F21DV - Lab 2 Report

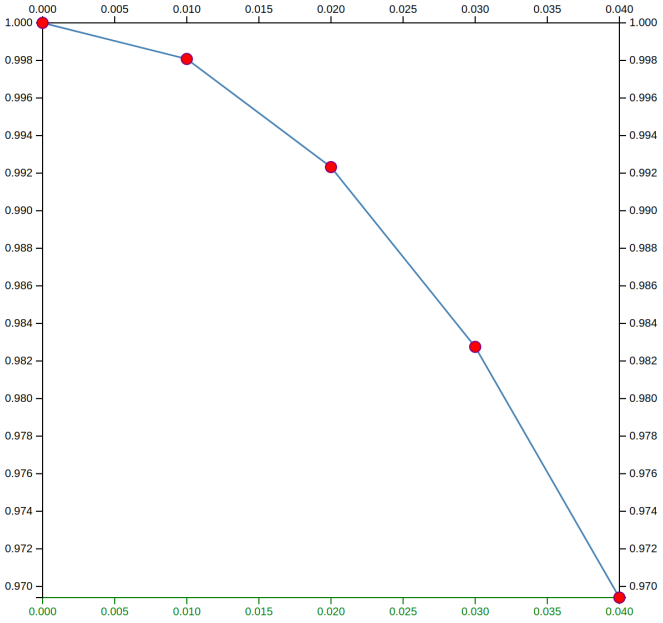
Adam Malek (H00272745)

Presented on: 25.02.2022
Presented to: Benjamin Kenwright

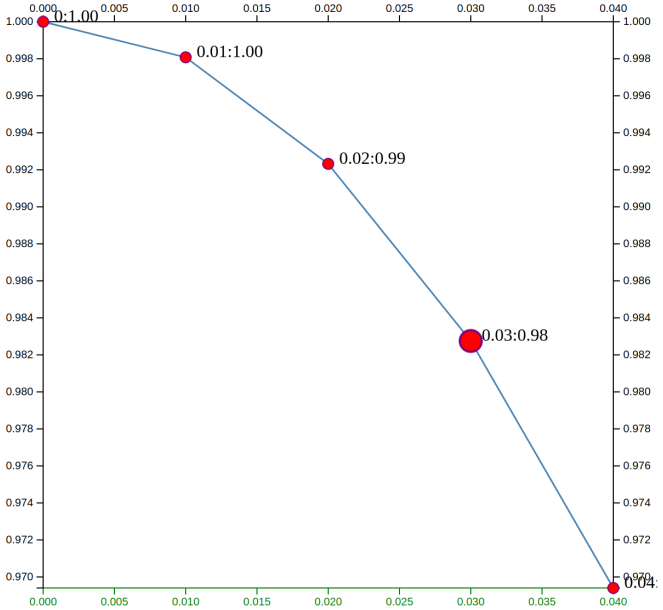
Exercise 1

As required by the exercise, when the mouse hovers over a point. Additionally, above each point (not just the selected one) a legend appears with the value. The legend is also animated, so that it appears and disappears smoothly by varying the opacity.

Without mouse hover:



With mouse hover:



Exercise 2

In this exercise, additional text appears when hovering over “Hover over me!” text. The additional text appears and disappears smoothly by using varying opacity. The duration of the animation (speed of appearance/disappearance) can be controlled using a custom CSS variable –fade-in-duration.

Without mouse hover:

Hover over me!

With mouse hover:

Hover over me!

I will show on hover

Exercise 3

In this exercise, when a mouse hovers over a green rectangle, its size will increase, the colour will change to orange and a dotted border will appear. When the mouse exits the rectangle, the border will disappear, the rectangle will go back to its original size and it will become blue. (Background colour change due to some issue with the Drawing program)

Original state:

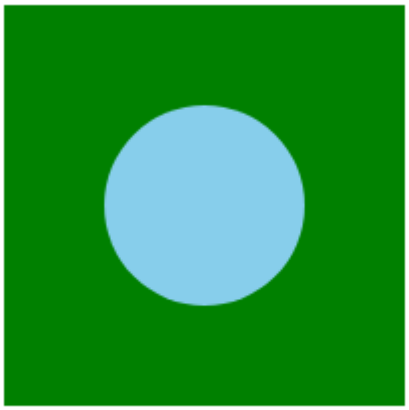
With mouse hover:

After mouse leave:

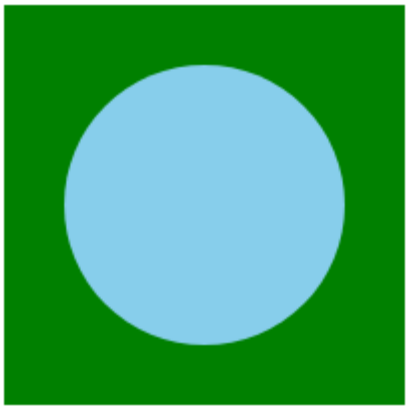
Exercise 4

As per the task description, when a user hovers over the green rectangle, its size will increase temporarily until the mouse cursor leaves the svg.

Original state:



With mouse hover:

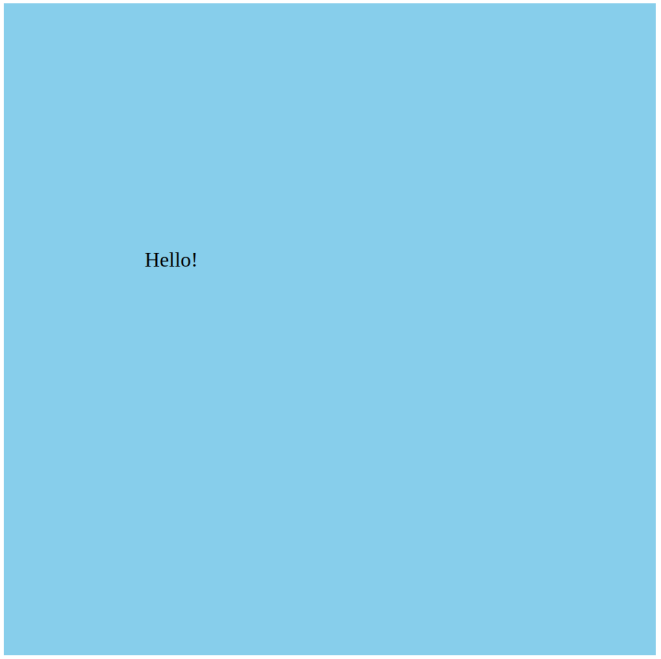


Exercise 5

For exercise 5, to get the text to follow the cursor, ‘mouseenter’ event is used which simply adds the text to the SVG, sets its text and starting position (current position of the cursor). Then, using ‘mousemove’ event the position of the text is updated. A minimal margin is applied to the y-coordinate of the text, so it does not appear exactly where the cursor is but slightly above for better readability. Finally, the ‘mouseout’ event is used for deleting the text.

With cursor outside the SVG:

With cursor inside the SVG:



Exercise 6

As per the exercise description, the rectangle changes the colour from blue to red after 1 second after the page is loaded. Then, after 2 seconds, the colour is changed again to green.

(Animation cannot be shown on screenshots, as the blue->red transition happens almost immediately after page load, due to how browsers read and execute the JS code)

Exercise 7

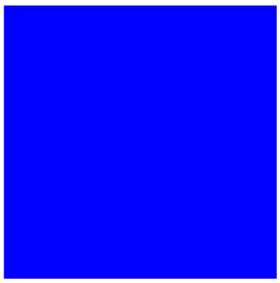
As per the exercise description, this exercise expands on the previous one. The second transition also increases the size of the square by the factor of 2.

(Animation cannot be shown on screenshots, as the blue->red transition happens almost immediately after page load, due to how browsers read and execute the JS code)

Exercise 8

Here, the rectangle will change its appearance (colour and size) when the cursor hovers over it. Once it leaves, it returns to its original state. A function was extracted for setting the original (default) properties to avoid code duplication.

Without hover:



With hover:

Exercise 9

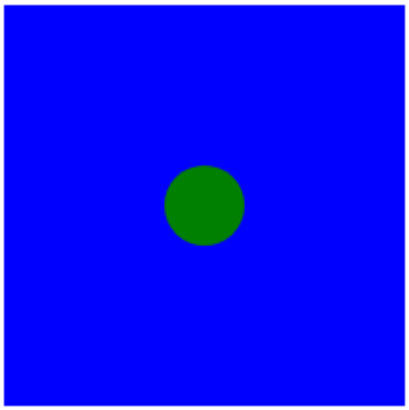
This exercise demonstrates different easing methods of the D3 library. As the only difference between divs is the animation method, a function was extracted which constructs the divs and accepts two parameters - easing method and div colour.

(Animation cannot be shown on screenshots as it happens immediately after the page is loaded).

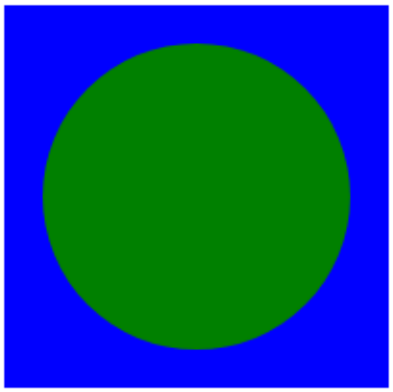
Exercise 10

As per the exercise instructions, in this task the circle grows when user hovers over it and when the mouse leaves the svg then the circle returns to its original size. The resizing is done smoothly through the D3 easing animations.

Without mouse hover:



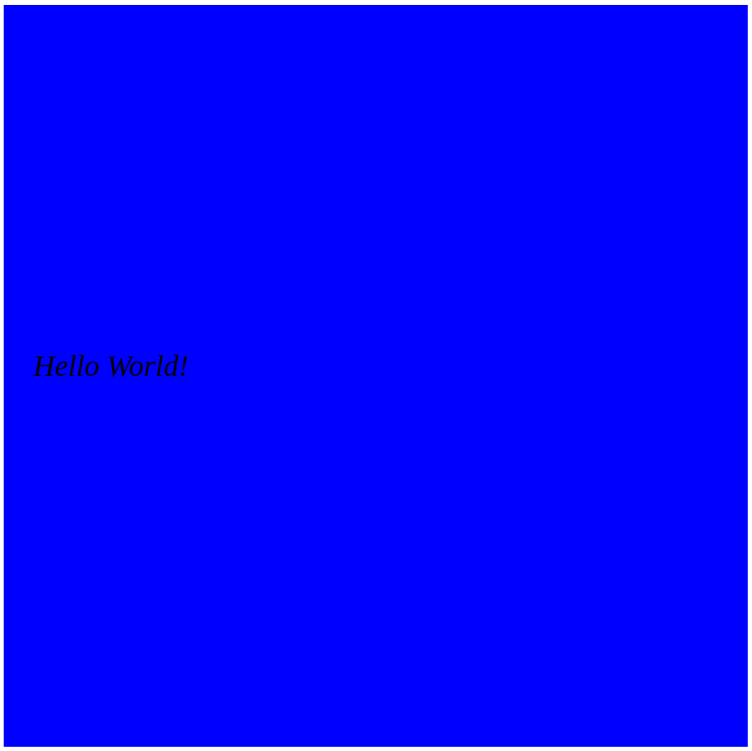
With mouse hover:



Exercise 11

In this exercise, a “Hello World!” text grows and changes the colour when the user hovers over it. Once the cursor leaves the text, it returns to its original state.

Without mouse hover:



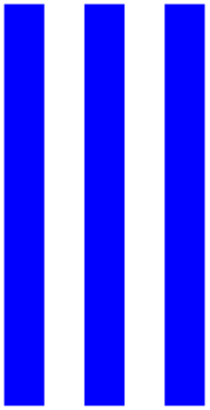
With mouse hovering over text:

Exercise 12

Here, there are 3 bars where each bar grows after the previous one reaches its full size.

Animation in progress:

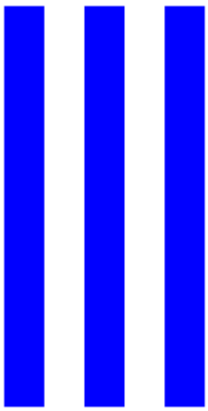
Animation complete:



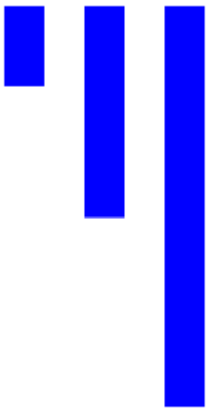
Exercise 13

The exercise is similar to exercise 12, but here, the bars go back to the previous state once they all reach their full size. This is done through chaining the transitions, but additionally, the transitions use delays so that only after the last bar grows to its full state, the reverse starts.

Animation halfway complete:



Animation complete:



Exercise 14

The exercise is more or less the same as exercise 13, but here, the colour of the bar depends on its height. A bar is blue when it's small, red when it reaches its full size.

Animation in progress:

Exercise 15

This exercise adds mouse events to the code snippet presented in the exercise description.

Mouse hovering over 2013 bar:

Exercise 16

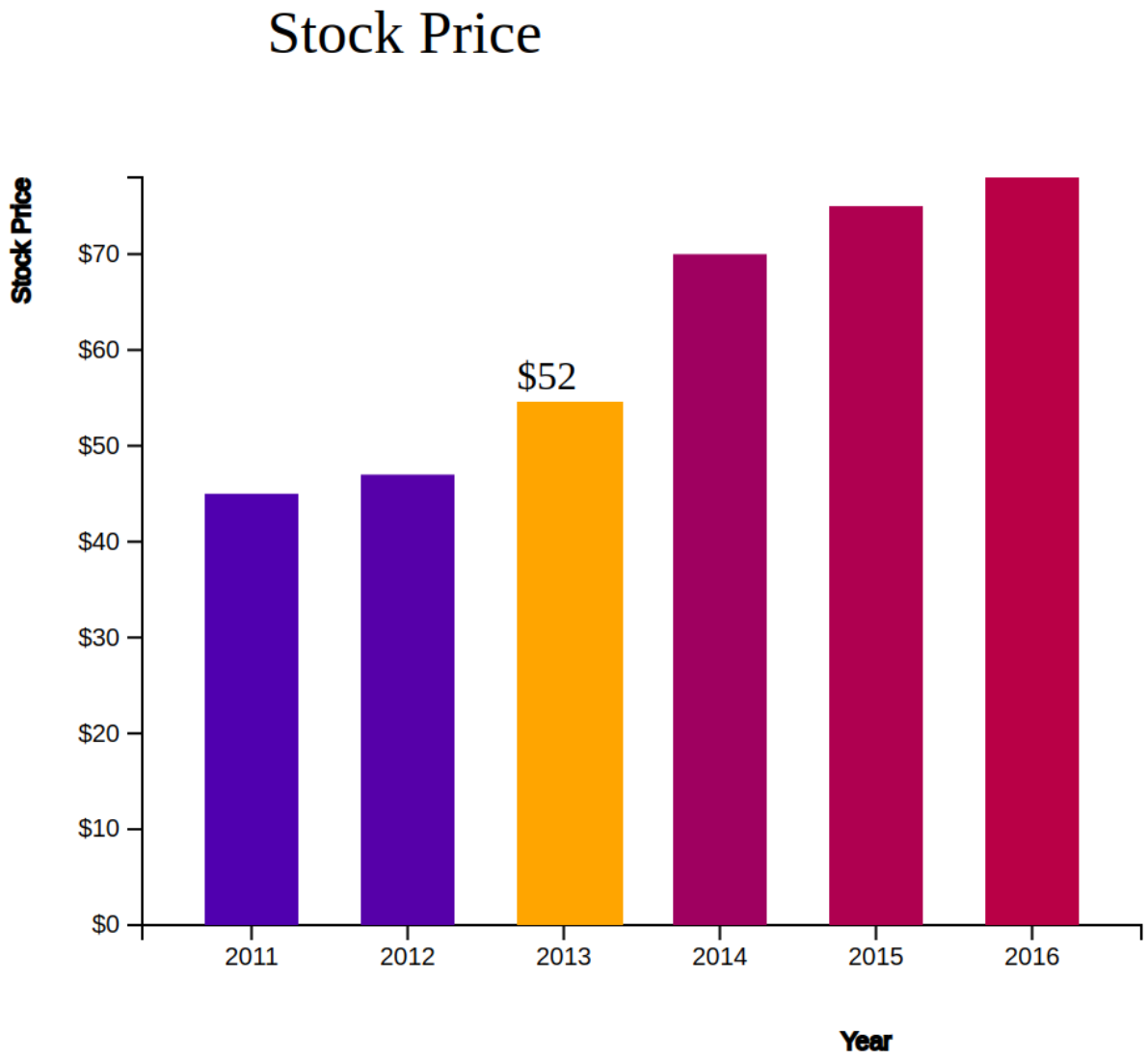
This exercise makes the value of the selected bar appear right on top of the bar, rather than top left corner. This is done by fixing seemingly intentional bug in the code where the wrong variable is used to establish the properties of the bar.

Mouse hovering over 2013 bar:

Exercise 17

The exercise is based on the previous exercise, except this time, the colour of each bar depends on its value. This, however, does not affect the selection - once the bar is hovered over, it still becomes orange.

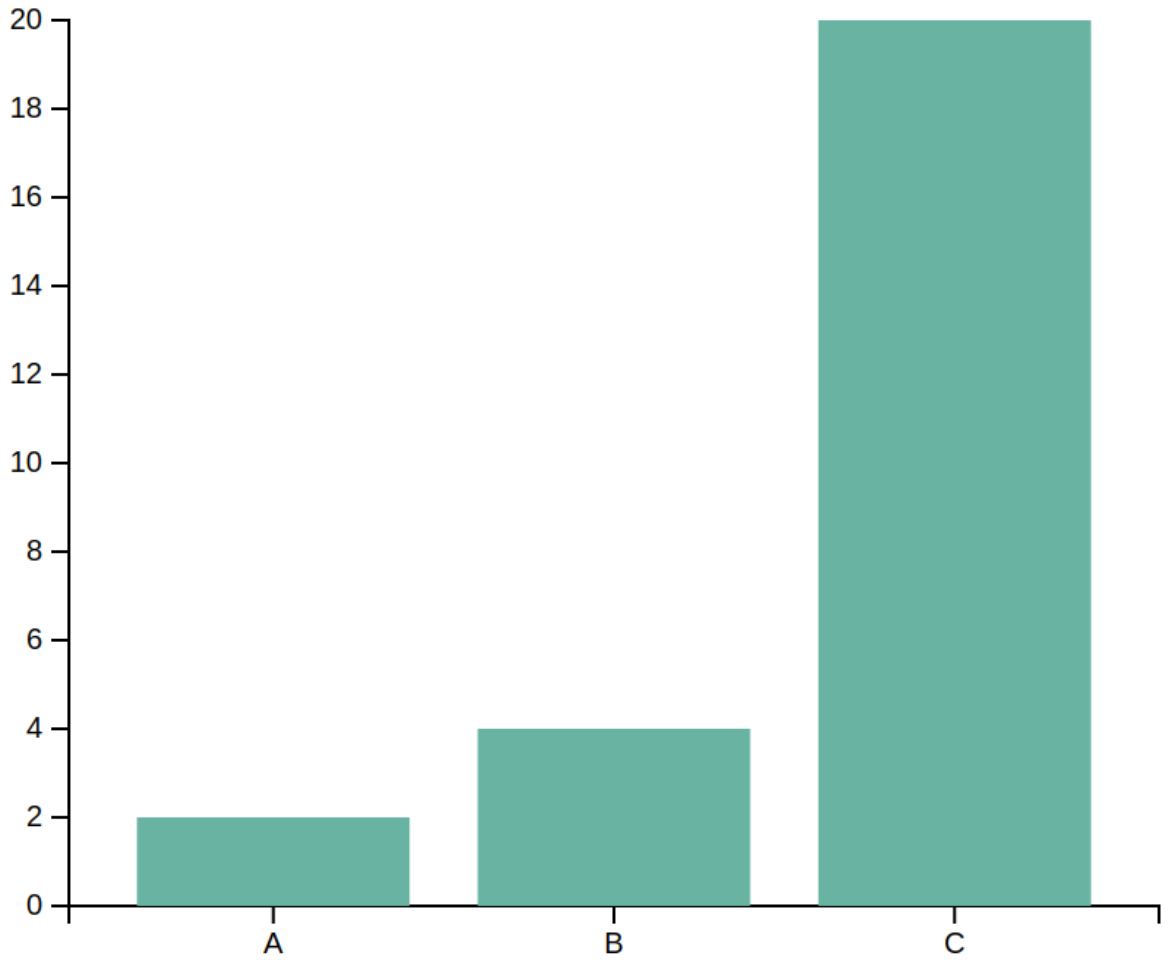
Mouse hovering over 2013 bar:



Exercise 18

This exercise adds additional dataset to the code snippet presented in the description of the exercise.

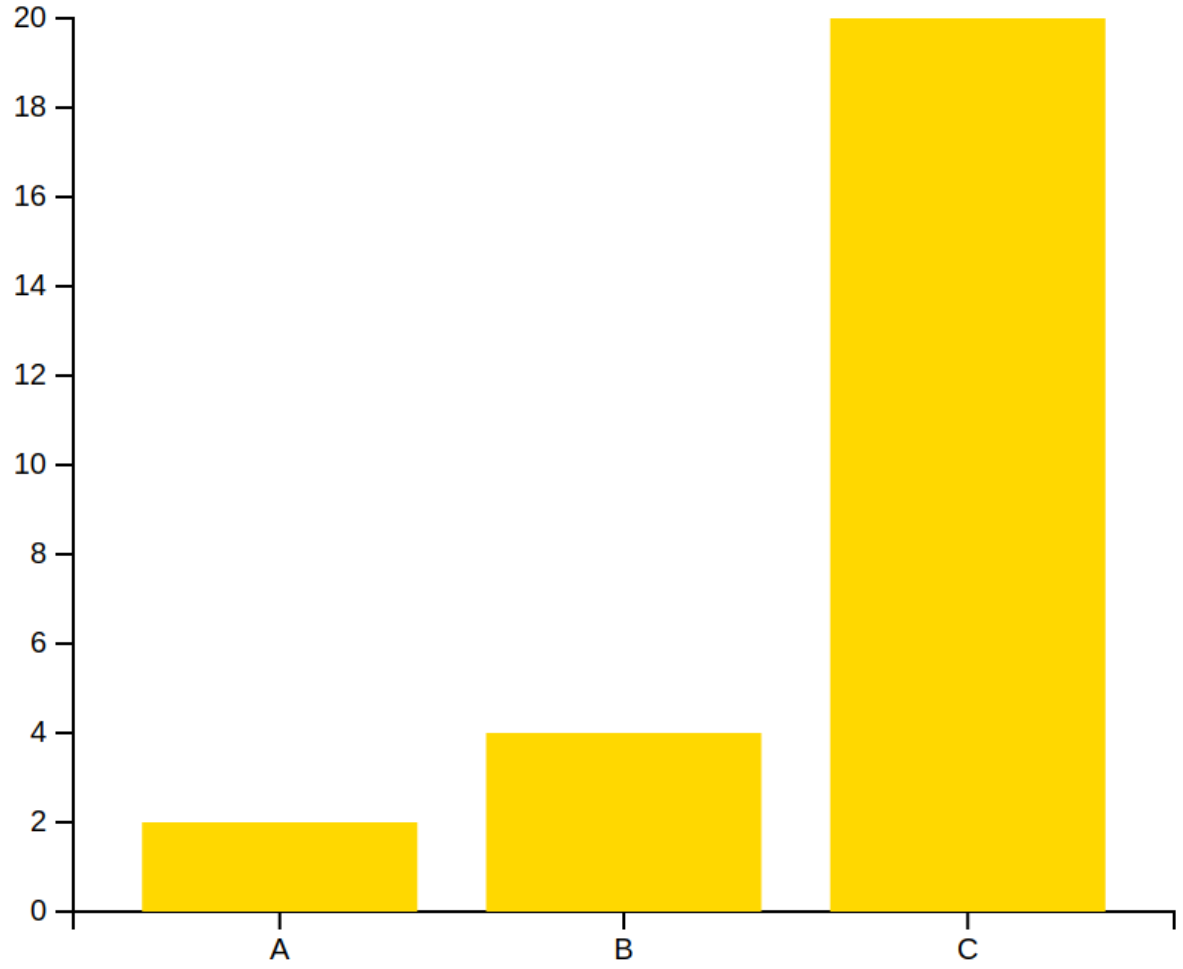
Bar graph of the new dataset:



Exercise 19

For exercise 19, each dataset gets a different colour. The update method has been updated, so that it not only accepts the data, but also the colour of the bars.

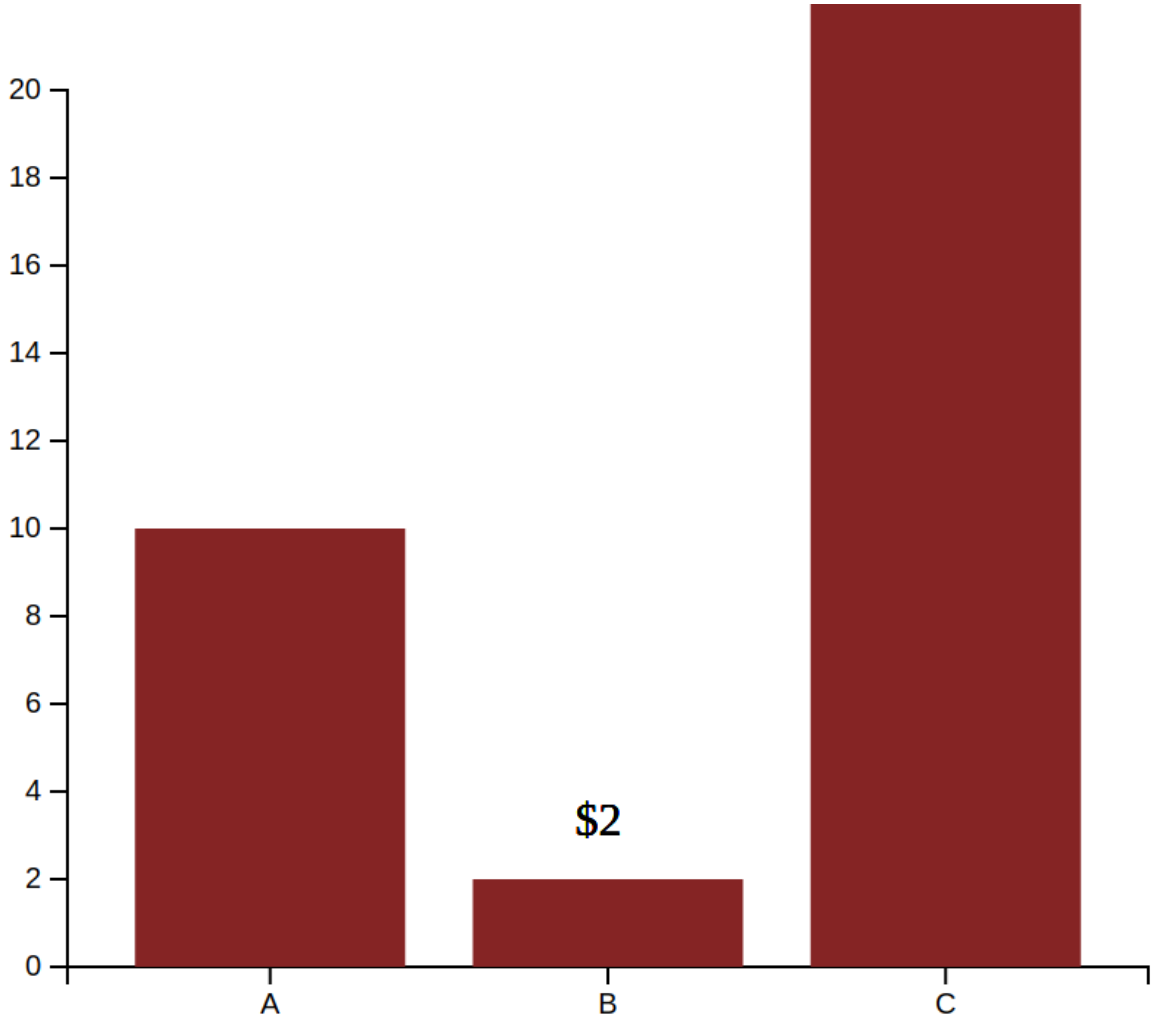
Bar graph of the dataset added in exercise 18, here in a different colour.



Exercise 20

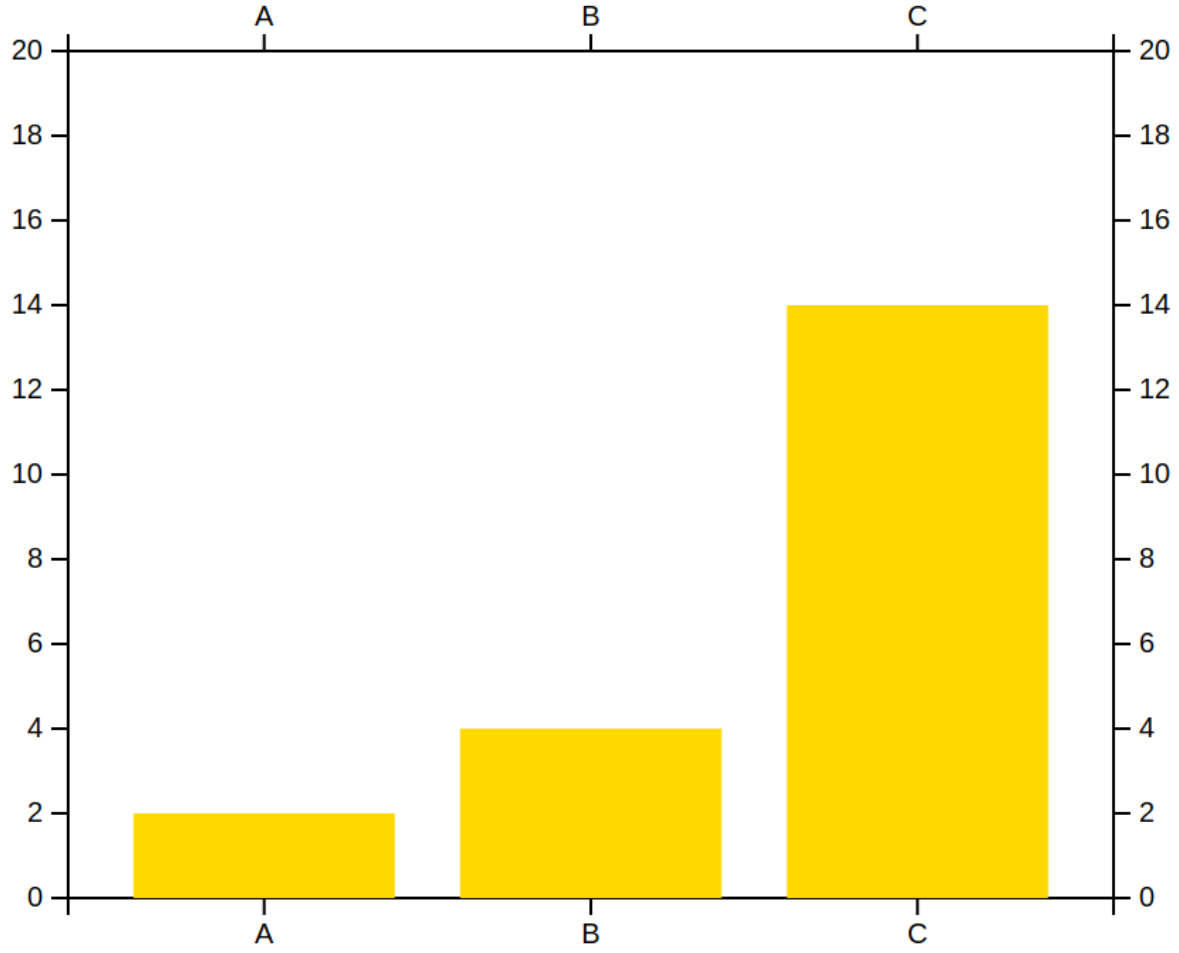
Here, when the user hovers over a bar, it’s value, as text, appears right on top of it.

Mouse hovering over B bar.



Exercise 21

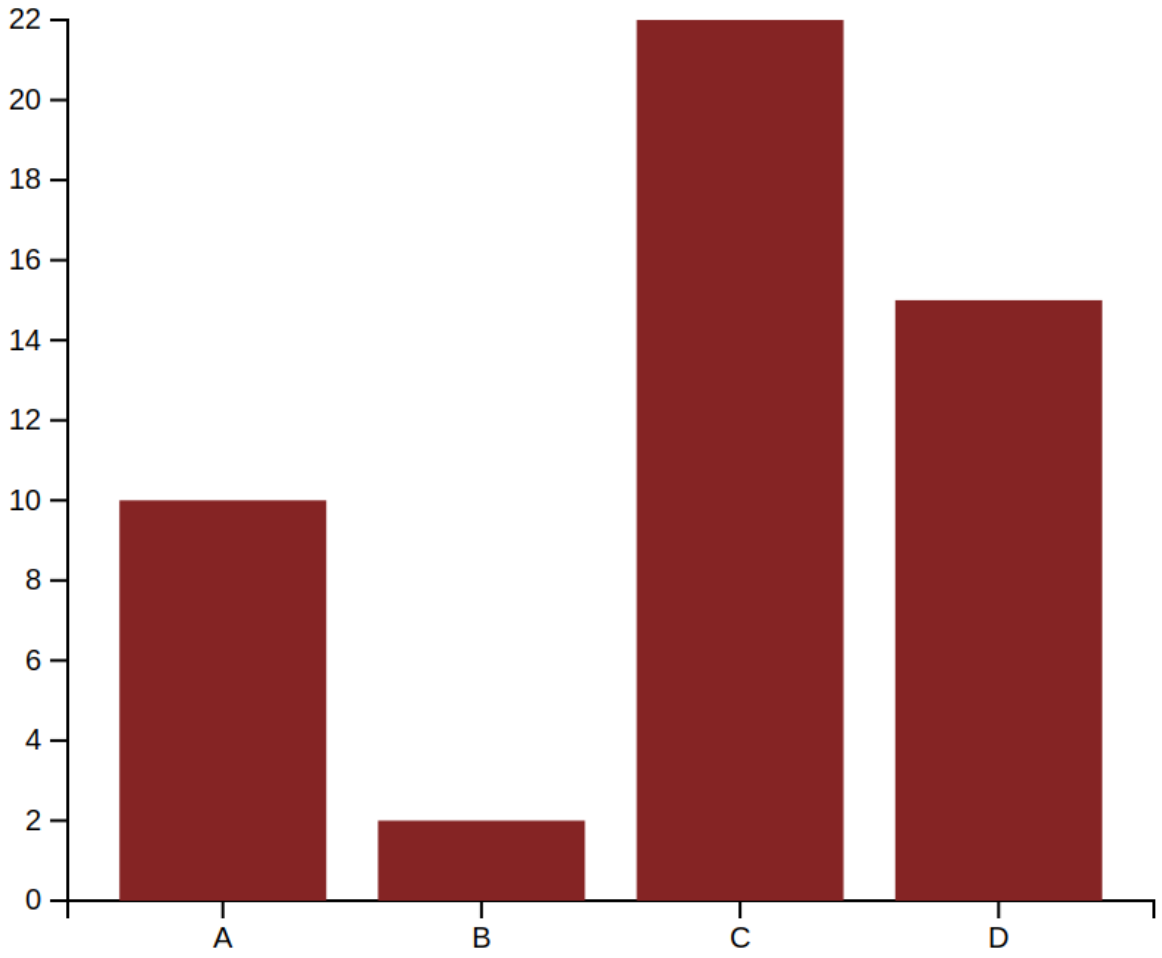
Top and right axes were added.



Exercise 22

For this exercise, each dataset has a different number of values. First dataset has 3, second one has 4, and the last one has 2. Because of this, the x and y axises need to be updated every time the user switches between the datasets. Luckily, smooth transitions can also be applied here and this is what has been done.

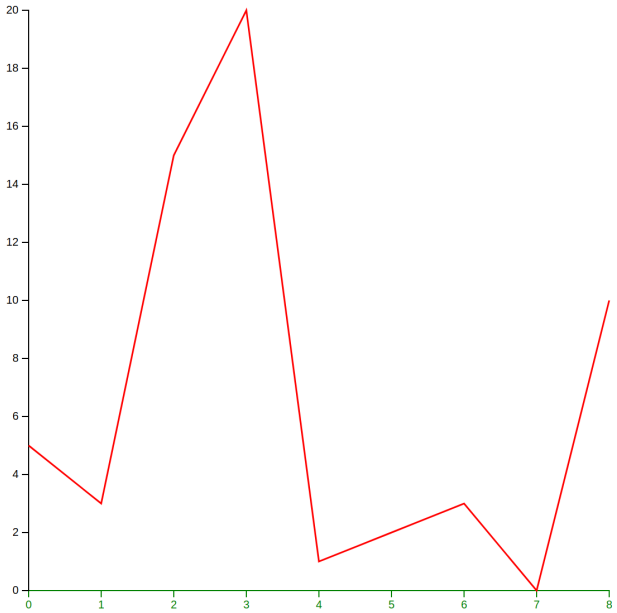
A dataset with 4 bars, whereas the first one can be seen in previous exercises. Here, the scales were scaled appropriately to fit the new data:



Exercise 23

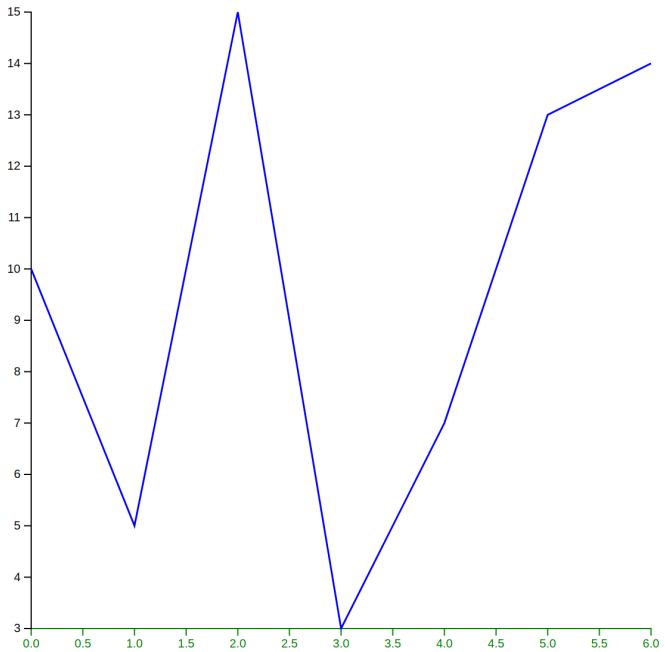
The line graph allows users to switch between the two different pieces of data using buttons. The transition is smooth as both axises get animated, through the same technique as in the previous exercise, and the lines are changed through a smooth opacity change.

Variable 1 graph:



Variable 1 | Variable 2

Variable 2 graph:

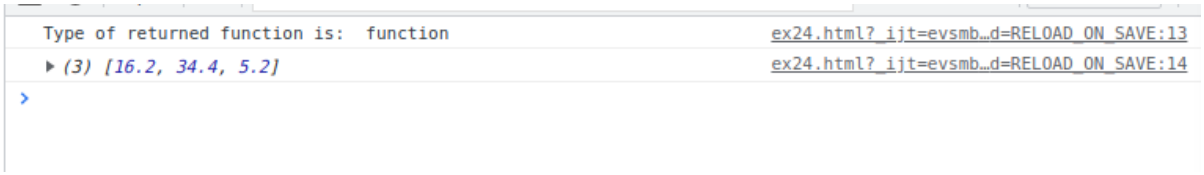


Variable 1 | Variable 2

Exercise 24

The formula for interpolation is
$x(t) = a \times (1 - t) + b \times t$
and this formula is applied to every element of the array. For example, for the first element of the result array we expect value to be:
$x(0.2) = 20 \times (1 - 0.2) + 1 \times 0.2 = 16.2$
and indeed this is the value that we get.

Output of the console:

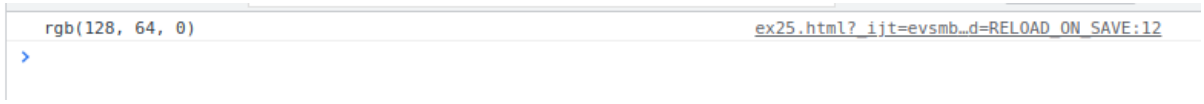


Exercise 25

We can convert the two colours to their RGB values:
Red - (255, 0, 0)
Green - (0, 128, 0)

Therefore, by using either using the formula presented in the exercise 24 or simply dividing the difference by two, the halfway value between the two colours should be (128, 64, 0) [note that 127.5 has been rounded up, not just truncated] which appears to be a dark orange colour.

Output of the console:



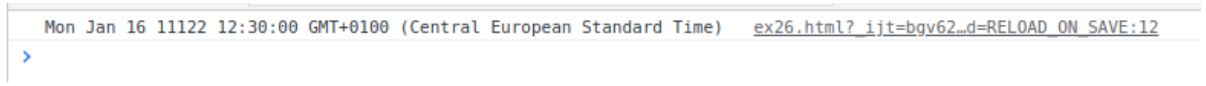
Exercise 26

The interpolation of dates can be done using D3's interpolateDate function. Internally, the function uses the fact that the Date class in Javascript has overload for the standard operators (*, -, +), when they are used, each date is converted to milliseconds (Unix epoch). The answer is a new Date object, where its value, the date and time, is set using setTime function which accepts the milliseconds. The code is shown below.

```
<script>
 const cc = d3.interpolateDate(new Date('20222-01-01'), new
Date('2022-01-31'))
 console.log(cc(0.5));
</script>
```

The advantage of using a library function, rather than writing a solution ourselves, is that we can safely assume that the output will be corrected and that the function has been tested by multiple unit-tests. However, this is not to say that the correct output is 100% guaranteed, as with any projects, bugs can occur and some edge cases may not have been noticed, and hence handled, by the developers.

Output of the console:

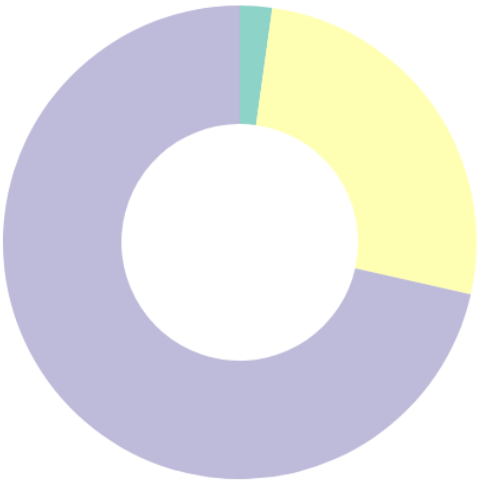


Exercise 27

In this example of a pie chart, the data is displayed and changed smoothly by animating each slice. It is worth noting, that removing a slice is also handled, and the animation that is used to achieve that is the inverse of the animation used to add one.

Pie chart with 5 pieces of data:

Pie chart with 3 pieces of data:



Exercise 28

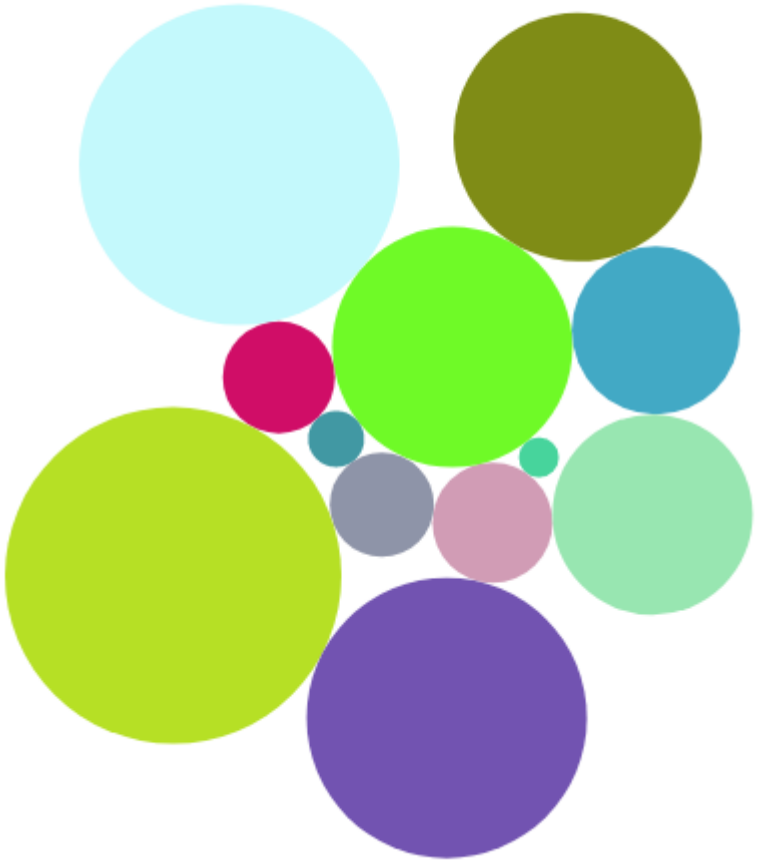
For this exercise, each sphere gets a random colour from a index-to-colour map. This map plays an important role, as without it, every time ticked() function would be called, each sphere would change its colour.

Output of the exercise:

Exercise 29

This example modifies the previous one by having a defined array of nodes instead of generating one randomly.

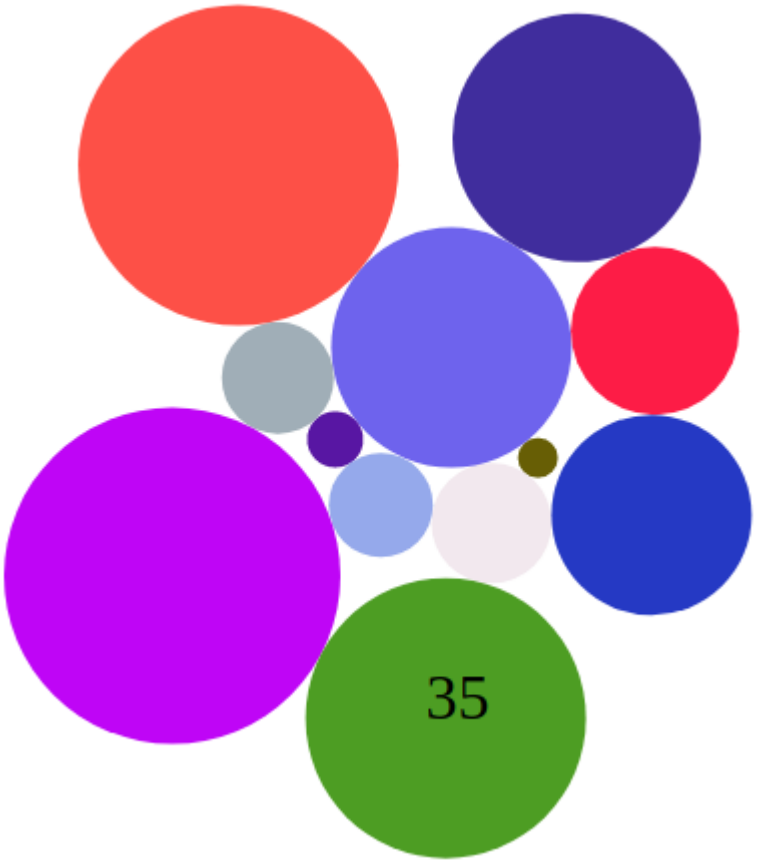
Output of the exercise:



Exercise 30

With this example, when a user hovers over a sphere, its value appears right above the cursor.

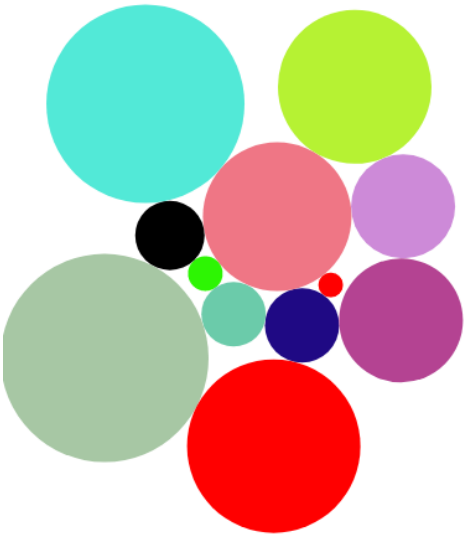
Output of the exercise:



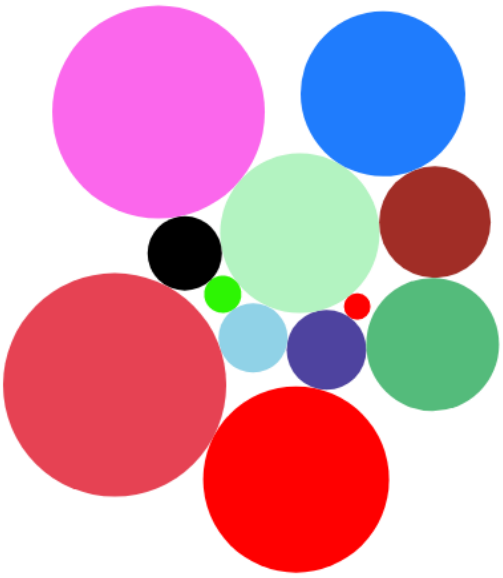
Exercise 31

Here, the mouseover events triggers logic which changes the colour of the sphere that has the cursor on it.

Original state:



After hovering over some of the spheres:



Exercise 32

(Misinterpretation of the task). This exercise adds another force called link, which links the first and the last nodes together. To make that visible to the user, these two spheres are marked in red and do not change the colour when there is a cursor on top of them.

Output of the exercise: