F21DV Coursework Lab 2 Report

Jonathan Song Yang, Lee (H00255553)

Demonstrated to: Amit Parekh (11/02/2022)

Contents

1 Introduction **2**
1.1 Set-up . . . 2

2 Exercises **4**
2.1 Exercise 1 . . . 4
2.2 Exercise 2 . . . 5
2.3 Exercise 3 . . . 6
2.4 Exercise 4 . . . 7
2.5 Exercise 10 . . . 7
2.6 Exercise 5 . . . 8
2.7 Exercise 6 . . . 9
2.8 Exercise 7 . . . 11
2.9 Exercise 8 . . . 11
2.10 Exercise 9 . . . 12
2.11 Exercise 11 . . . 13
2.12 Exercise 12 . . . 14
2.13 Exercise 13 . . . 15
2.14 Exercise 14 . . . 16
2.15 Exercise 15 . . . 17
2.16 Exercise 16 . . . 18
2.17 Exercise 17 . . . 19
2.17.1 Explaination for Exercise 15 to 17. . . . 22
2.18 Exercise 18 . . . 23
2.19 Exercise 19 . . . 26
2.20 Exercise 20 . . . 27
2.21 Exercise 21 . . . 28
2.22 Exercise 22 . . . 29
2.23 Exercise 23 . . . 30
2.24 Exercise 24-26 . . . 31
2.25 Exercise 27 . . . 32
2.26 Exercise 28 . . . 33
2.27 Exercise 29 . . . 34
2.28 Exercise 30 . . . 35
2.29 Exercise 31 . . . 36
2.30 Exercise 32 . . . 37

1 Introduction

Lab 2 focuses on using `d3.js` for dynamic and interactive visualisation concepts. It was meant to be a step-up from Lab 1, where we were taught the basics of d3.js.

Github Repo: https://github.com/jonleesy/F21DV-Coursework

Github Pages: https://jonleesy.github.io/F21DV-Coursework/public

1.1 Set-up

The set-up for this lab is the same as the Lab 1, but instead of using hard coded div properties, I have implemented CSS grid for aligning divs and webpage objects, as recommended by my lab 1's lab helper.

```
1     /**
2     * Create div's for each question systematically.
3     * @param {*} exerciseNumber Task number.
4     */
5    export function createDiv(exerciseNumber) {
6        d3.select('body')
7            .append('div')
8                .attr('class', 'container')
9                .append('div')
10                   .attr('class', 'answerCenter')
11                   .append('p')
12                       .append('strong')
13                           .text('Exercise ' + exerciseNumber + ':')
14   }
```

Listing 1.1: Old Method

```
1     /**
2     * Similar to createDiv(). Was told to look into
3     * grid instead of using hard coded div settings.
4     * This one focuses on that, and will be used starting
5     * from lab2.
6     * @param {*} exerciseNumber
7     */
8    export function createAnswerDiv(exerciseNumber) {
9        d3.select('body')
10           .append('div')
11               .attr('class', 'grid-container')
12               .append('div')
13                   .attr('class', 'title-grid')
14                   .append('p')
15                           .append('strong')
16                               .text('Exercise ' + exerciseNumber + ':')
17       d3.select('.grid-container')
18           .append('div')
19           .attr('class', 'answer-grid')
20   }
```

Listing 1.2: New Method

```
    /* Part 2 onwards CSS using grid */
    .grid-container {
        display: grid;
        grid-template-columns: auto 100px 100px 100px 100px auto;
        grid-template-rows: 60px auto auto auto auto auto;
        grid-gap: 10px;
        /* background-color: #262c3046; */
        padding: 10px;
    }

    .grid-container > div {
        background-color: rgb(226, 238, 240);
        padding: 20px 0;
    }

    .title-grid {
        grid-area: 1 / 2 / 1 / 6;
        text-align: left;
        text-indent: 20%;
    }

    .answer-grid {
        grid-area: 2 / 2 / 6 / 6;
        text-align: center;
        align-content: center;
    }

    .answer-grid-small {
        grid-area: 2 / 3 / 6 / 5;
        text-align: center;
        align-content: center;
    }
```

Listing 1.3: New CSS Method

Looking at listing 1.1 and 1.2, the only difference is with the new grid (`grid-container`) being added, then adding a smaller answer fiv with class `answer-grid` afterwards. The preperties of these grids above are shown in listing 1.3.

2 Exercises

2.1 Exercise 1

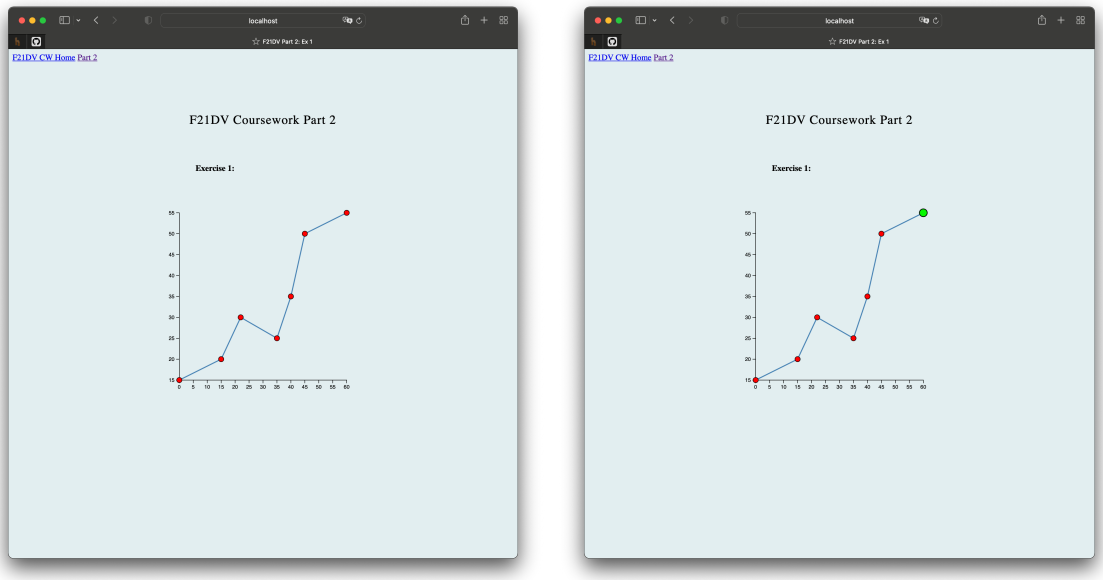

Figure 2.1: Exercise 1

Figure 2.1 shows a line chart with data points plotted on it. The right figure shows what happens when the mouse hovers upon a data point, the data point would pulse between red and green. **Can't seem to take a screenshot and capture the pointer.*

2.2 Exercise 2

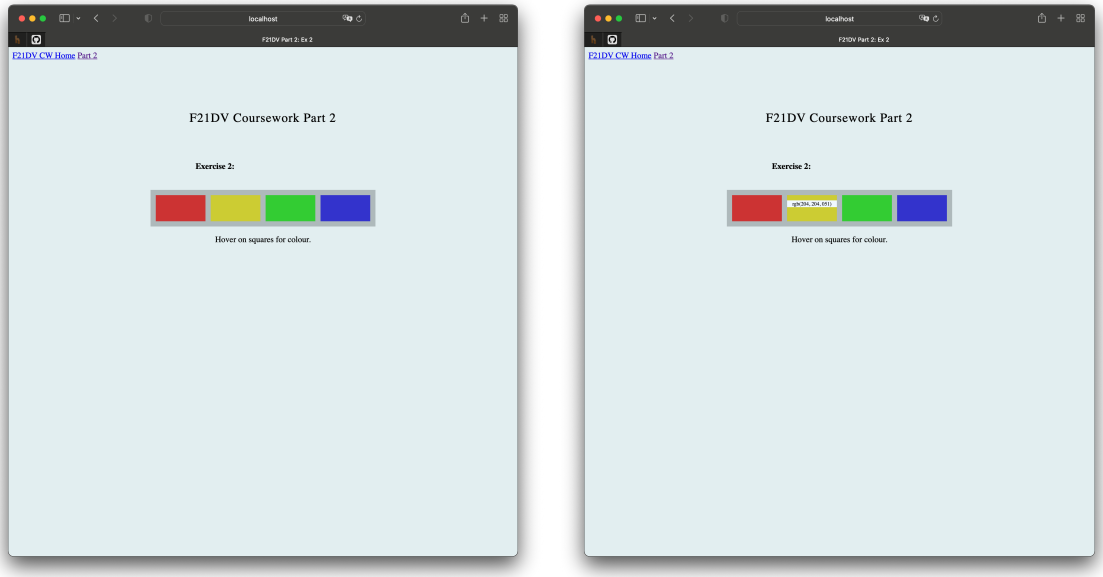

Figure 2.2: Exercise 2

Figure 2.2 shows 4 blocks with different colours. The blocks are just a div defined within a grid. Each block also has a fill colour defined using the `Rgb()` colour method, as shown in line 10. Upon hovering the mouse on the div, the div would show a text element displaying the Rgb colour. This effect is done using the css `:hover` method.

2.3 Exercise 3

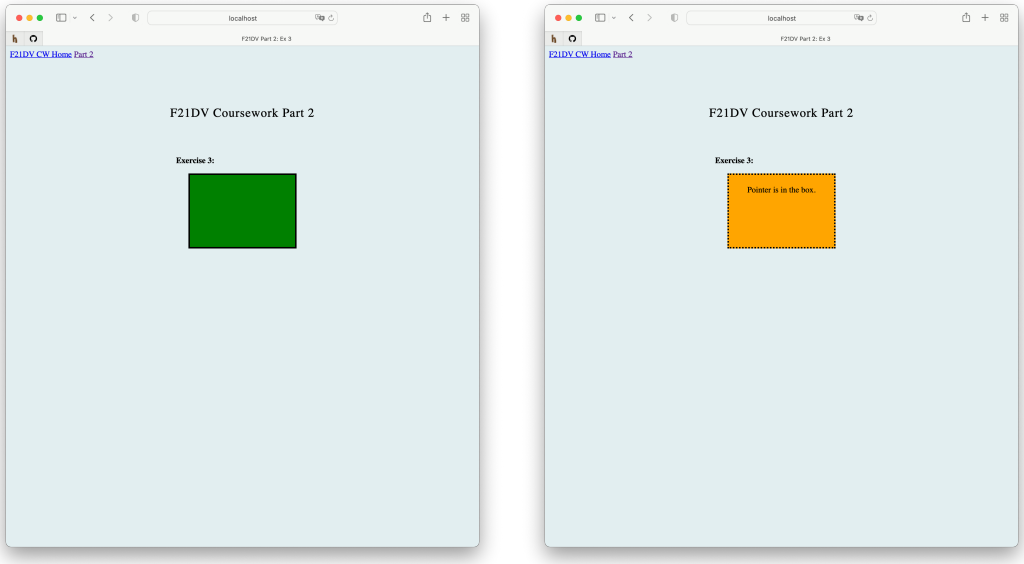

Figure 2.3: Exercise 3

```
1  // .js script for exercise:
2  const ex = 3;
3
4  // Imports of functions.
5  import { createAnswerDiv } from '../functions.js';
6
7  // Creating base <div>s systematically.
8  createAnswerDiv(ex);
9
10 // Adding the original div.
11 d3.select('.grid-container')
12     .append('div')
13         .attr('class', 'answer-grid-small')
14         .style('width', 'auto')
15         .style('height', '100px')
16         .style('background-color', 'green')
17         .style('border-style', 'solid');
18
19 // Change colour, border and text, using .on() mouse over and out.
20 d3.select('.answer-grid-small')
21     .on('mouseover', function() {
22         d3.select(this)
23             .style('background-color', 'orange')
24             .style('border-style', 'dotted')
25             .text('Pointer is in the box.');
26     })
27     .on('mouseout', function() {
28         d3.select(this)
29             .style('background-color', 'green')
30             .style('border-style', 'solid')
31             .text('');
32     });
```

../../public/js/part2/task3.js

Figure 2.3 shows a block thats green colour. Upon mouse hover on the box, the box now shows a text that says 'Pointer is in the box', its colour is now orange, and it has a new type of border. This is done using the `.on()` function.

2.4 Exercise 4

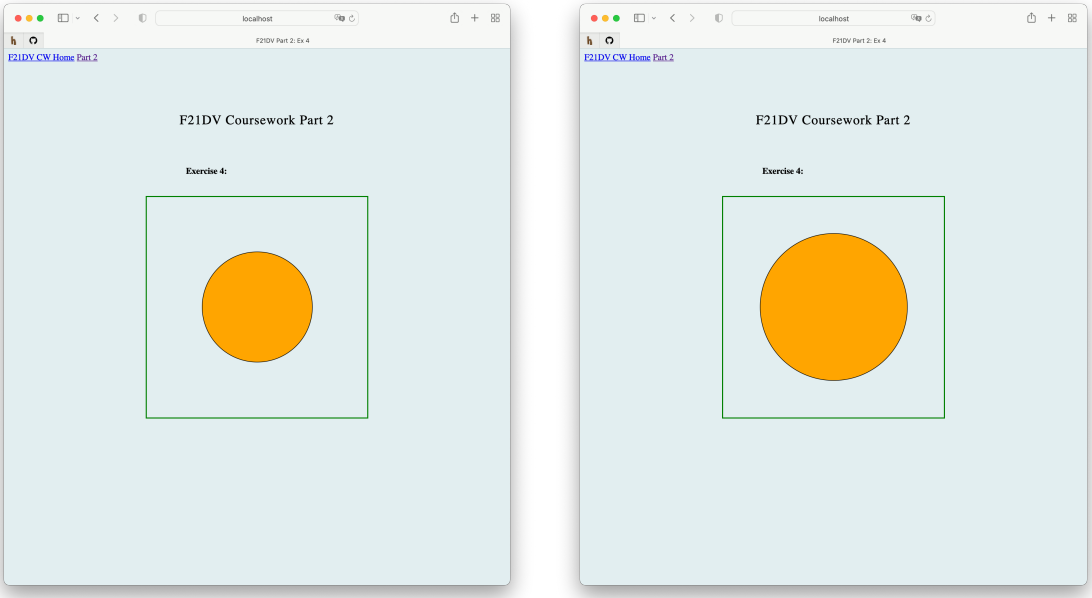

Figure 2.4: Exercise 4

Figure 2.4 shows an SVG object with a circle in the middle. Upon hovering on the circle, the circle enlarges. This time, the action was modeled using d3 transitions instead of css :hover method. I've added transtitions upon mouse hover and out.

2.5 Exercise 10

Exactly the same as exercise 4, just that the ease method has been changed to `d3.easeBounce`.

2.6 Exercise 5

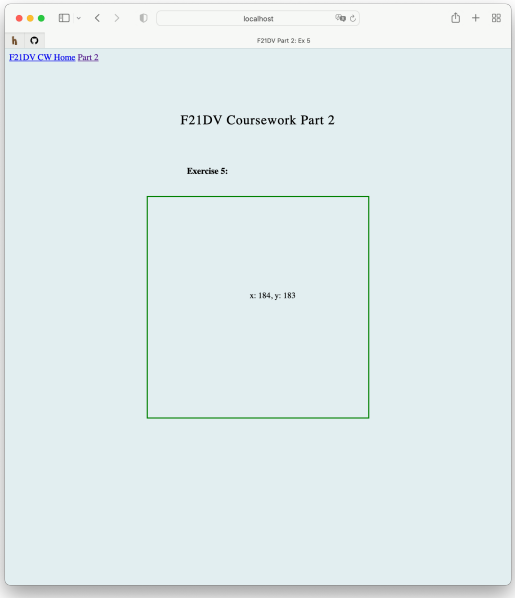

Figure 2.5: Exercise 5

Figure 2.6 shows an emptey svg object, and upon mouse hover, it would show the corredinated of the mouse. There was also a pre-appended empty text box. To show the x-y coordinates, this is done using the event data of the mouse movement. Then using the data, I modified the 'x' and 'y' attribute of the text box.

2.7 Exercise 6

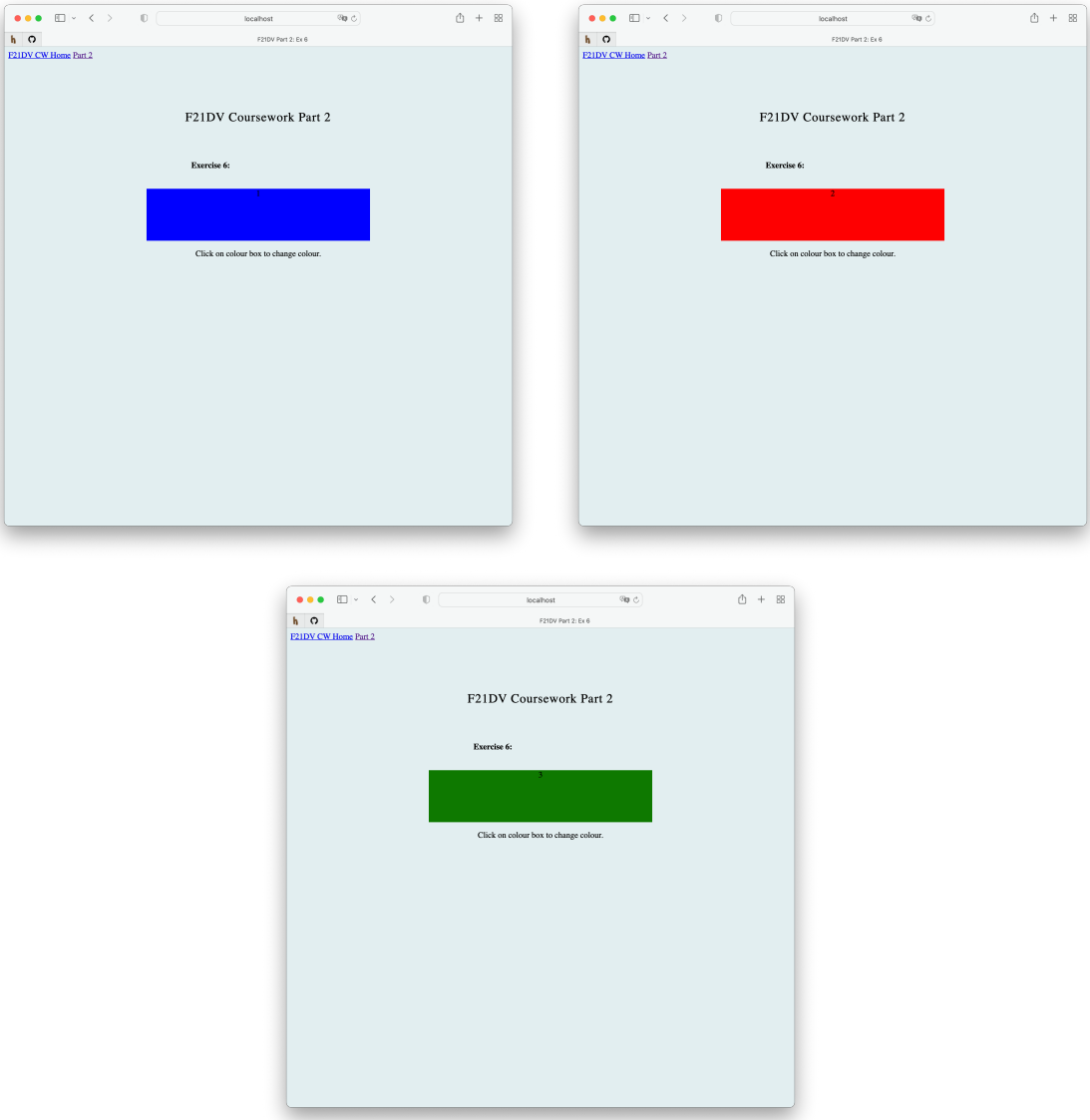

Figure 2.6: Exercise 6

```
// .js script for exercise:
const ex = 6;

// Imports of functions.
import { createAnswerDiv } from '../functions.js';

// Creating base <div>s systematically
createAnswerDiv(ex);

// Add original div.
d3.select('.answer-grid')
    .append('div')
        .style('width', 'auto')
        .style('height', '100px')
        .style('background-color', 'blue')
        .text('1');

// Add Transition.
d3.select('.answer-grid div')
    .on('click', function() {
```

```
21          d3.select(this)
22              // Ori -> 2(red)
23              .transition()
24                  .duration(1000)
25                  .style('background-color', 'red')
26                  .text('2')
27              // 2(red) -> 3(green)
28              .transition()
29                  .duration(1000)
30                  .style('background-color', 'green')
31                  .text('3')
32              // 3(green) -> Ori
33              .transition()
34                  .duration(1000)
35                  .style('background-color', 'blue')
36                  .text('1')
37      });
38
39  // Add Instructions.
40  d3.select('.answer-grid')
41      .append('p')
42      .text('Click on colour box to change colour.')
```

../../public/js/part2/task6.js

Figure 2.7 shows a coloured div with a number on it. On click, this div will transition to "2" and change to red, and then to "3" and then to green. Finally it will transition back to the original form. From listing above, we could see that this is done by using the transition upon mouse click, then changing the attributes of the div.

2.8 Exercise 7

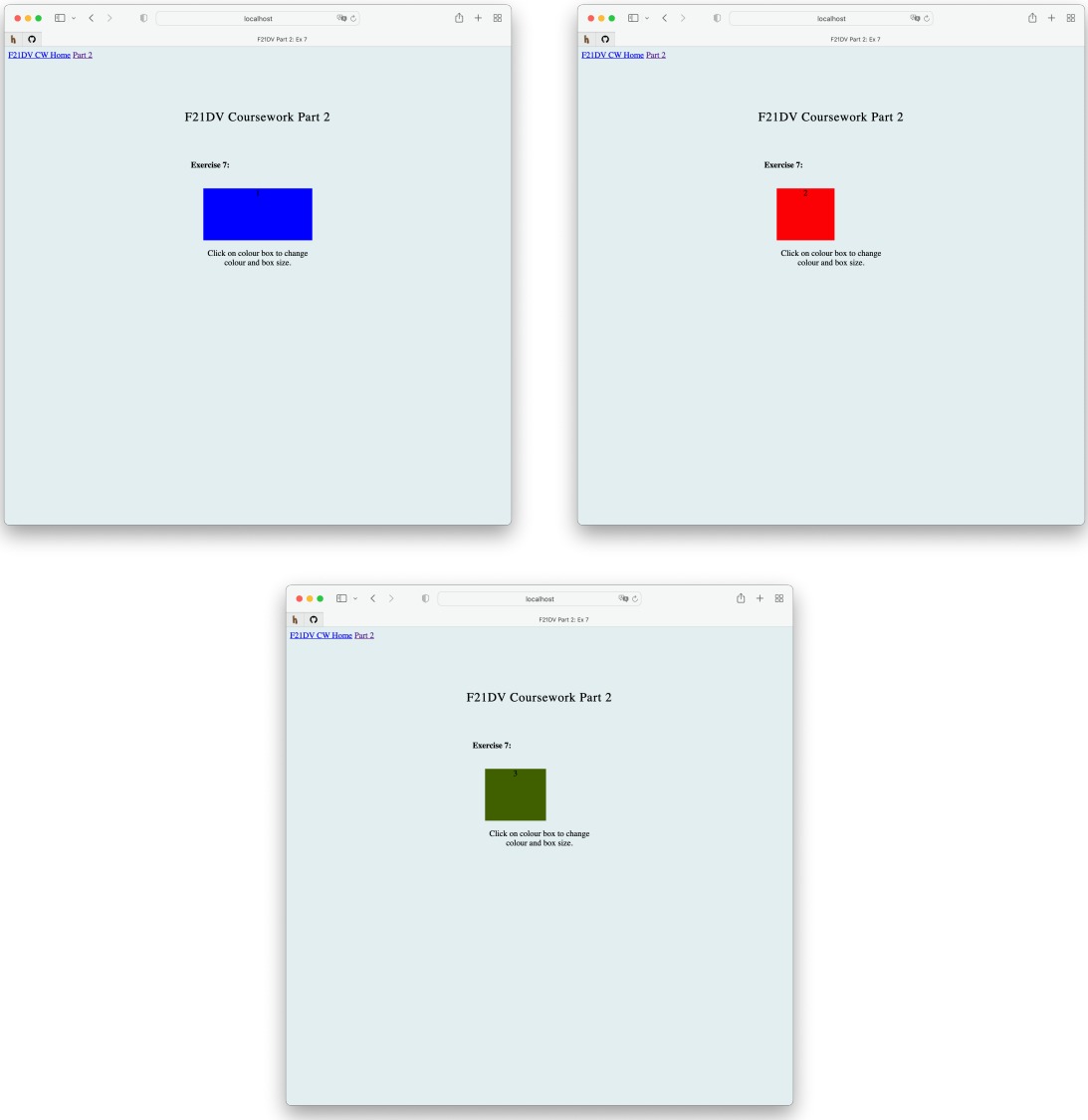

Figure 2.7: Exercise 7

Figure 2.8 shows the changes of a div upon clicking. This is pretty much the same as the one in exercise 6, except that I have added a transition to the size of the div.

2.9 Exercise 8

Exercise 8 is the same as 7, instead, the mouse action is now changed from click to hover.

2.10 Exercise 9

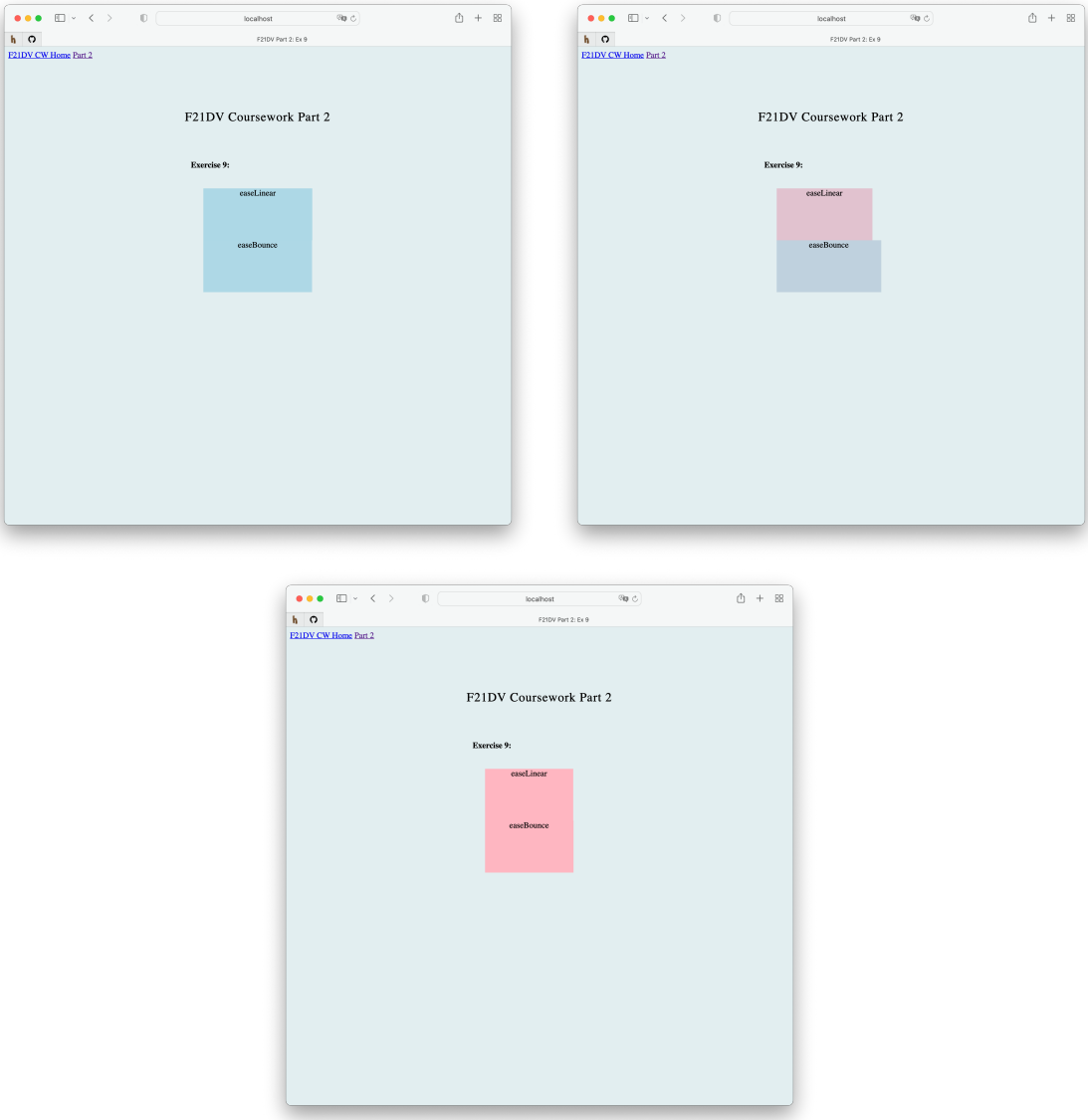

Figure 2.8: Exercise 9

Figure 2.10 shows the difference in the transition ease method. The figure shows two divs, one using the `d3.easeLinear` method and the other using `d3.easeBounce`. Both divs have the same transition endpoint and duration, but in between, due to the different ease method they woudl appear differently as shown in the second picture. This page also uses the d3 `.on('end', ...)` function, which allows the transtion to start on load of the page, and just loop through.

2.11 Exercise 11

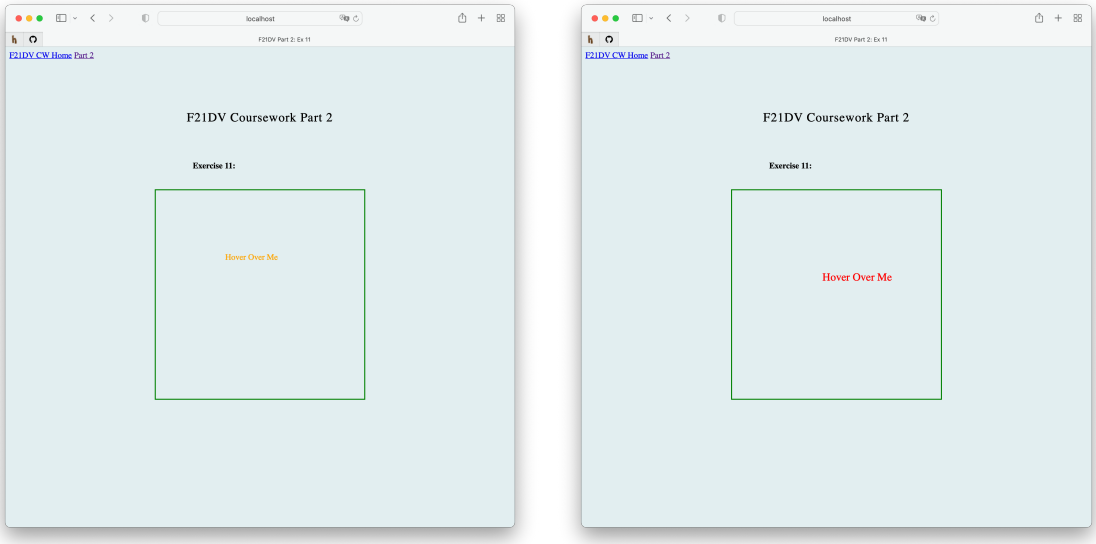

Figure 2.9: Exercise 11

Figure 2.11 shows a text on an svg, that changes colour and size on hover, and reverts back to its original state after the mouse leaves.

2.12 Exercise 12

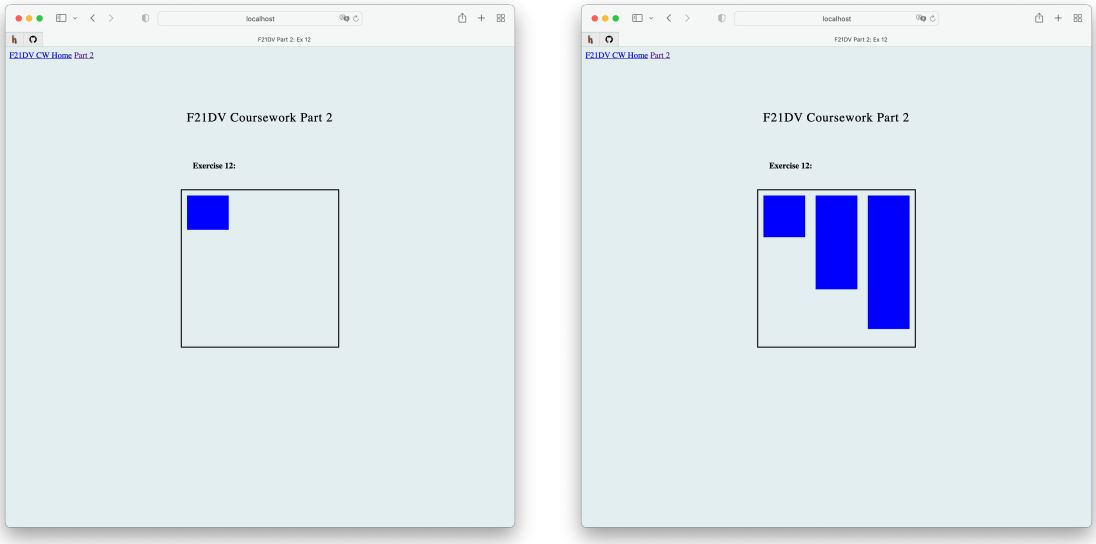

Figure 2.10: Exercise 12

Figure 2.12 shows an svg annimation that works on load. On load, three bars of different sizes start would start to form, one after another. This transition starts on page load. The transition between each block is being delayed so that they start one after another.

2.13 Exercise 13

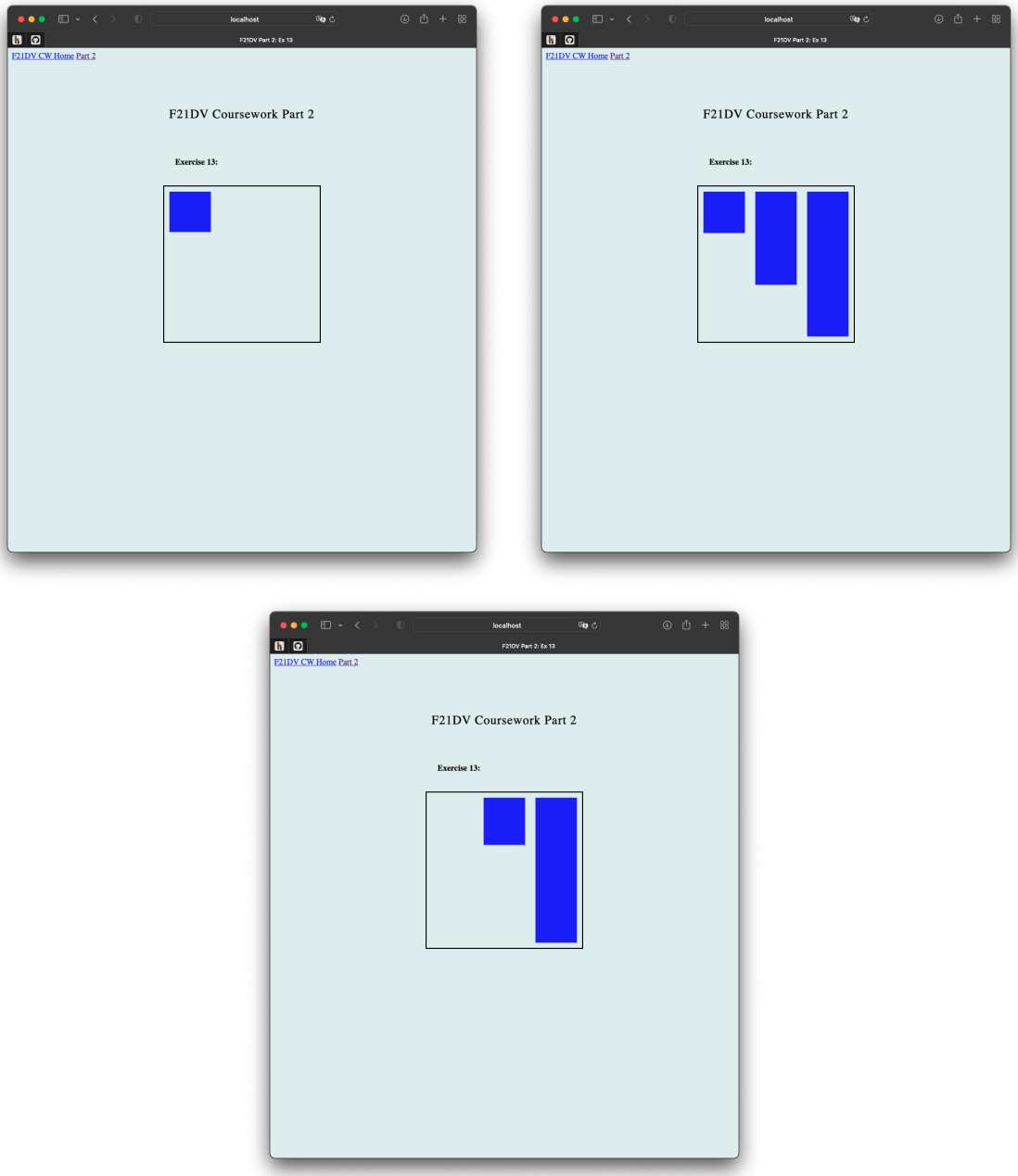

Figure 2.11: Exercise 13

Figure 2.13 shows the same svg animation as the one in exercise 12. This time, the annimation would annimate out just like the figure above, in the same sequence and the starting annimation.

2.14 Exercise 14

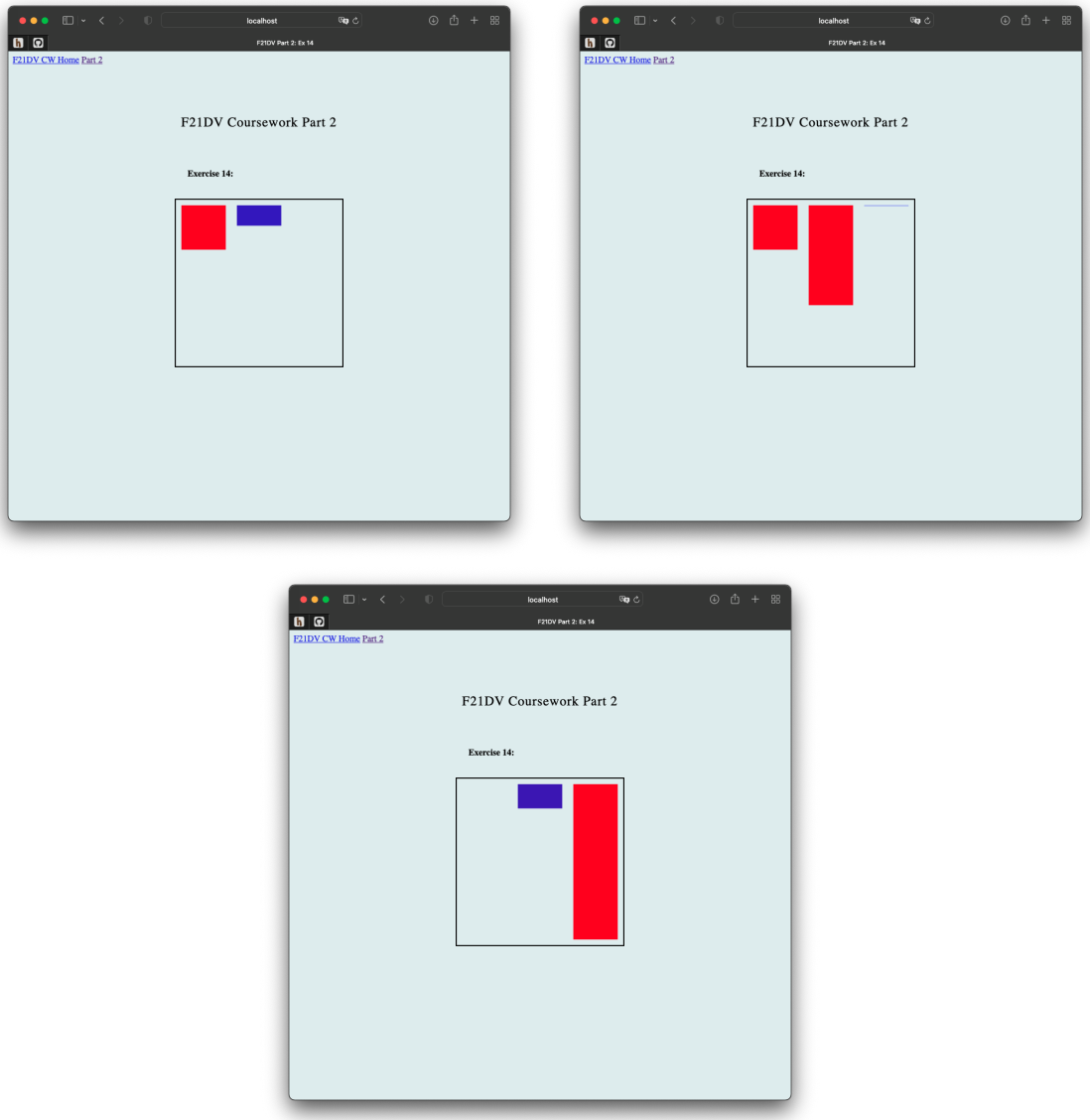

Figure 2.12: Exercise 14

Figure 2.14 shows the same svg animation as the one in exercise 13. THe difference is that now the annimation would change from blue to red upon the start annimation, and would change from red back to blue for the eciting annimation.

2.15 Exercise 15

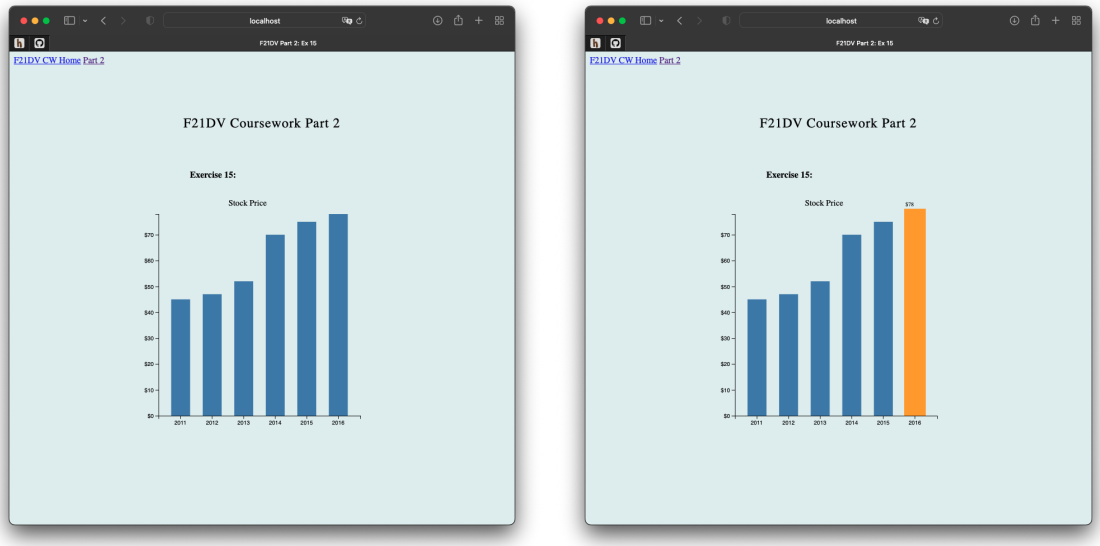

Figure 2.13: Exercise 15

Figure 2.15 shows a bar chart upon load. Upon hover of a specific bar chart, the bar chart would enlarge itself, and show its actual y-axis value (price of stock) on top of the bar.

2.16 Exercise 16

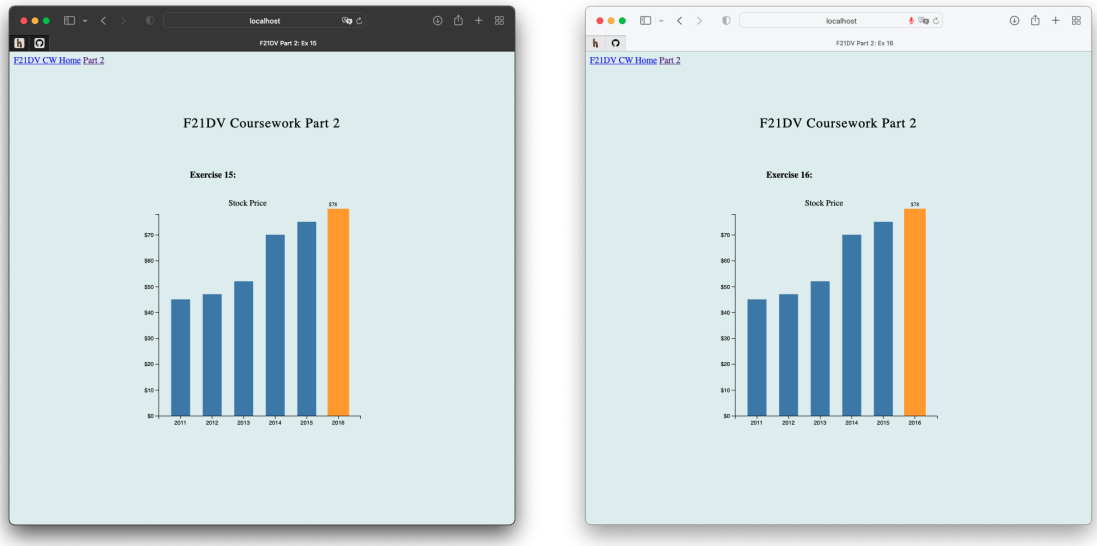

Figure 2.14: Exercise 16

Figure 2.16 shows a bar chart upon load, asme as the one shown in example 15, figure 2.15. The difference is that now on hover, the price shown on top of the bar is centered in the middle instead of on the side.

2.17 Exercise 17

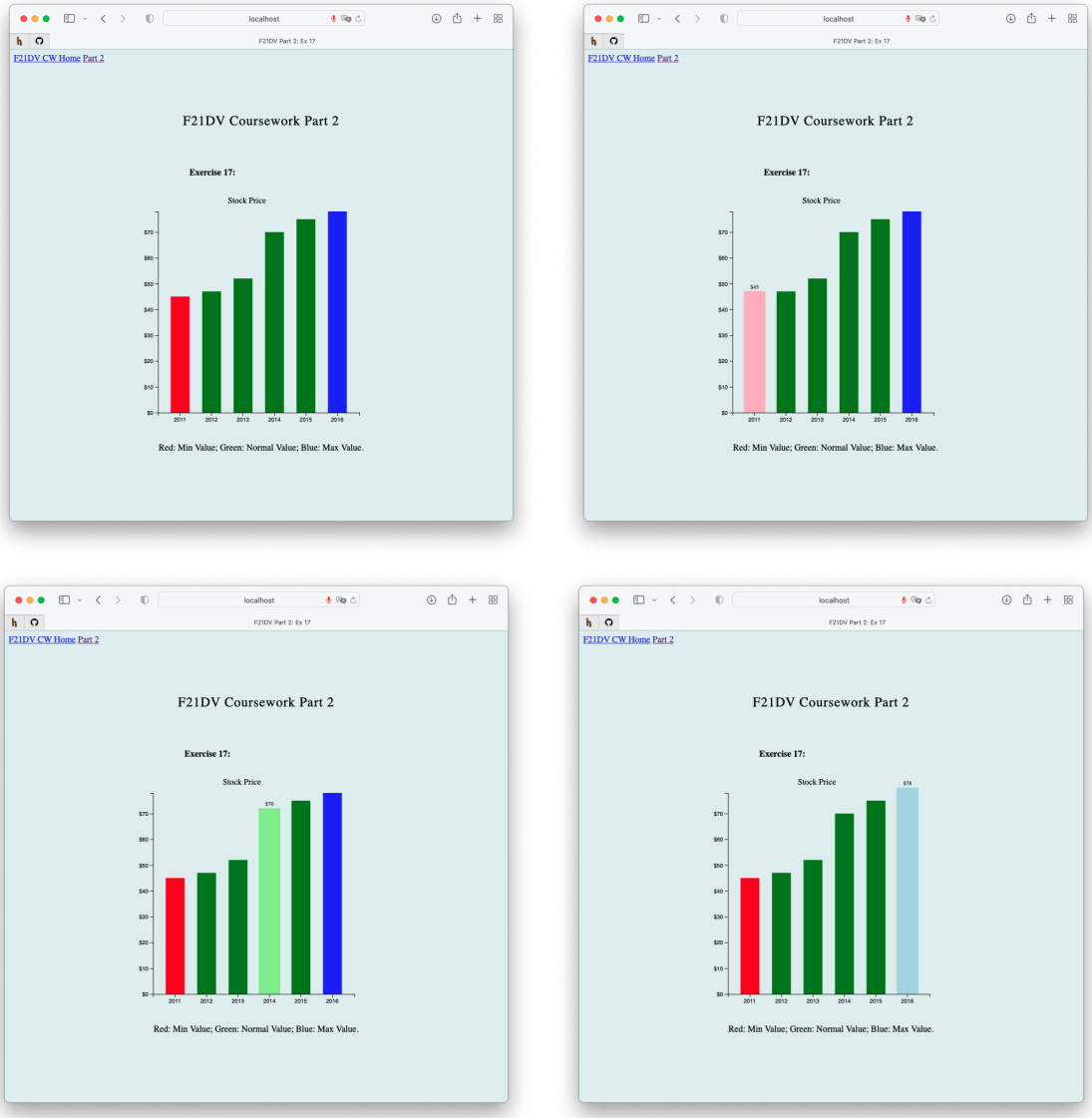

Figure 2.15: Exercise 17

```
// .js script for exercise:
const ex = 17;

// Imports of functions.
import { createAnswerDiv } from '../functions.js';

// Creating base <div>s systematically.
createAnswerDiv(ex);

// Reading in data for task 15.
const data15 = d3.csv('../../data/part2/task15.csv');

// Svg Constants.
let svgLength = 420;

// Adding the svg object.
const svg = d3.select('.answer-grid')
              .append('svg')
              .attr('width', svgLength)
              .attr('height', svgLength)
```

```
21
22 // More Constants.
23 let margin = 40,
24     length = svg.attr('width') - margin;
25
26 // Add Title
27 svg.append('text')
28     .attr('transform', 'translate(100,0)')
29     .attr('x', 50)
30     .attr('y', 10)
31     .attr('font-size', '15px')
32     .text('Stock Price');
33
34 // Defining the Axes.
35 const horScale = d3.scaleBand()
36                    .range([0, length])
37                    .padding(0.4);
38 const verScale = d3.scaleLinear()
39                    .range([length, 0]);
40
41 // Append a 'g' element.
42 const g = svg.append('g')
43                 .attr('transform', 'translate(25,25) scale(0.95)')
44
45 // Processing the Csv.
46 data15.then(function(data) {
47     // Local Variables for conditional fill later on.
48     let scaleMargin = 10,
49         minVal = d3.min(data, d => d.value),
50         maxVal = d3.max(data, d => d.value),
51         minCol = 'red', maxCol = 'blue';
52
53     // Defining the hor/ver axis domain.
54     horScale.domain(data.map(d => d.year));
55     verScale.domain([0, d3.max(data, d => d.value)]);
56
57     // Adding the x-axis.
58     g.append('g')
59         .attr('transform', `translate(0,${length})`)
60         .call(d3.axisBottom(horScale))
61         .append('text')
62         .attr('y', length - 150)
63         .attr('x', length - 100)
64         .attr('text-anchor', 'end')
65         .attr('stroke', 'black')
66         .text('Year');
67
68     // Add the y-axis.
69     g.append('g')
70         .call(d3.axisLeft(verScale).tickFormat(d => `$${d}`)
71             .ticks(10))
72         .append('text')
73             .attr('transform', 'rotate(-90)')
74             .attr('y', 6)
75             .attr('dy', '-5.1em')
76             .attr('text-anchor', 'end')
77             .attr('stroke', 'black')
78             .text('Stock Price');
79
80     // Add the Bars.
81     g.selectAll('.bar')
82         .data(data)
83         .join('rect')
84             .attr('class', 'bar')
85             .on('mouseover', onMouseOver)
86             .on('mouseout', onMouseOut)
```

```
87              .attr('x', d => horScale(d.year))
88              .attr('y', d => verScale(d.value))
89              .attr('width', horScale.bandwidth())
90              .style('fill', d => (d.value === minVal) ? minCol : ((d.value === maxVal)
     ? maxCol : 'green'))
91              .transition()
92                  .duration(400)
93                  .ease(d3.easeLinear)
94                  .delay((_, i) => i * 50)
95                  .attr('height', d => length - verScale(d.value))
96
97      // define mouse over function.
98      function onMouseOver(_, d) {
99          // Add a high light class.
100         d3.select(this)
101             .attr('class', 'highlight')
102             .transition()
103                 .duration(400)
104                 .attr('width', horScale.bandwidth() + scaleMargin/2)
105                 .attr('y', verScale(d.value) - scaleMargin)
106                 .attr('height', length - verScale(d.value) + scaleMargin)
107                 .attr('transform', `translate(${-scaleMargin/4},0)`)
108                 // Conditional Fill.
109                 .style('fill', (d.value === minVal) ? 'lightpink' : ((d.value ===
    maxVal) ? 'lightblue' : 'lightgreen'));
110         // Ass a text on top og the bar.
111         g.append('text')
112             .attr('class', 'val')
113             .attr('x', horScale(d.year) + scaleMargin)
114             .attr('y', verScale(d.value) - scaleMargin * 3/2)
115             .attr('font-size', '10px')
116             .text(`$${d.value}`);
117     }
118
119     // Mouse Out function.
120     function onMouseOut(_, d) {
121         d3.select(this)
122             .attr('class', 'bar')
123         // Undo bar change.
124         d3.select(this)
125             .transition()
126                 .duration(400)
127                 .attr('width', horScale.bandwidth())
128                 .attr('x', horScale(d.year))
129                 .attr('y', verScale(d.value))
130                 .attr('height', length - verScale(d.value))
131                 .attr('transform', `translate(0,0)`)
132                 .style('fill', (d.value === minVal) ? minCol : ((d.value === maxVal)
    ? maxCol : 'green'));
133         // Remove the text.
134         d3.selectAll('.val')
135             .remove()
136     }
137 });
138
139 // Colour explaination.
140 d3.select('.answer-grid')
141     .append('p')
142     .text('Red: Min Value; Green: Normal Value; Blue: Max Value.')
```

../../public/js/part2/task17.js

Figure 2.17 shows a bar chart upon load, same as the one shown in example 15 and 16. The difference is that the bars are now colour coded. Upon Hover, the bars would still show a different colour, as shown in the images above.

2.17.1 Explaination for Exercise 15 to 17.

Exercise 15-17's price tag text is transitioned by adding a `text` item each time on mouse over. And Removed everytime on mouse out.

2.18 Exercise 18

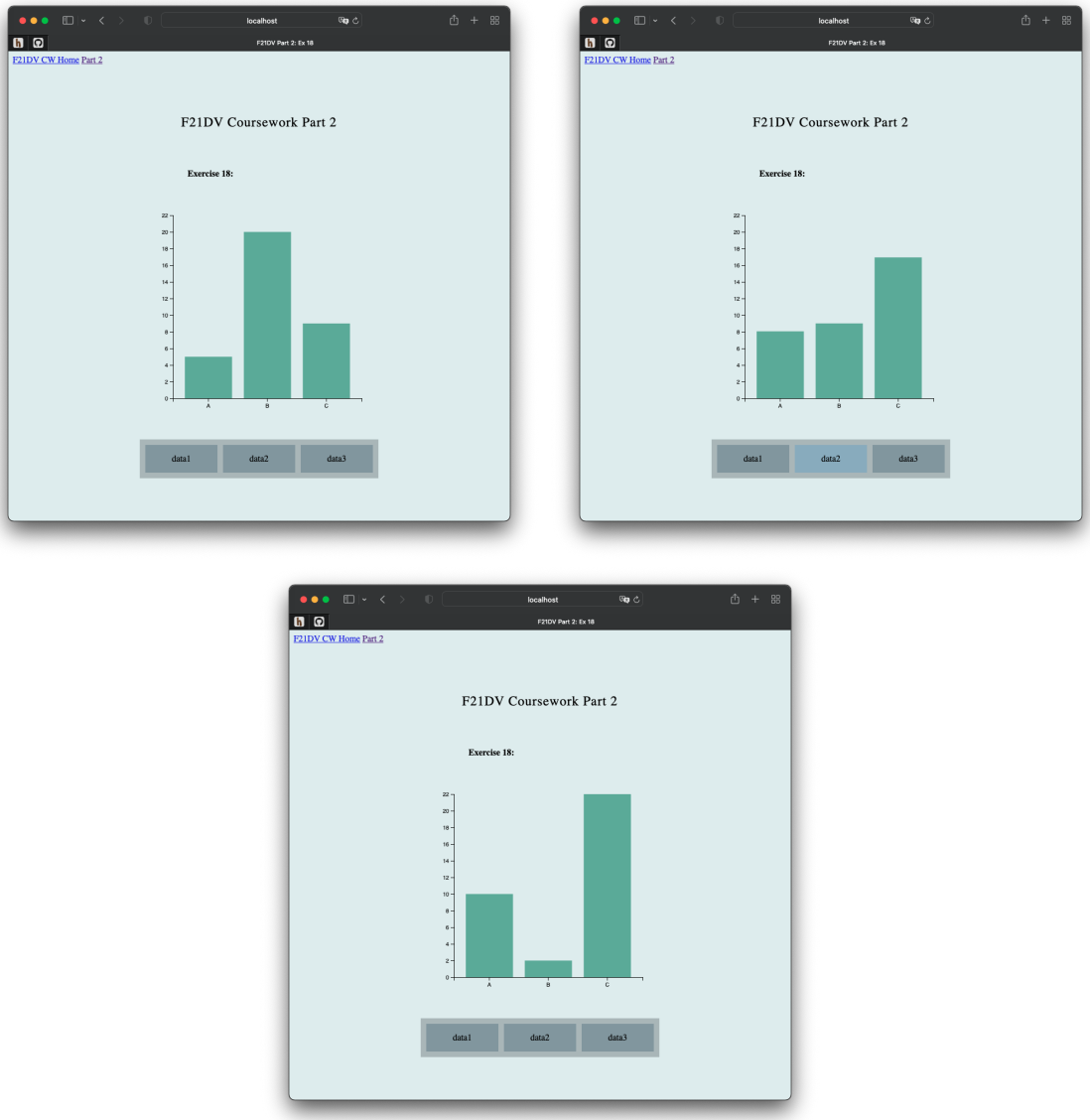

Figure 2.16: Exercise 18

```
// .js script for exercise:
const ex = 18;

// Imports of functions.
import { createAnswerDiv } from '../functions.js';

// Creating base <div>s systematically.
createAnswerDiv(ex);

// Creating the data
const data1 = [
    {group:'A', value:5},
    {group:'B', value:20},
    {group:'C', value:9}
];

const data2 = [
    {group:'A', value:10},
    {group:'B', value:2},
    {group:'C', value:22}
```

```
];

const data3 = [
    {group:'A', value:6},
    {group:'B', value:18},
    {group:'C', value:11}
];

// Svg and Graph Constants
const margin = {top: 30, right: 30, bottom: 70, left: 60},
        width = 400 - margin.left,
        height = 400 - margin.bottom;

// Create the svg object.
const svg = d3.select('.answer-grid')
                .append('div')
                    .append('svg')
                        .attr('width', width + margin.left + margin.right)
                        .attr('height', height + margin.top + margin.bottom)
                        .append('g')
                            .attr('transform', `translate(${margin.left}, ${margin.
    top})`);

// Define the Axes
const horScale = d3.scaleBand()
                    .range([0, width])
                    .domain(data1.map(d => d.group))
                    .padding(0.2);
const verScale = d3.scaleLinear()
                    .domain([0, 22])
                    .range([height, 0]);

// Adding the Axes
svg.append('g')
    .attr('transform', `translate(0, ${height})`)
    .call(d3.axisBottom(horScale));
svg.append('g')
    .attr('class', 'myYAxis')
    .call(d3.axisLeft(verScale));

// Bar Chart Function. Clears and add.
function update(data) {
    // Using a variable to implement a .merge() function.
    const u = svg.selectAll('rect')
                .data(data)

    // enter function.
    u.enter()
        .append('rect')
        .merge(u)
        .transition()
            .duration(1000)
                .attr('x', d => horScale(d.group))
                .attr('y', d => verScale(d.value))
                .attr('width', horScale.bandwidth())
                .attr('height', d => height - verScale(d.value))
                .attr('fill', '#69b3a2');
}

// Opening a data upon launch.
update(data1);

// Add buttons to activate data 1,2,3
const buttonGrid = d3.select('.answer-grid')
                    .append('div')
                        .attr('class', 'inner-grid');
```

```
86
87  // Button names for div population.
88  const buttonData = ['data1', 'data2', 'data3'];
89
90  // Add buttons (divs).
91  buttonGrid.selectAll('div')
92              .data(buttonData)
93              .join('div')
94                  .attr('class', d => d)
95                  .append('p')
96                      .text(d => d);
97
98  // Add mouse events.
99  d3.select('.data1')
100     .on('click', function() {
101         update(data1);
102     })
103 d3.select('.data2')
104     .on('click', function() {
105         update(data2);
106     })
107 d3.select('.data3')
108     .on('click', function() {
109         update(data3);
110     })
```

../../public/js/part2/task18.js

Figure 2.18 shows a bar chart that transitions into a start position showing data 1 upon load. Upon clicking on "Data 2", each bar would transition to the new position as shown in the 3-part image above. This is done by using the `.merge()` function as shown.

2.19 Exercise 19

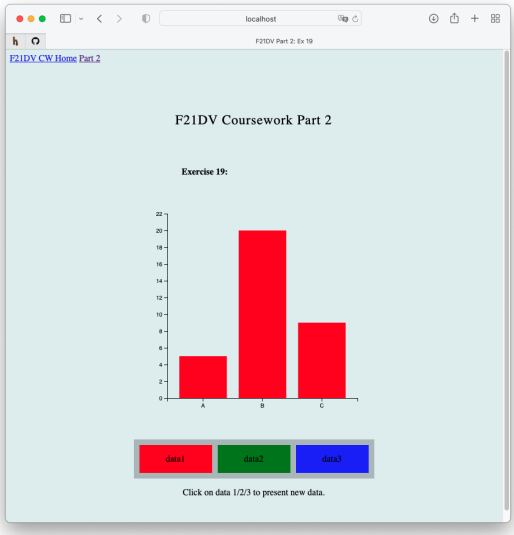

Figure 2.17: Exercise 19

Figure 2.19 shows the same bar chart as the one on exercise 18. The only difference is that now each data has its own colour. Transitions are still the same, just that now an extra 'color' attribute is being added into the transitions.

2.20 Exercise 20

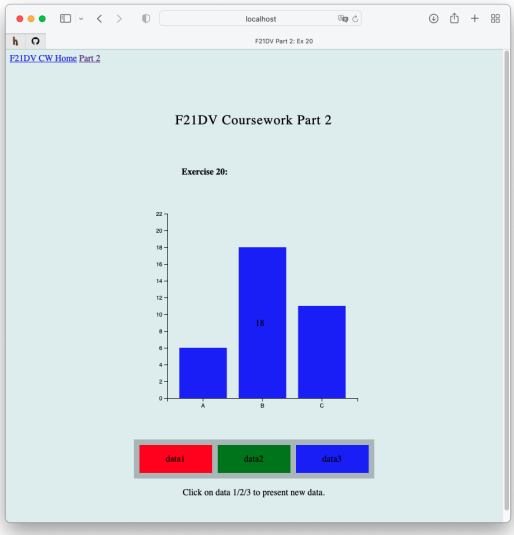

Figure 2.18: Exercise 20

Figure 2.20 shows the same bar chart, just that I have now added a mouse over function to show the actual value of the bars upon hover.

2.21 Exercise 21

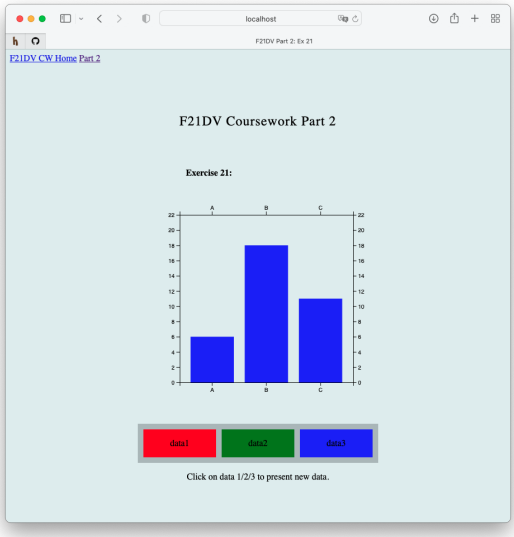

Figure 2.19: Exercise 21

Figure 2.21 shows the same bar chart as before, except that there is now an extra axis on top and on the right of the svg.

2.22 Exercise 22

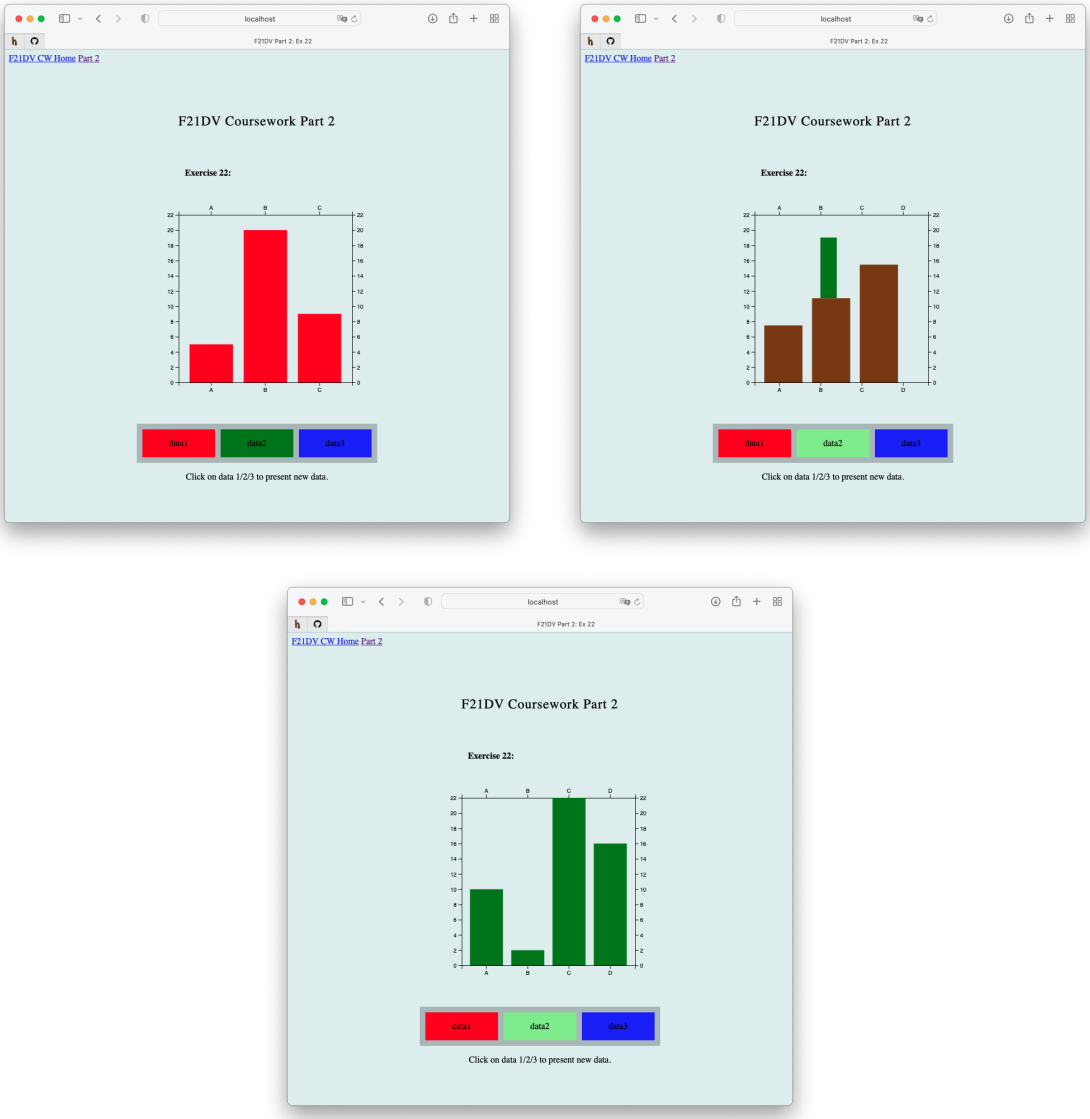

Figure 2.20: Exercise 22

Figure 2.22 shows the same bar chart as before, but now I have modified data 2 to contain an extra data entry. The transition here makes used og the `.merge()` function and would help add the extra bar when needed, and would remove it when not in need, all these with annimation.

2.23 Exercise 23

Figure 2.21: Exercise 23

Figure 2.23 shows a line chart for data 1 on load. Upon clicking the data 2 button, the lines would transition using its datapoints to data 2. The axes would also transition, adding or removing values.

2.24 Exercise 24-26

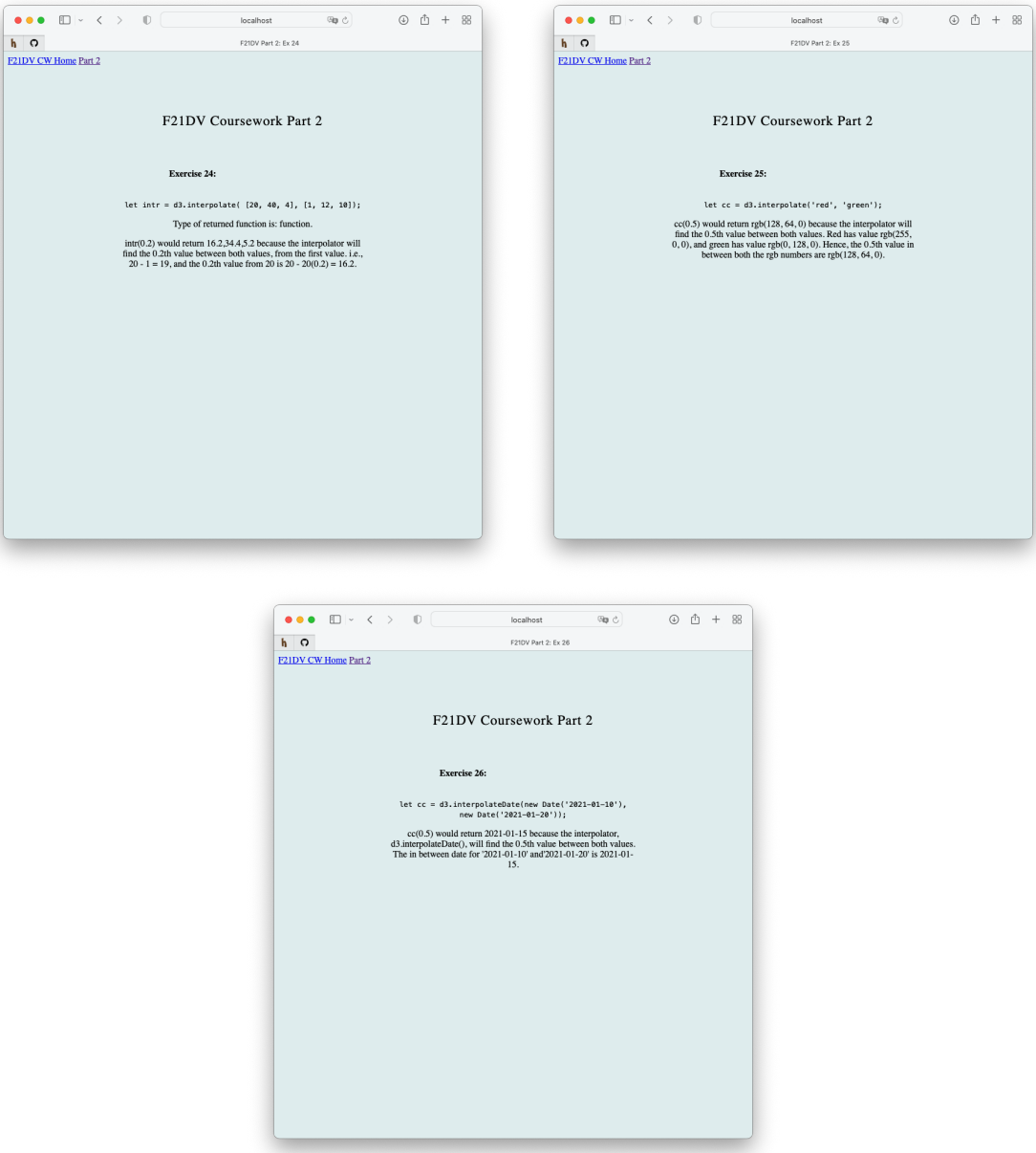

Figure 2.22: Exercise 24 to 26

Figure 2.24 shows results of interpolation within `d3`. Interpolation would interpolate between any interpolatable values to find the percentage point between values. Each example and its results are shown in the figure 2.24.

2.25 Exercise 27

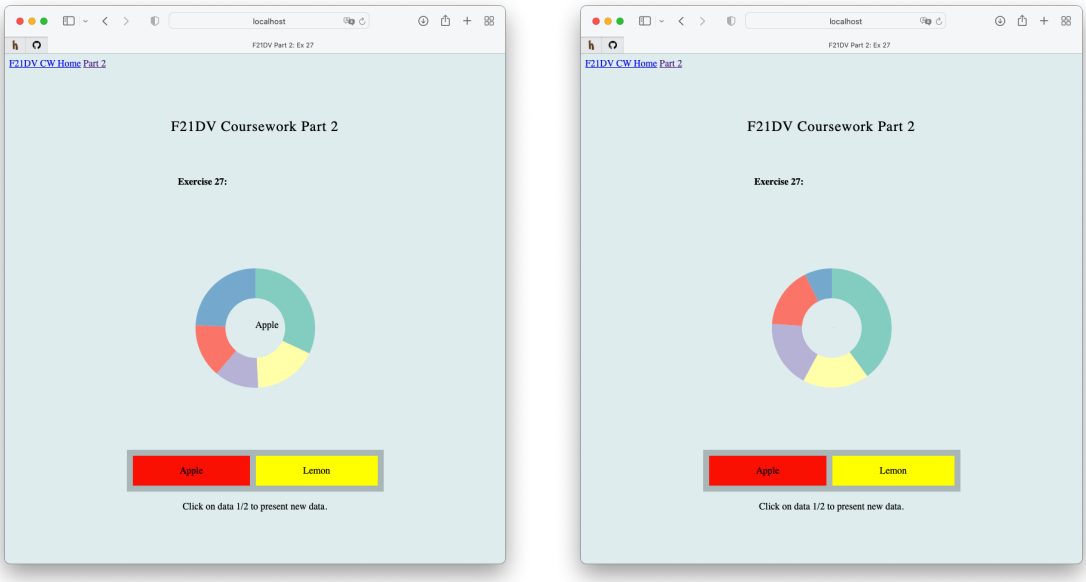

Figure 2.23: Exercise 27

Figure 2.25 shows a hollow pie chart with data about Apple. Upon clicking the Lemon button, the pie chart would transition and show data on Lemon instead. This is done semalessly once again using the `.merge()` function on the pie chart `line` object.

2.26 Exercise 28

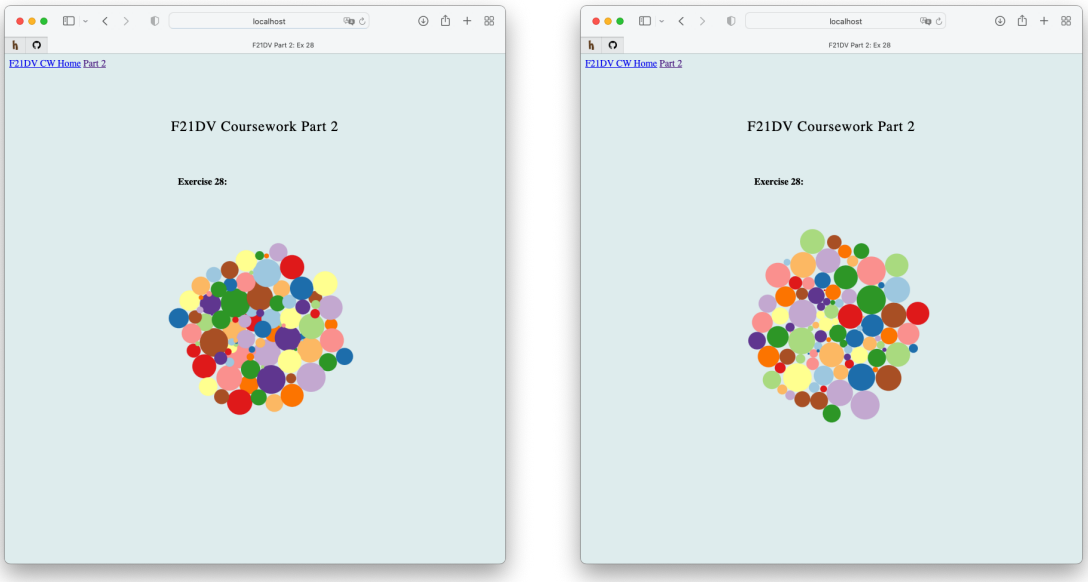

Figure 2.24: Exercise 28

Figure 2.26 shows two randomised circles on an svg using d3 force layout. This will be different each time the page is bring loaded. This will be the same for all exercises 28 through 32. This is done by using the d3 `.forceSimulation()` to created node values with radius and many other informnation (force simulation overwrites the nodes data). Then using these information, polulate the svg with circles.

2.27 Exercise 29

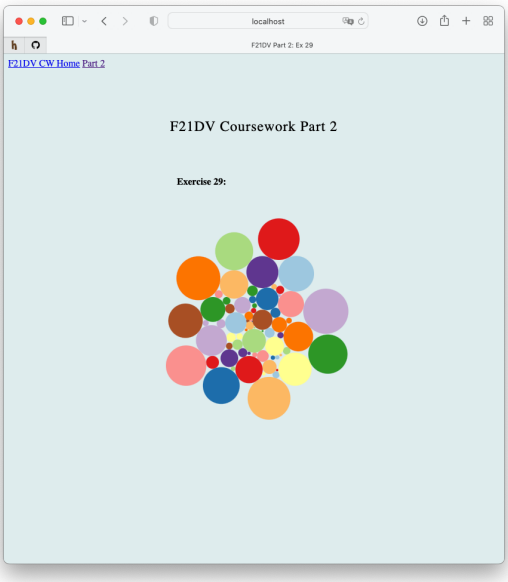

Figure 2.25: Exercise 29

Figure 2.27 shows identical for simulation from the previous exercise. Difference is this is now based off radius values from a `.csv()` file, containing values $1, 1.2, 1.4, ...$ Hence each time the page would show circles with the same position and size upon load each time.

2.28 Exercise 30

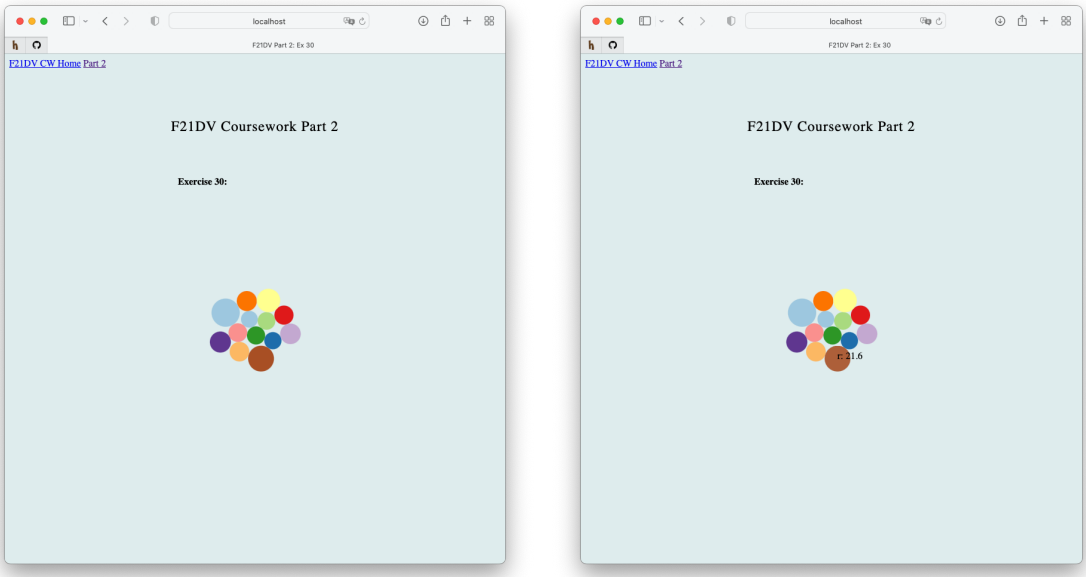

Figure 2.26: Exercise 30

Figure 2.28 shows similar force sumulation circles as before. This time, I have reduced the number of circles to improve simulation performance, and added a text box upon hover of each cirlce, displaying the circle data — the circle radius. This is shwon in the figure above, before mouse over and after mouse over.

2.29 Exercise 31

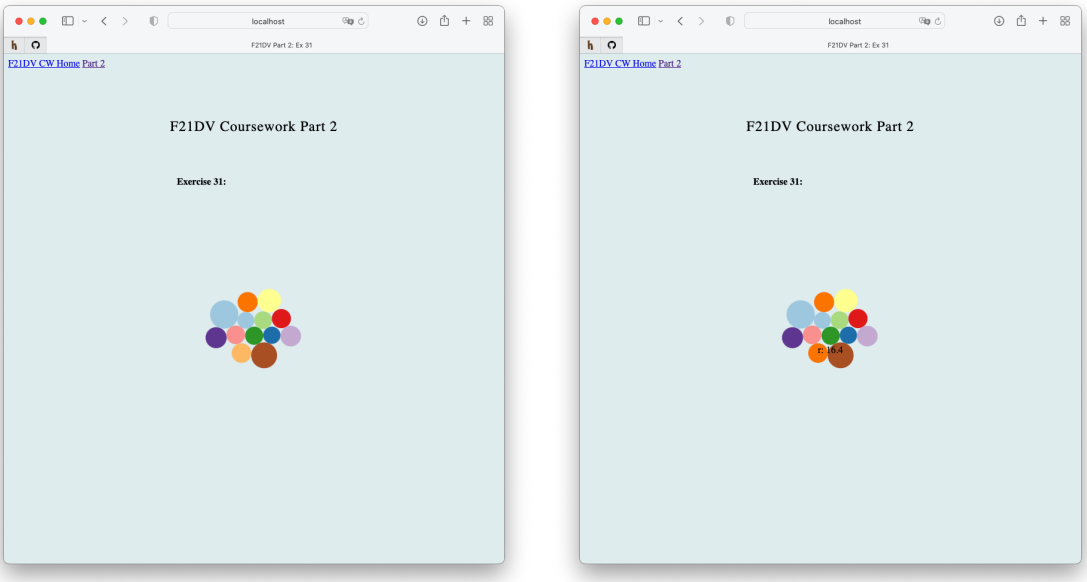

Figure 2.27: Exercise 31

Figure 2.29 shows same force sumulation circles as before. This time, there is just an added colour change to mouse over and out events.

2.30 Exercise 32

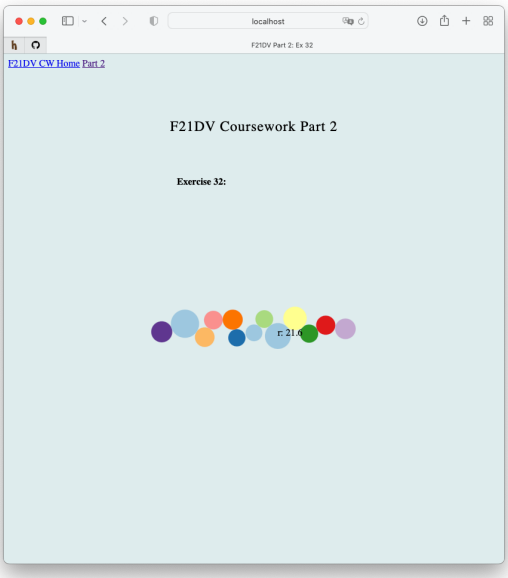

Figure 2.28: Exercise 32

Figure 2.30 shows same force sumulation circles as before. This time, I have expeeimented with different forces types. Mainly, this shows the `.force('y', ...)` type.