F21DV Lab 2 Report

Date Demonstrated: 25/02/2022

Demonstrated To: Shuangjiang Xue

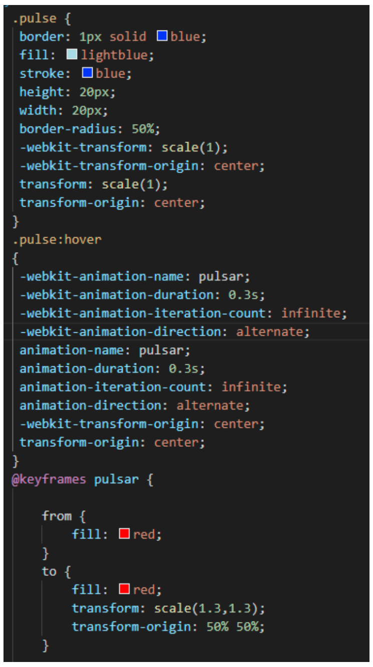
All exercises fully completed as per the specification.

Part 1. CSS Effects/Animations

Exercise 1.

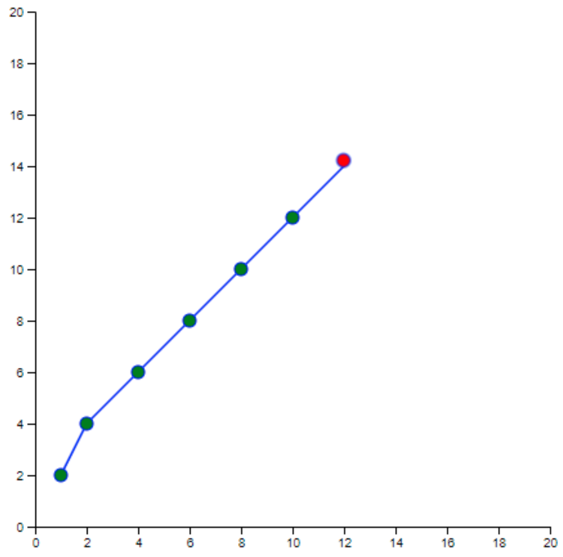
I struggled to get the dot to pulse properly while on the path of the line graph despite the fact the CSS correctly makes the circle I created outside the SVG pulse. The dots on the line graph seem to move off the axis of the line while increasing/decreasing in size and the colour fill changes to red.

CSS:



```css
.pulse {
 border: 1px solid blue;
 fill: lightblue;
 stroke: blue;
 height: 20px;
 width: 20px;
 border-radius: 50%;
 -webkit-transform: scale(1);
 -webkit-transform-origin: center;
 transform: scale(1);
 transform-origin: center;
}
.pulse:hover
{
 -webkit-animation-name: pulsar;
 -webkit-animation-duration: 0.3s;
 -webkit-animation-iteration-count: infinite;
 -webkit-animation-direction: alternate;
 animation-name: pulsar;
 animation-duration: 0.3s;
 animation-iteration-count: infinite;
 animation-direction: alternate;
 -webkit-transform-origin: center;
 transform-origin: center;
}
@keyframes pulsar {

    from {
        fill: red;
    }
    to {
        fill: red;
        transform: scale(1.3,1.3);
        transform-origin: 50% 50%;
    }
```

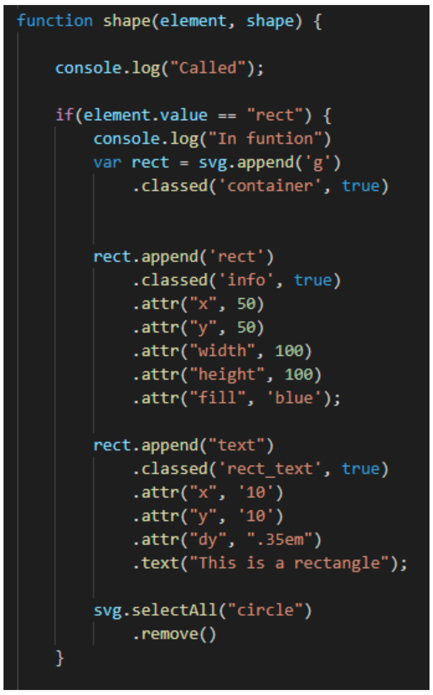
Result:



Exercise 2.

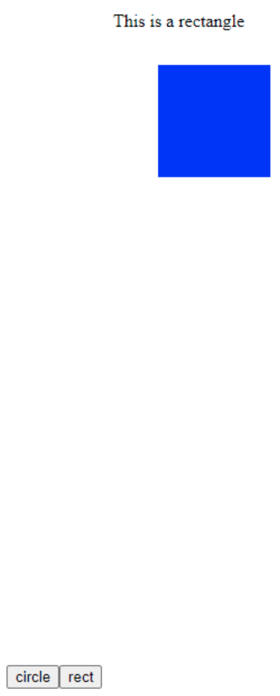
For this exercise I created an array of shapes that can be added to the SVG when the user clicks the corresponding button. Initially I bind button objects to the array of shape names so a button is added to the body of the window for each shape in the shape array that have the value of the shape name. I then add the shape() function as a function to execute when the each button is clicked using attr('onclick', 'shape(this)'). When each button is clicked, the value is checked and a shape corresponding to that value is added to the SVG. A class is also added to the shape so that text will appear above it when a user hovers their mouse over it. When another button is pressed, the previous shape is removed from the screen.

Onclick function:



```
function shape(element, shape) {

    console.log("Called");

    if(element.value == "rect") {
        console.log("In funtion")
        var rect = svg.append('g')
            .classed('container', true)


        rect.append('rect')
            .classed('info', true)
            .attr("x", 50)
            .attr("y", 50)
            .attr("width", 100)
            .attr("height", 100)
            .attr("fill", 'blue');

        rect.append("text")
            .classed('rect_text', true)
            .attr("x", '10')
            .attr("y", '10')
            .attr("dy", ".35em")
            .text("This is a rectangle");

        svg.selectAll("circle")
            .remove()
    }
```

Result:

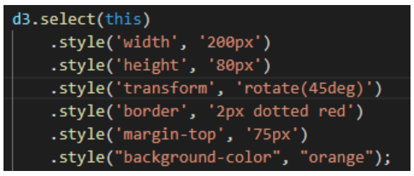


Part 2. Events

Exercise 3

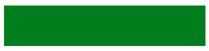
In addition to the colour change of the shape, I increased the size, added a dotted red border, rotated the shape 45 degrees and added a top margin of 75px so that the full shape showed on screen after being rotated.

onmouseover properties:

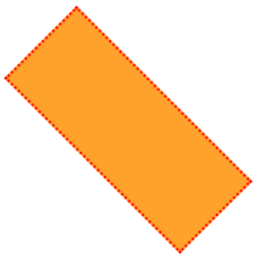


```
d3.select(this)
    .style('width', '200px')
    .style('height', '80px')
    .style('transform', 'rotate(45deg)')
    .style('border', '2px dotted red')
    .style('margin-top', '75px')
    .style("background-color", "orange");
```

Before:



After:

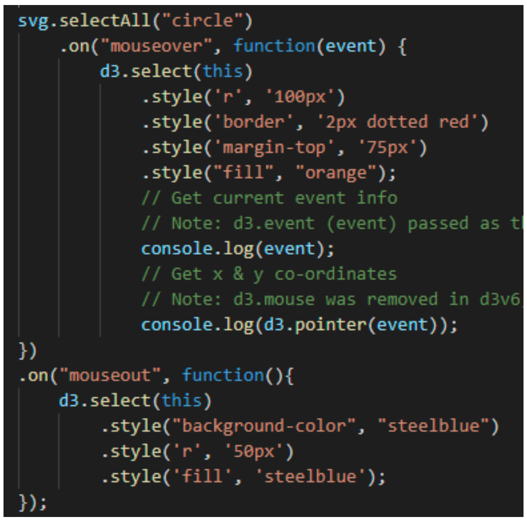


Exercise 4.

I successfully replaced the div with an SVG appended to the body. I then appended a circle to the SVG and added the .on(‘mouseover’,…) property. When the mouse hovers over the circle the radius doubles in size and the fill changes from green to orange. Once the mouse has moved away from the circle, it returns to 50px (it’s original size) and the fill changes to steelblue.

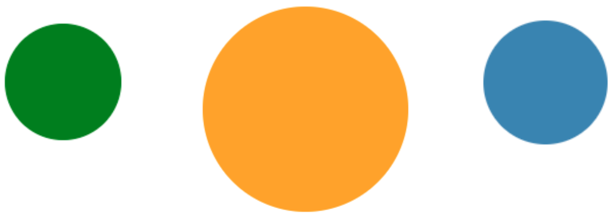
Name: Elliot Whitehouse Lab 2 F21DV

onmouseover properties:



```
svg.selectAll("circle")
    .on("mouseover", function(event) {
        d3.select(this)
            .style('r', '100px')
            .style('border', '2px dotted red')
            .style('margin-top', '75px')
            .style("fill", "orange");
            // Get current event info
            // Note: d3.event (event) passed as t
            console.log(event);
            // Get x & y co-ordinates
            // Note: d3.mouse was removed in d3v6
            console.log(d3.pointer(event));
})
.on("mouseout", function(){
    d3.select(this)
        .style("background-color", "steelblue")
        .style('r', '50px')
        .style('fill', 'steelblue');
});
```

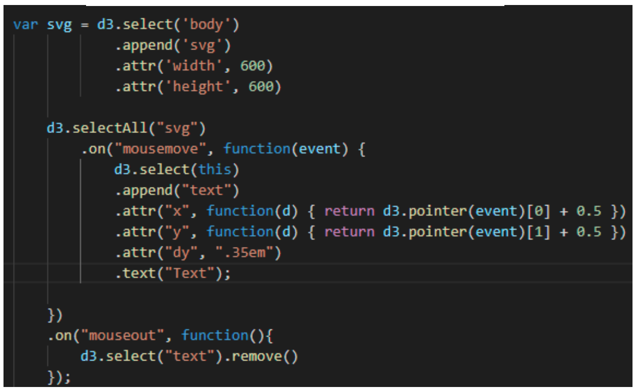
Before: During: After:



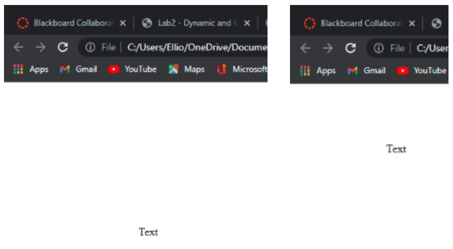
Exercise 5.

To make the text follow the mouse pointer I used the d3.pointer(event) array with the value at index 0 being the x coordinate and the value at index 1 being the y coordinate. Inside the mouseover function I set the x and y attributes of the text box to be the x and y values in the d3.pointer(event) array. In the mouseout event I removed the text from the SVG so that it didn't create a trail of text across the screen. I also offset the text box from the pointer by 0.5 so it is clearly visible.

Code:



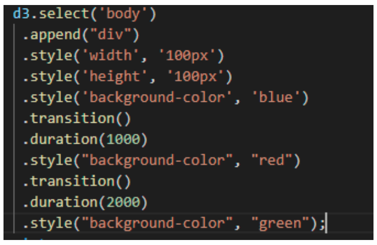
```
var svg = d3.select('body')
            .append('svg')
            .attr('width', 600)
            .attr('height', 600)

    d3.selectAll("svg")
        .on("mousemove", function(event) {
            d3.select(this)
            .append("text")
            .attr("x", function(d) { return d3.pointer(event)[0] + 0.5 })
            .attr("y", function(d) { return d3.pointer(event)[1] + 0.5 })
            .attr("dy", ".35em")
            .text("Text");

    })
    .on("mouseout", function(){
        d3.select("text").remove()
    });
```



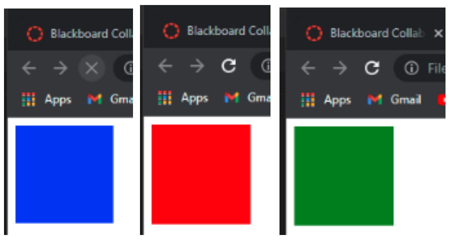
Part 3. Transitions

Exercise 6.

To complete this exercise I added another transition with a duration of 2000 milliseconds and a style change of background-color green to the chain already present.



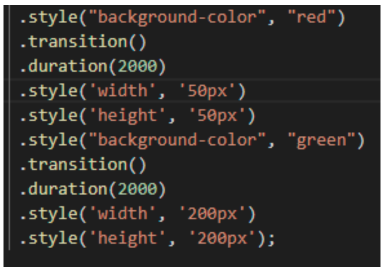
```
d3.select('body')
 .append("div")
 .style('width', '100px')
 .style('height', '100px')
 .style('background-color', 'blue')
 .transition()
 .duration(1000)
 .style("background-color", "red")
 .transition()
 .duration(2000)
 .style("background-color", "green");
```



Exercise 7.

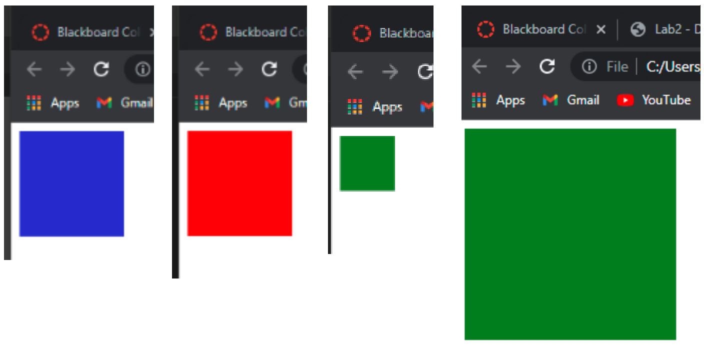
To achieve the resizing I used the same code as in exercise 7 and added in a width and height change and an additional transition to increase the size to 200px x 200px after the color change to green.

Code:



```
.style("background-color", "red")
.transition()
.duration(2000)
.style('width', '50px')
.style('height', '50px')
.style("background-color", "green")
.transition()
.duration(2000)
.style('width', '200px')
.style('height', '200px');
```

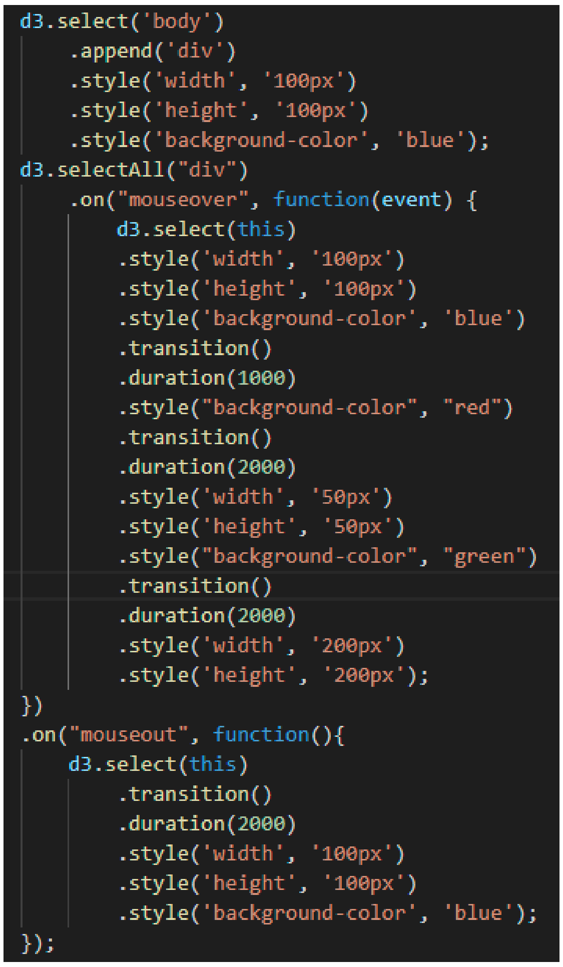
Result:



Exercise 8.

For this transition I used the mouseover event from exercise 3 but instead used the transitions and attributes from exercise 7. This achieved the desired effect of completing the transition of exercise 7 only when the mouse hovers over the square. As soon as the mouse moves away from the square the square transitions back to it's original size and colour with a transition duration of 2 seconds.

Code:

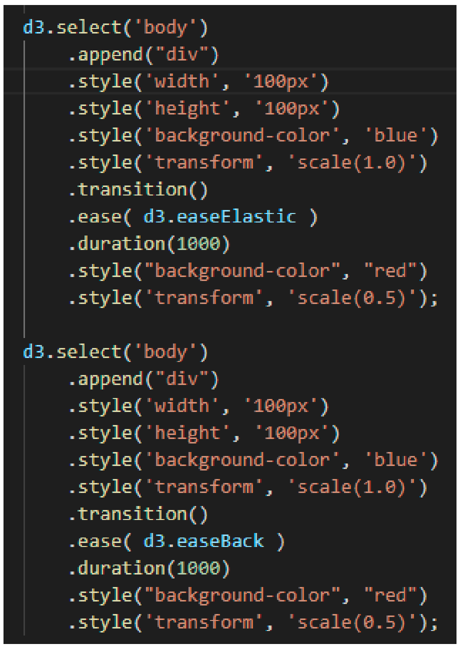


```
d3.select('body')
    .append('div')
    .style('width', '100px')
    .style('height', '100px')
    .style('background-color', 'blue');
d3.selectAll("div")
    .on("mouseover", function(event) {
        d3.select(this)
        .style('width', '100px')
        .style('height', '100px')
        .style('background-color', 'blue')
        .transition()
        .duration(1000)
        .style("background-color", "red")
        .transition()
        .duration(2000)
        .style('width', '50px')
        .style('height', '50px')
        .style("background-color", "green")
        .transition()
        .duration(2000)
        .style('width', '200px')
        .style('height', '200px');
})
.on("mouseout", function(){
    d3.select(this)
        .transition()
        .duration(2000)
        .style('width', '100px')
        .style('height', '100px')
        .style('background-color', 'blue');
});
```

Exercise 9.

For this exercise I copied the d2.select chain for the first div and changed the easing property. In the second div I used d3.easeElastic and in the third div I used d3.easeBack.

Code:

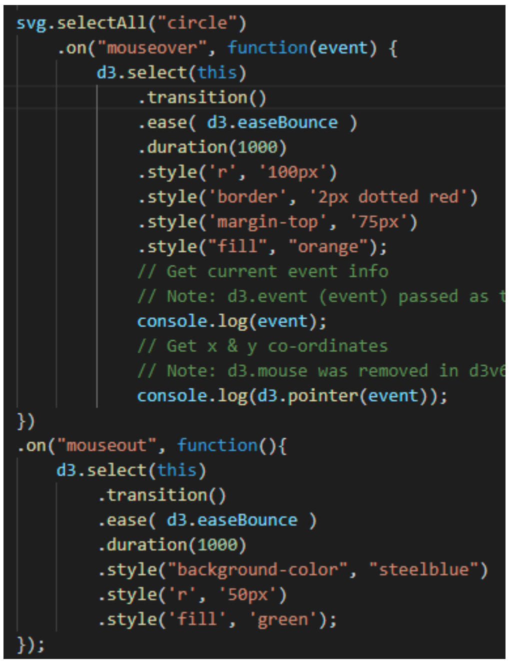


```
d3.select('body')
    .append("div")
    .style('width', '100px')
    .style('height', '100px')
    .style('background-color', 'blue')
    .style('transform', 'scale(1.0)')
    .transition()
    .ease( d3.easeElastic )
    .duration(1000)
    .style("background-color", "red")
    .style('transform', 'scale(0.5)');

d3.select('body')
    .append("div")
    .style('width', '100px')
    .style('height', '100px')
    .style('background-color', 'blue')
    .style('transform', 'scale(1.0)')
    .transition()
    .ease( d3.easeBack )
    .duration(1000)
    .style("background-color", "red")
    .style('transform', 'scale(0.5)');
```

Exercise 10.

For this exercise I modified the code from exercise 5 by adding an ease function to the mouseover function chain of functions. I passed in as the ease d3.easeBounce as specified in the lab sheet and also use the same ease for the mouseout function so the circle returns to it's original shape with a 1 second easeBounce transition.

Code:

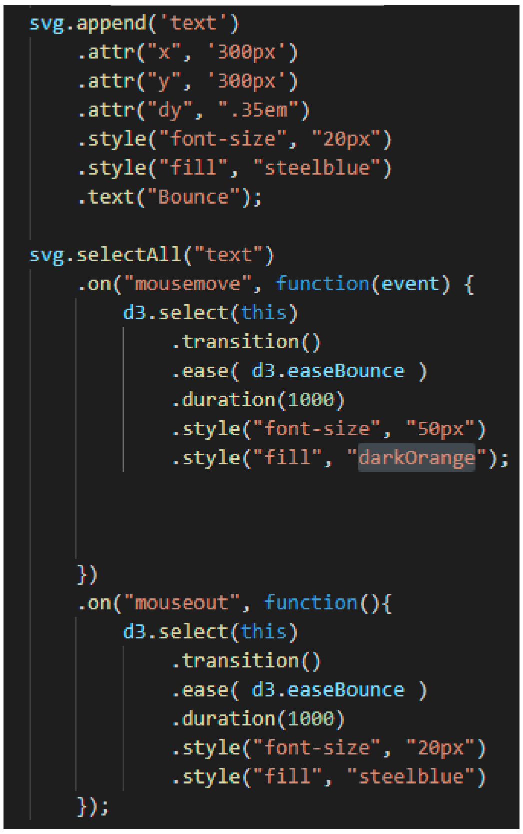


```
svg.selectAll("circle")
    .on("mouseover", function(event) {
        d3.select(this)
            .transition()
            .ease( d3.easeBounce )
            .duration(1000)
            .style('r', '100px')
            .style('border', '2px dotted red')
            .style('margin-top', '75px')
            .style("fill", "orange");
            // Get current event info
            // Note: d3.event (event) passed as t
            console.log(event);
            // Get x & y co-ordinates
            // Note: d3.mouse was removed in d3v6
            console.log(d3.pointer(event));
})
.on("mouseout", function(){
    d3.select(this)
        .transition()
        .ease( d3.easeBounce )
        .duration(1000)
        .style("background-color", "steelblue")
        .style('r', '50px')
        .style('fill', 'green');
});
```

Exercise 11.

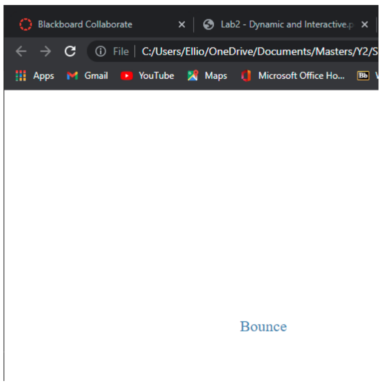
For exercise 11, I created an SVG and appended a text element with font-size 20px. During the transition in the mouseover function the font-size increases to 50px, the text colour changes to darkOrange and the transition happens with a d3.easeBounce ease. The same transition happens in reverse in the mouseout function where the text returns to font-size 20px and a steel blue colour.

Code:

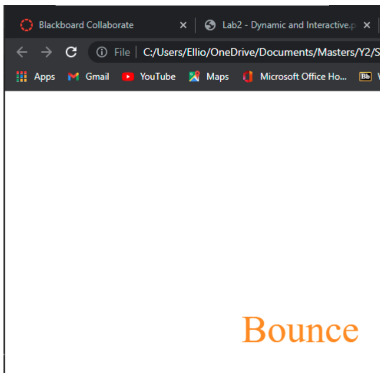


```
svg.append('text')
    .attr("x", '300px')
    .attr("y", '300px')
    .attr("dy", ".35em")
    .style("font-size", "20px")
    .style("fill", "steelblue")
    .text("Bounce");

svg.selectAll("text")
    .on("mousemove", function(event) {
        d3.select(this)
            .transition()
            .ease( d3.easeBounce )
            .duration(1000)
            .style("font-size", "50px")
            .style("fill", "darkOrange");



    })
    .on("mouseout", function(){
        d3.select(this)
            .transition()
            .ease( d3.easeBounce )
            .duration(1000)
            .style("font-size", "20px")
            .style("fill", "steelblue")
    });
```

Before/After mouseover:


Blackboard Collaborate
Lab2 - Dynamic and Interactive
File | C:/Users/Ellio/OneDrive/Documents/Masters/Y2/S
Apps
Gmail
YouTube
Maps
Microsoft Office Ho...
Bounce

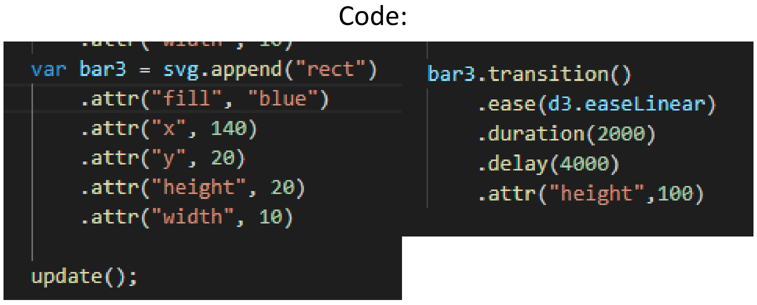
On mouseover:


Blackboard Collaborate
Lab2 - Dynamic and Interactive
File | C:/Users/Ellio/OneDrive/Documents/Masters/Y2/S
Apps
Gmail
YouTube
Maps
Microsoft Office Ho...
Bounce

Exercise 12.

To add a third bar, I copied the same code as for the first two bars in order to keep them identical. I also added the same transition and height property so it would match the first two, however this timeammending the duration function to be 4000 ms.

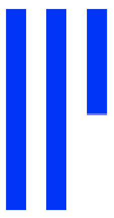
Code:



```
var bar3 = svg.append("rect")
    .attr("fill", "blue")
    .attr("x", 140)
    .attr("y", 20)
    .attr("height", 20)
    .attr("width", 10)

update();
```

```
bar3.transition()
    .ease(d3.easeLinear)
    .duration(2000)
    .delay(4000)
    .attr("height",100)
```

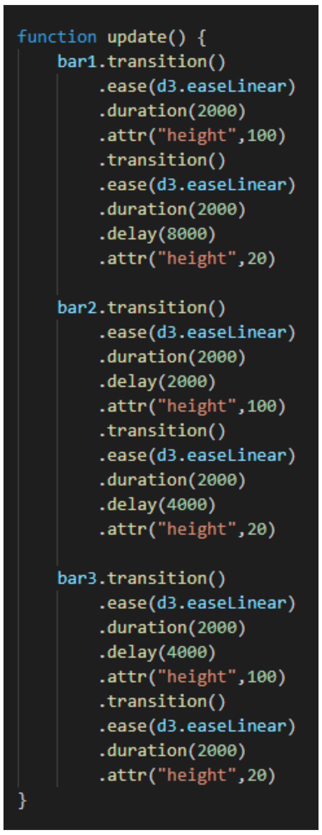
During Transition:



Exercise 13.

To reverse the transition I chained the reverse transition to each of the bar transition functions, paying particular attention to the delay time. The third bar starts reducing in height first for a duration of 2 seconds and so the second bar has a delay of 4000ms before it starts reducing to account for the 2000ms for the third bar to get bigger and 2000ms to get smaller. The final bar has a reduction transition delay of 8000ms to account for the second and third bars growing and shrinking transition times.

Code:

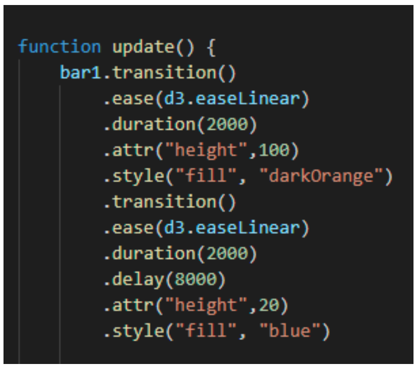


```
function update() {
    bar1.transition()
        .ease(d3.easeLinear)
        .duration(2000)
        .attr("height",100)
        .transition()
        .ease(d3.easeLinear)
        .duration(2000)
        .delay(8000)
        .attr("height",20)

    bar2.transition()
        .ease(d3.easeLinear)
        .duration(2000)
        .delay(2000)
        .attr("height",100)
        .transition()
        .ease(d3.easeLinear)
        .duration(2000)
        .delay(4000)
        .attr("height",20)

    bar3.transition()
        .ease(d3.easeLinear)
        .duration(2000)
        .delay(4000)
        .attr("height",100)
        .transition()
        .ease(d3.easeLinear)
        .duration(2000)
        .attr("height",20)
}
```

Exercise 14.

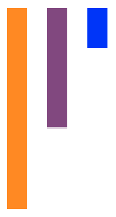
For this exercise I again chained more functions to the bar transaction functions, this time adding style functions to change the fill. The bars begin blue, transition to darkOrange as the bars grow and then back to blue as they reduce in size.

Code:



```
function update() {
    bar1.transition()
        .ease(d3.easeLinear)
        .duration(2000)
        .attr("height",100)
        .style("fill", "darkOrange")
        .transition()
        .ease(d3.easeLinear)
        .duration(2000)
        .delay(8000)
        .attr("height",20)
        .style("fill", "blue")
```

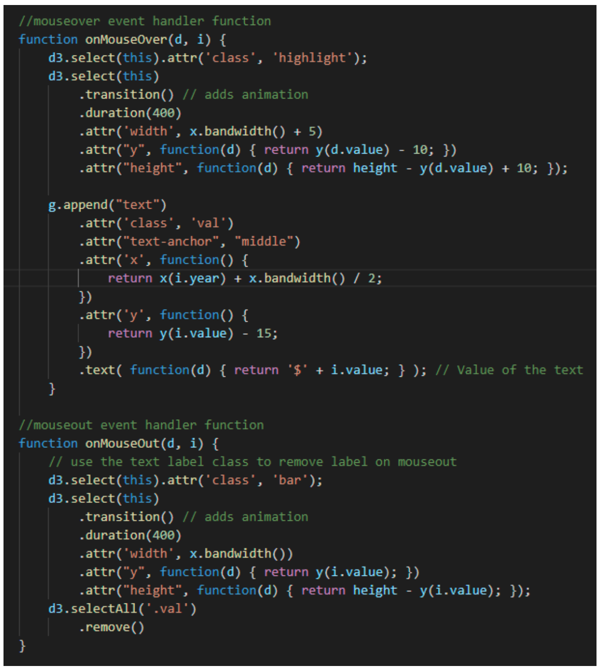
During Transition:



Part 4. Animated Chart

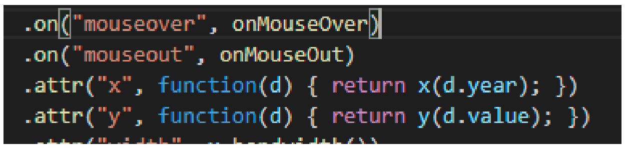
Exercise 15.

Functions:



```
//mouseover event handler function
function onMouseOver(d, i) {
    d3.select(this).attr('class', 'highlight');
    d3.select(this)
        .transition() // adds animation
        .duration(400)
        .attr('width', x.bandwidth() + 5)
        .attr("y", function(d) { return y(d.value) - 10; })
        .attr("height", function(d) { return height - y(d.value) + 10; });

    g.append("text")
        .attr('class', 'val')
        .attr("text-anchor", "middle")
        .attr('x', function() {
            return x(i.year) + x.bandwidth() / 2;
        })
        .attr('y', function() {
            return y(i.value) - 15;
        })
        .text( function(d) { return '$' + i.value; } ); // Value of the text
    }

//mouseout event handler function
function onMouseOut(d, i) {
    // use the text label class to remove label on mouseout
    d3.select(this).attr('class', 'bar');
    d3.select(this)
        .transition() // adds animation
        .duration(400)
        .attr('width', x.bandwidth())
        .attr("y", function(d) { return y(i.value); })
        .attr("height", function(d) { return height - y(i.value); });
    d3.selectAll('.val')
        .remove()
}
```

In selectAll(".bar") function chain:

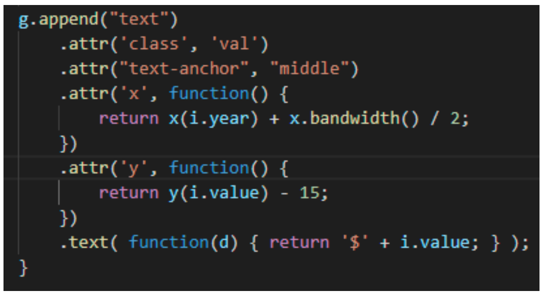


```
.on("mouseover", onMouseOver)
.on("mouseout", onMouseOut)
.attr("x", function(d) { return x(d.year); })
.attr("y", function(d) { return y(d.value); })
```

Exercise 16.

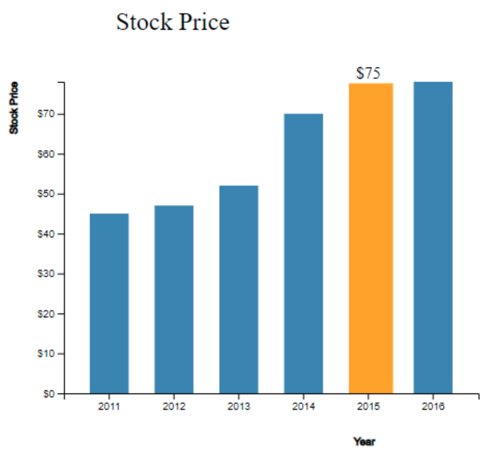
To append the text to the screen I added to the onMouseOver function. When the user moves their mouse over a bar, in addition to the bar getting slightly larger and turning orange, a text box is added to the "g" that contains the bar with the dollar value of the bar. To align it above the bar itself the x position is taken using the x() function to align it with the corresponding year on the x axis and to it is added the width of the bar divided by 2 which aligns the text in the centre of the bar. On the y axis, 15px are taken away from the value after passing the value through the y() function so it sits above the bar and its readable.

Code:



```
g.append("text")
    .attr('class', 'val')
    .attr("text-anchor", "middle")
    .attr('x', function() {
        return x(i.year) + x.bandwidth() / 2;
    })
    .attr('y', function() {
        return y(i.value) - 15;
    })
    .text( function(d) { return '$' + i.value; } );
}
```

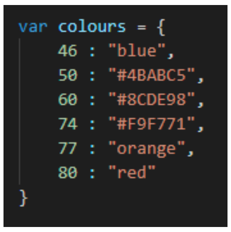
Result:



Exercise 17.

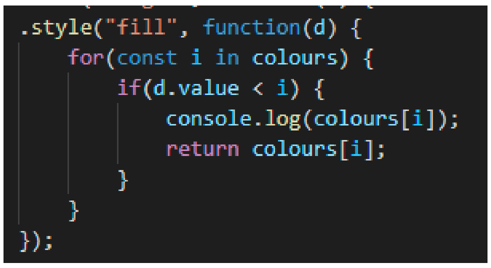
For this exercise I created an object with 6 key:value pairs. Each key is a number and the value is a colour. The keys represents thresholds for the data to separate each bar out into its own colour, For example, the first colour threshold is 46 and when searching through the key:value pairs, any bar with a value less than 46 will be given the associated colour. This splits all the bars out into a different colour. These colours show when the mouse is hovered over the bar but for the purposes of demonstration they are all being shown in the screenshot below.

Declaring colours:



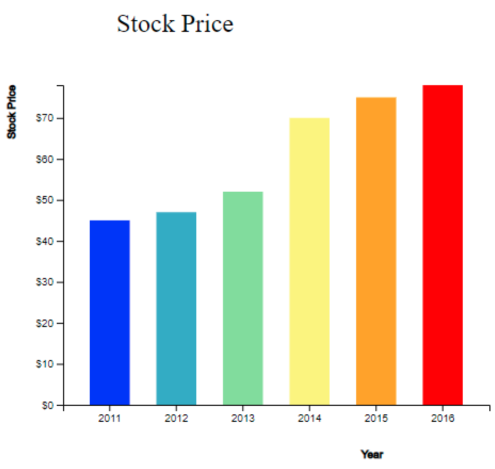
```
var colours = {
    46 : "blue",
    50 : "#4BABC5",
    60 : "#8CDE98",
    74 : "#F9F771",
    77 : "orange",
    80 : "red"
}
```

Assigning colours:



```
.style("fill", function(d) {
    for(const i in colours) {
        if(d.value < i) {
            console.log(colours[i]);
            return colours[i];
        }
    }
});
```

Result:

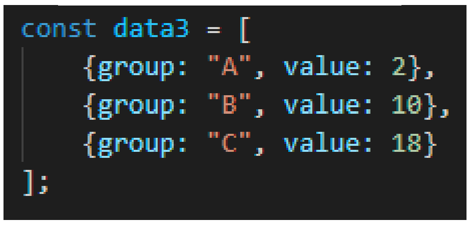


Part 5. Changing Data and Transitioning

Exercise 18.

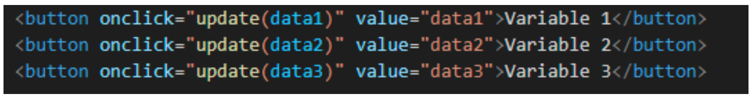
To add more data to the visual, I created another data array and gave each group a random value. I then added another button that when clicked, called the update function and sends the “data3” object as an argument.

Declaring new data:



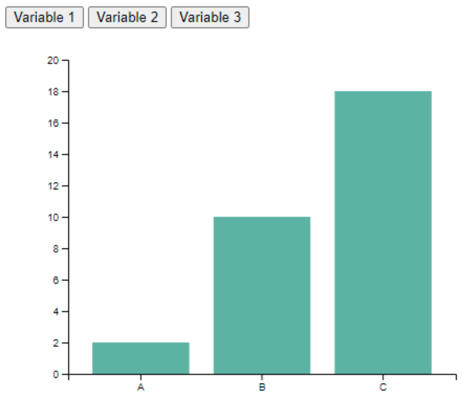
```
const data3 = [
    {group: "A", value: 2},
    {group: "B", value: 10},
    {group: "C", value: 18}
];
```

Adding a new Button:



```
<button onclick="update(data1)" value="data1">Variable 1</button>
<button onclick="update(data2)" value="data2">Variable 2</button>
<button onclick="update(data3)" value="data3">Variable 3</button>
```

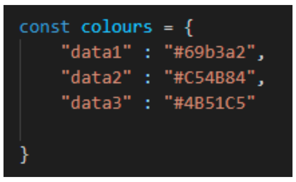
Result:



Exercise 19.

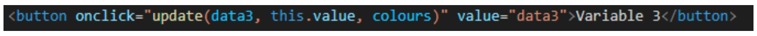
To achieve exercise 19, I used a similar approach as in exercise 17 whereby I created an object where the keys are the data arrays and the values are colours for the bars. These are then assigned to the correct dataset through a function in the attr("fill"…) function. If the name of the dataset being used (passed as an argument in update function when it is called by the button) matches the key, the colour that is the value is used to fill the bars.

Creating the colour object:



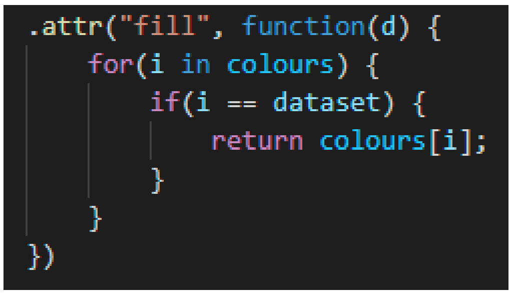
```
const colours = {
    "data1" : "#69b3a2",
    "data2" : "#C54B84",
    "data3" : "#4B51C5"

}
```

Passing the arguments through the update function:



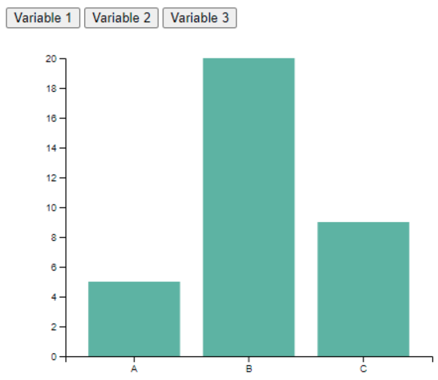
```
<button onclick="update(data3, this.value, colours)" value="data3">Variable 3</button>
```

Updating the colours:

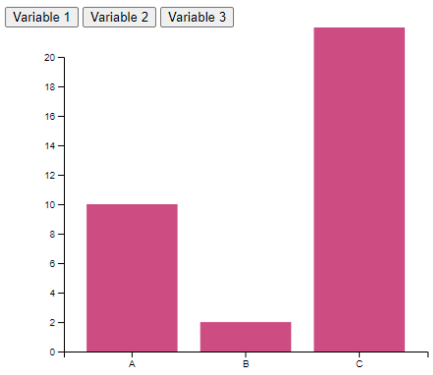


```
.attr("fill", function(d) {
    for(i in colours) {
        if(i == dataset) {
            return colours[i];
        }
    }
})
```

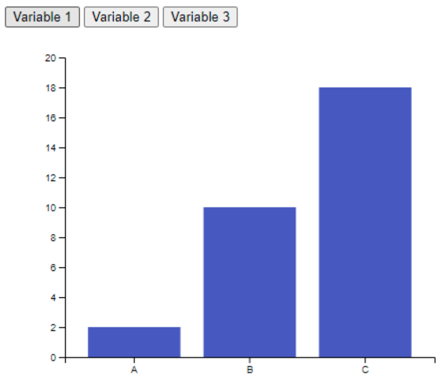
Variable 1 Colour:



Variable 2 Colour:



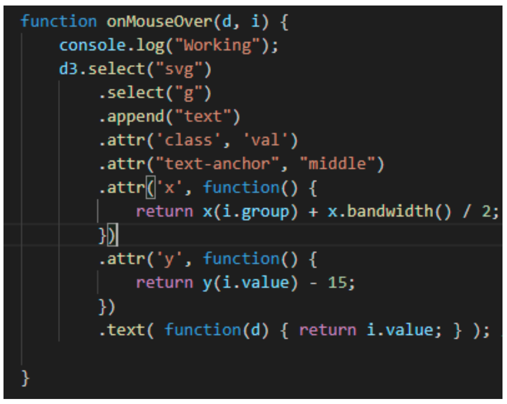
Variable 3 Colour:



Exercise 20.

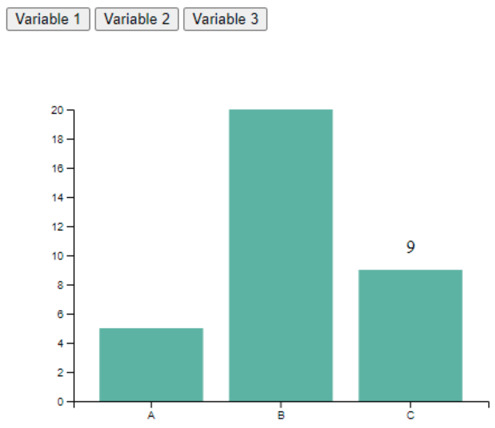
For this exercise I used the same principles as in exercise 16 whereby I used half of the bandwidth() function to align the text over the centre of the bar.

Code:



```
function onMouseOver(d, i) {
    console.log("Working");
    d3.select("svg")
        .select("g")
        .append("text")
        .attr('class', 'val')
        .attr("text-anchor", "middle")
        .attr('x', function() {
            return x(i.group) + x.bandwidth() / 2;
        })
        .attr('y', function() {
            return y(i.value) - 15;
        })
        .text( function(d) { return i.value; } );

}
```

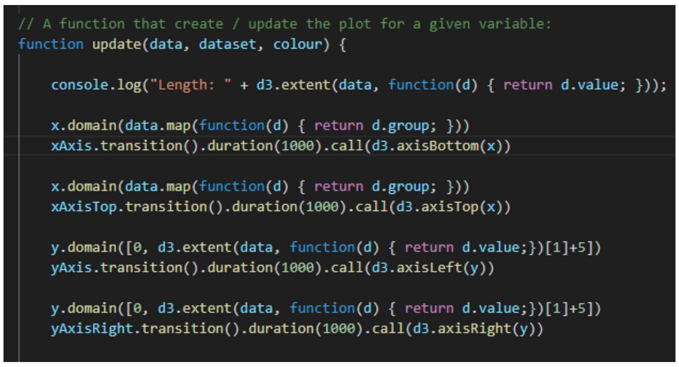
Result:



Exercise 22:

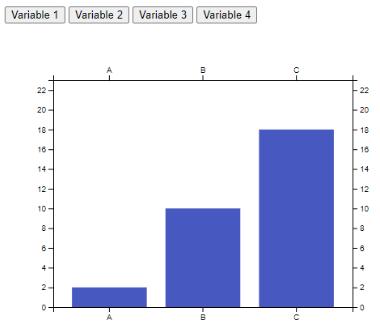
Firstly, I created a new array with 4 data objects, added a fourth colour to the colour object and added a new button so that there was a new variable able to be displayed with it's own colour. To update the axes in order to accommodate new data I updated the domain of the x and y axis variables (which are also used to set the top and right axis) inside the update function calling data.map(function(d) { return d.group; } to get the x axis variables of the new data. I did the same for the y axis data by calling d3.extent(data, function(d) { return d.value; }[1] to get the uppser value of the y axis data. I then updated the x and y axis by calling a transition on the xAxis and y Axis variables and calling the new axes.

Code:



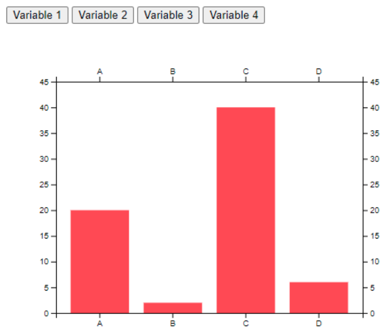
```
// A function that create / update the plot for a given variable:
function update(data, dataset, colour) {

    console.log("Length: " + d3.extent(data, function(d) { return d.value; }));

    x.domain(data.map(function(d) { return d.group; }))
    xAxis.transition().duration(1000).call(d3.axisBottom(x))

    x.domain(data.map(function(d) { return d.group; }))
    xAxisTop.transition().duration(1000).call(d3.axisTop(x))

    y.domain([0, d3.extent(data, function(d) { return d.value;})[1]+5])
    yAxis.transition().duration(1000).call(d3.axisLeft(y))

    y.domain([0, d3.extent(data, function(d) { return d.value;})[1]+5])
    yAxisRight.transition().duration(1000).call(d3.axisRight(y))
```

Variable 3:



s

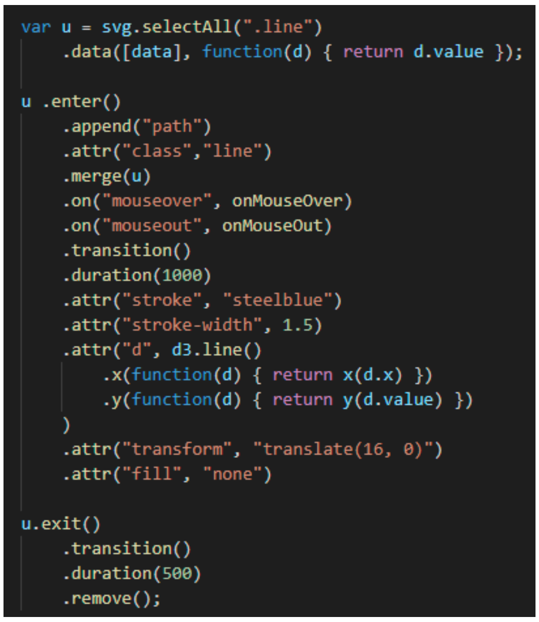
Variable 4:



Exercise 23.

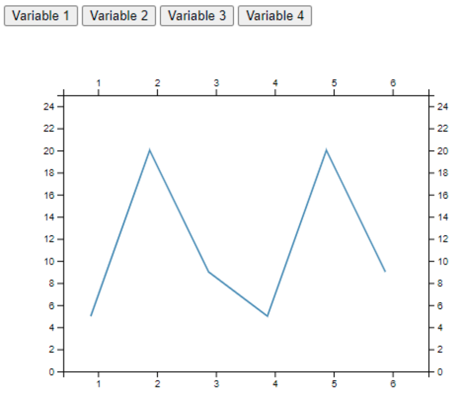
I tried a few approaches to complete this exercise as initially I found the difficulty to be that when the update function is called and the data is bound to the path calling svg.selectAll("path"), this would also affect all of the axis paths in the SVG too. To overcome this, I added a class to the path that formed the line on the graph and used that as the selector in the svg.selectAll when updating the data as that way only the relevant path was selected and the axis paths were left unaffected.

Code:

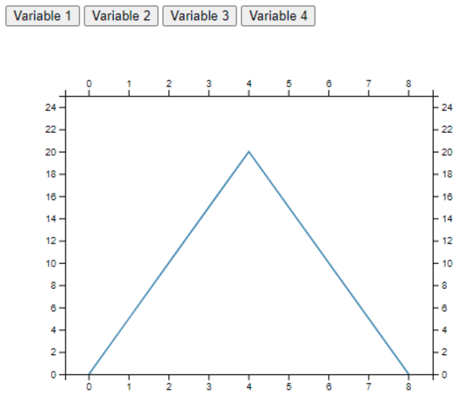


```
var u = svg.selectAll(".line")
    .data([data], function(d) { return d.value });

u .enter()
    .append("path")
    .attr("class","line")
    .merge(u)
    .on("mouseover", onMouseOver)
    .on("mouseout", onMouseOut)
    .transition()
    .duration(1000)
    .attr("stroke", "steelblue")
    .attr("stroke-width", 1.5)
    .attr("d", d3.line()
        .x(function(d) { return x(d.x) })
        .y(function(d) { return y(d.value) })
    )
    .attr("transform", "translate(16, 0)")
    .attr("fill", "none")

u.exit()
    .transition()
    .duration(500)
    .remove();
```

Variable 1:

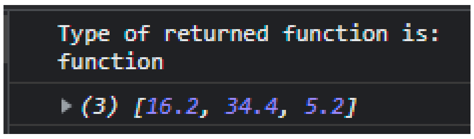


Variable 4:



Exercise 24.

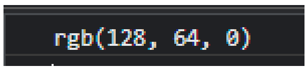
The output (shown below) occurs because the interpolate function takes in as arguments two arrays of 3 elements and returns a function that returns intermediate values between that values in the two arrays. For example calling intr(0) would return 20 as the first value in the returned array and intr(1) would return 1 as they are the upper and lower values of the interpolate function, whereas intr(0.5) returns 10.5 as the first value as it is the value half way between 1 and 20.



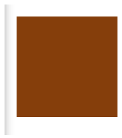
```
Type of returned function is:
function
▸ (3) [16.2, 34.4, 5.2]
```

Exercise 25.

The returned colours is rgb(128, 64, 0) which is a dark orange/brown as the interpolated function is given 0.5 as an argument, it returns a colour that is half way between red and green.



```
rgb(128, 64, 0)
```



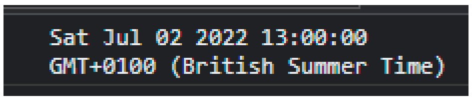
Exercise 26.

Two interpolate two dates, I used the d3.interpolateDate function, passing in as arguments two date objects which form the lower and upper bound values of the returned interpolate function. When called with a value between 0 and 1, it returns a date object between the two dates used to create the function.

Code:

```
let date = d3.interpolateDate(new Date("2022-01-01"), new Date("2023-01-01"))
console.log( date(0.5) );
```

Result:

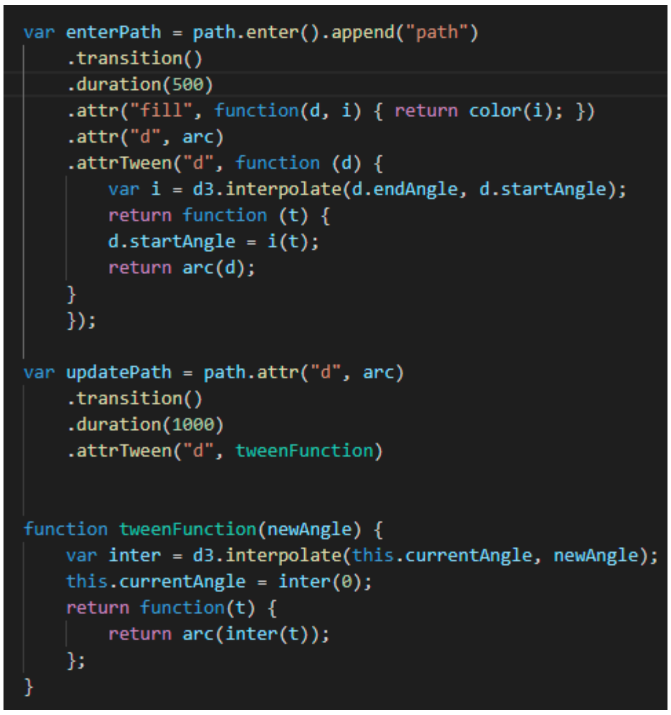


```
Sat Jul 02 2022 13:00:00
GMT+0100 (British Summer Time)
```

Exercise 27.

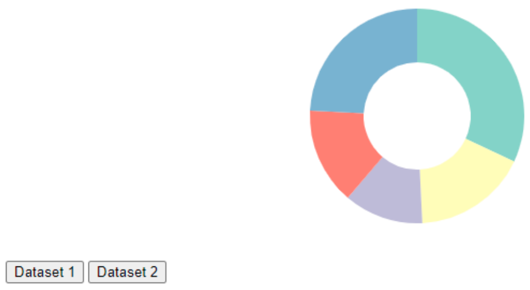
To get the pir chart to transition smoothly I created a separate tween function that took in as an argument the angle of the new data that the pie paths were going to have to reach. The current angles are also stored in the path object and this is used in with the new angle argument to create a new interpolate function where the lower value is the current angle and the upper value of the new angle. The tween function then returns a function that will return each of the angles necessary smoothly transition the pie from it's current values to the new data values.

Code:

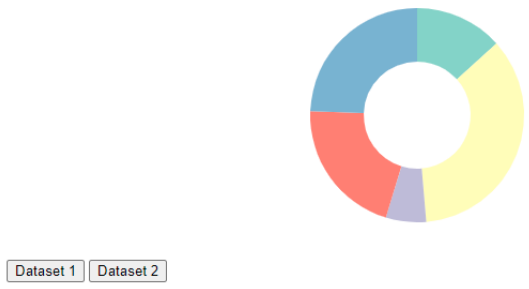


```
var enterPath = path.enter().append("path")
    .transition()
    .duration(500)
    .attr("fill", function(d, i) { return color(i); })
    .attr("d", arc)
    .attrTween("d", function (d) {
        var i = d3.interpolate(d.endAngle, d.startAngle);
        return function (t) {
        d.startAngle = i(t);
        return arc(d);
    }
    });

var updatePath = path.attr("d", arc)
    .transition()
    .duration(1000)
    .attrTween("d", tweenFunction)


function tweenFunction(newAngle) {
    var inter = d3.interpolate(this.currentAngle, newAngle);
    this.currentAngle = inter(0);
    return function(t) {
        return arc(inter(t));
    };
}
```

Dataset 1:



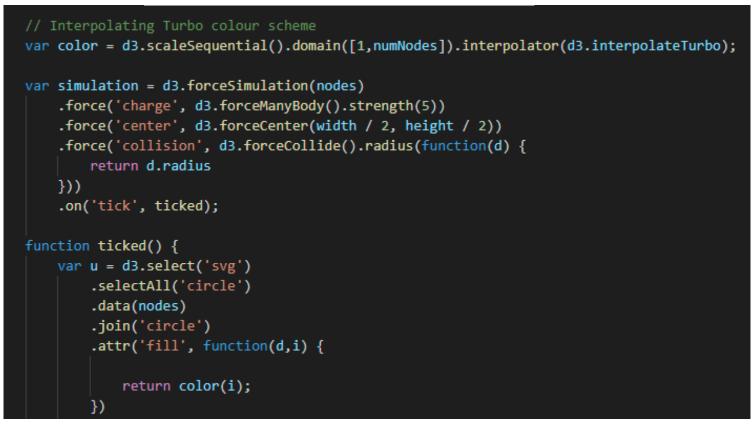
Dataset 2:



Exercise 28.

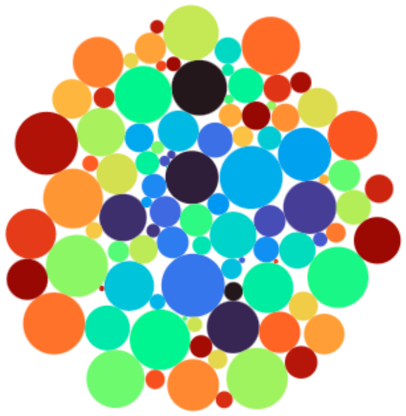
To give each circle a different colour I used an interpolator function using the d3.interpolateTurbo function as an argument. I then set the domain of the colour interpolator function to be between 1 and the number of circles in the dataset so that each circle is guaranteed to have its own colour as the index of each circle is its own colour in the interpolated colour function.

Code:



```
// Interpolating Turbo colour scheme
var color = d3.scaleSequential().domain([1,numNodes]).interpolator(d3.interpolateTurbo);

var simulation = d3.forceSimulation(nodes)
    .force('charge', d3.forceManyBody().strength(5))
    .force('center', d3.forceCenter(width / 2, height / 2))
    .force('collision', d3.forceCollide().radius(function(d) {
        return d.radius
    }))
    .on('tick', ticked);

function ticked() {
    var u = d3.select('svg')
        .selectAll('circle')
        .data(nodes)
        .join('circle')
        .attr('fill', function(d,i) {

            return color(i);
        })
```

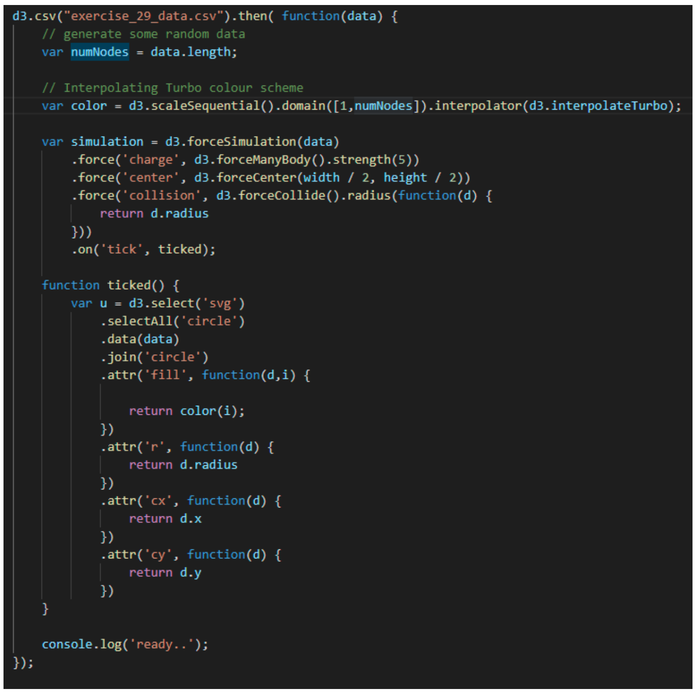
Result:



Exercise 29.

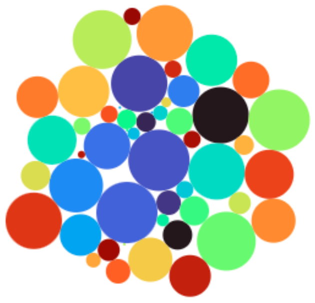
For this exercise, I read in the data from a csv and so wrapped the code to create and colour the circles in a d3,csv function. I also reduced the number of circles to 50 in the csv.

Code:



```
d3.csv("exercise_29_data.csv").then( function(data) {
    // generate some random data
    var numNodes = data.length;

    // Interpolating Turbo colour scheme
    var color = d3.scaleSequential().domain([1,numNodes]).interpolator(d3.interpolateTurbo);

    var simulation = d3.forceSimulation(data)
        .force('charge', d3.forceManyBody().strength(5))
        .force('center', d3.forceCenter(width / 2, height / 2))
        .force('collision', d3.forceCollide().radius(function(d) {
            return d.radius
        }))
        .on('tick', ticked);

    function ticked() {
        var u = d3.select('svg')
            .selectAll('circle')
            .data(data)
            .join('circle')
            .attr('fill', function(d,i) {

                return color(i);
            })
            .attr('r', function(d) {
                return d.radius
            })
            .attr('cx', function(d) {
                return d.x
            })
            .attr('cy', function(d) {
                return d.y
            })
    }

    console.log('ready..');
});
```

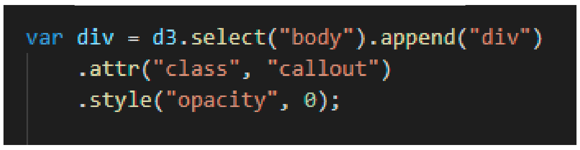
Result:



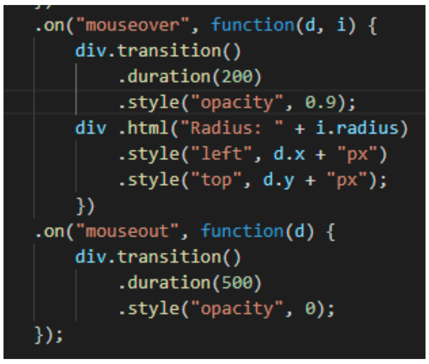
Exercise 30.

To display extra information, when the mouse hovers each circle a div that has been added to the screen is placed near the circle using the x and y coordinates of the mouseover event and goes from being transparent to visible and displays the radius of the circle. Upon the mouseout function, the div becomes transparent again and cannot be seen. Using a div means there are no circled overlapping the information box and it can be easily styled using css.

Code:

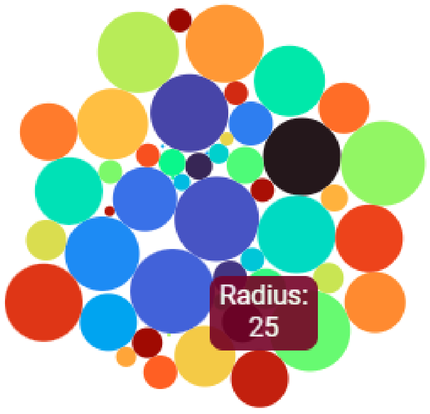


```
var div = d3.select("body").append("div")
    .attr("class", "callout")
    .style("opacity", 0);
```



```
.on("mouseover", function(d, i) {
    div.transition()
        .duration(200)
        .style("opacity", 0.9);
    div .html("Radius: " + i.radius)
        .style("left", d.x + "px")
        .style("top", d.y + "px");
    })
.on("mouseout", function(d) {
    div.transition()
        .duration(500)
        .style("opacity", 0);
});
```

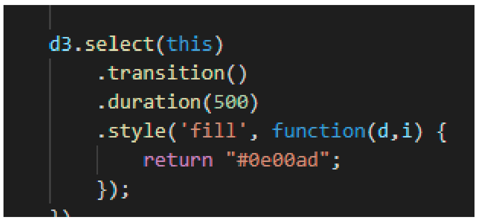
Result:



Exercise 31.

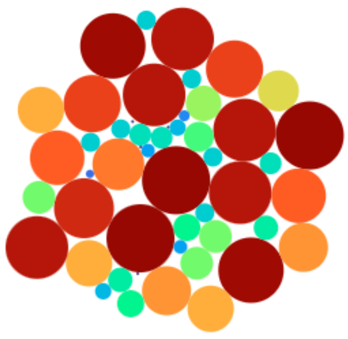
To change the colour of the circles when the mouse hovers it I added a d3.select(this) to the mouseover event and added a transition to change the style("fill"…) property to a shade of blue. When the mouse moves away from the circle the same transition happens in reverse to return the circle to its original colour using the interpolateTurbo colour palette and the circles radius as the value.

Code:

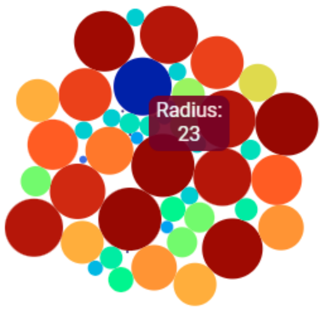


```
d3.select(this)
    .transition()
    .duration(500)
    .style('fill', function(d,i) {
        return "#0e00ad";
    });
```

Before mouseover event:



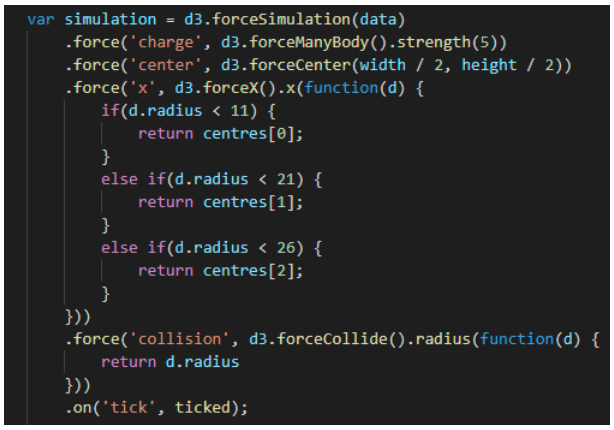
After mouseover event:



Exercise 32.

The extra force I added was the d3.forceX() and I split the circles into 3 groups: circles with a radius of 10 or less, circles with a radius between 11 and 20 and circles with a radius between 21 and 25. This successfully split the data into 3 groups along the x axis of the SVG and added another component to visualising the data in addition to their colour being based on their radius.

Code:



```
var simulation = d3.forceSimulation(data)
    .force('charge', d3.forceManyBody().strength(5))
    .force('center', d3.forceCenter(width / 2, height / 2))
    .force('x', d3.forceX().x(function(d) {
        if(d.radius < 11) {
            return centres[0];
        }
        else if(d.radius < 21) {
            return centres[1];
        }
        else if(d.radius < 26) {
            return centres[2];
        }
    }))
    .force('collision', d3.forceCollide().radius(function(d) {
        return d.radius
    }))
    .on('tick', ticked);
```

Result: