3.EaseCircleIn`

`d3.easeCircleOut`

`d3.easeElasticOut`

See d3-ease for a full list of ease transitions.

Make a demo page that allows users to try out at least two different transitions.

```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .delay(1000)
    .duration(2000)
```

Step 3: Using delays

You can also delay when a transition occurs. We can add a static delay or we could use delays dynamically to stagger the application of a transition. This can make it easier for people to follow the transition.

This delay function takes the index of the data value and multiplies it by 100 ms so each element is delayed 100s more than the the previous one. Add a delay that gives a nice staggered animation where the transition rolls a cross the bars.

When the date updates, the number of data elements increases, then so does the length of the transition. Scale the length of a transition by dividing `i` by the time you want the transition to take.

```
svg1.selectAll("rect")
    .data(dataset)
    .transition()
    .delay(function(d, i) {
        return i/dataset.length * 1000;
    })
```

Now the animation will always take 1000 ms no matter how many data points are in the data set… a feature which will come in handing in our next task where we will look at adding and removing data.

Add some buttons to trigger different animations.