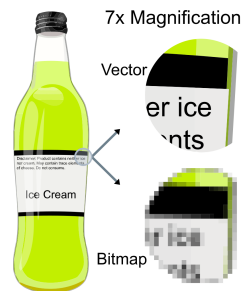


## STATS 401 Lab 7 Tutorial: D3.js Introduction

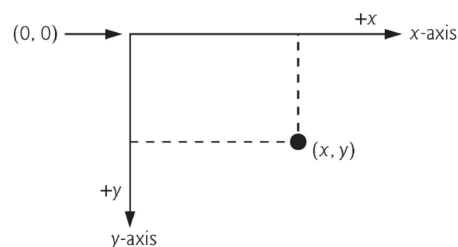
In this lab, we will introduce D3.js (data-driven document). D3 is a JavaScript library that helps you manipulate documents based on data. In lab 3, you learned some basic knowledge about HTML and CSS. With D3, you can better bring data to life using HTML, CSS, and SVG (scalable vector graphics).









**Figure 1-1.** Why we need SVG [\[image source\]](#)

D3 allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. For example, you can use D3 to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction.

When creating images using SVG, we follow the coordinate system shown in figure 1-2. Table 1 introduces some commonly used SVG shapes.



**Figure 1-2.** Coordinates for computer graphics [\[image source\]](#)

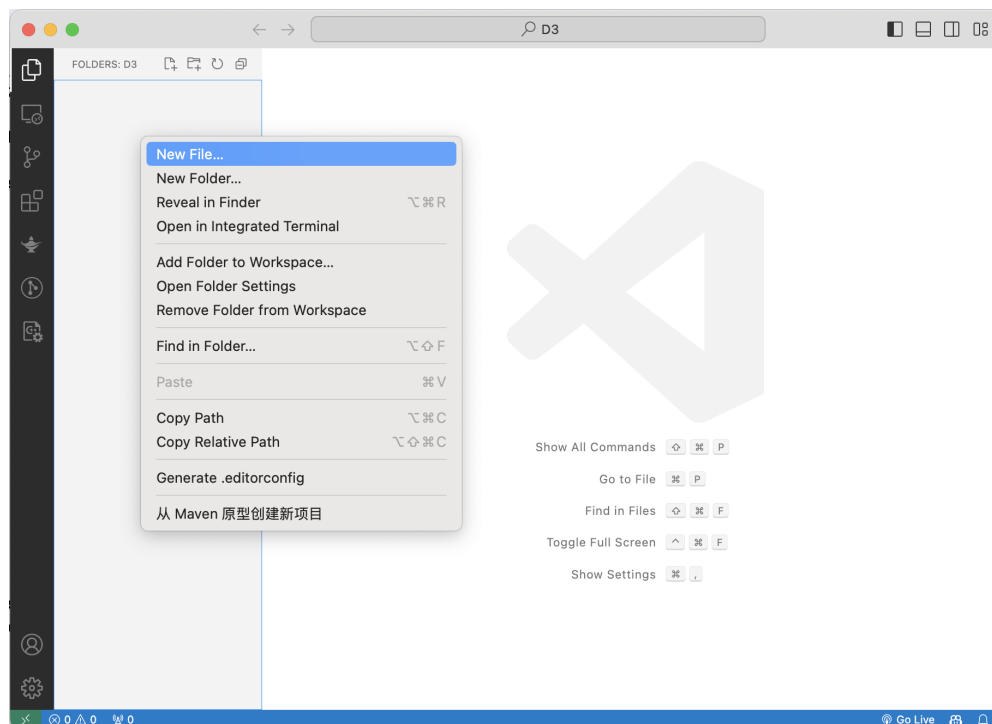
	Shape	Function	Mandatory Attributes	Optional Attributes
	rect	.attr()	x, y, width, height	rx, ry, class, transform
	circle	.attr()	cx, cy, r	class, transform
	ellipse	.attr()	cx, cy, rx, ry	class, transform
	line	.attr()	x1, y1, x2, y2	class, transform
	text	.text() .attr()	x, y	dx, dy, class, transform, text-anchor
	path	.attr()	d	class, transform, pathLength

**Table 1.** Commonly used SVG shapes

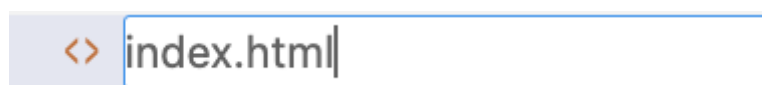
By the end of this lab, you will learn how to utilize simple SVG shapes to visualize your data and inspect the attributes of these elements in the web browser. We will use sample data to help you grasp the concept this time and introduce how to create visualizations with imported .csv/.json files in later labs.

**Task 0.** Set up an organized project folder with d3.v7 imported.

Create a folder named “D3” and open it in the Atom editor. Right-click on the blank space and choose “New File” or “New Folder” to create new files/folders. (Figure 2-1 & 2-2)

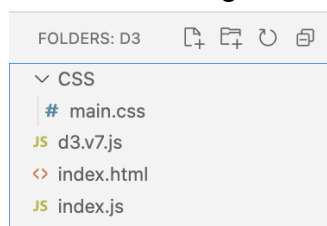


**Figure 2-1.** New files/folders



**Figure 2-2.** Create files

Create main.css, d3.v4.js, index.html, and index.js files and a CSS folder, and make them follow the hierarchy structure shown in figure 3.



**Figure 3.** An organized D3 project folder with necessary files

Go to <https://d3js.org/d3.v7.js> and copy everything there into the d3.v7.js blank file you just created. This step imports the D3 library while providing you a reference if you want to look at specific functions in detail.

Then, go to the newly created index.html file. We are going to initialize an HTML template with a magic trick: enter “!” in the first line and tap the “Tab” key on your keyboard. After this, you will see an auto-generated template as shown in figure 4.



```
JS d3.v7.js  <> index.html X
<> index.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9
10 </body>
11 </html>
```

**Figure 4.** An auto-generated template

Based on this, we also need to add two more lines to link to the .css file and introduce script type and source in the head (figure 5). I also change my webpage title to “Lab 7”.

```
<link rel="stylesheet" href="CSS/main.css">
<script type="text/javascript", src = "d3.v7.js"> </script>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="CSS/main.css">
    <script type = "text/javascript", src = "d3.v7.js"></script>
    <title>Lab 7</title>
</head>
```

**Figure 5.** A working “head”

In the “body” part, you can still add content unrelated to D3 (e.g., things we learned in lab 6). For example, I added a navigation bar and some placeholder text here.

```
<body>
  <nav>
    <li><a href="#">Home</a></li>
    <li><a href="#">About</a></li>
```

```

</nav>
<p> placeholder text placeholder text placeholder text
placeholder text</p>
<script type="text/javascript", src = "index.js"> </script>
</body>

```

As we learned in lab 6, we can define the style of the in the main.css file. The code I have in the main.css file is shown below.

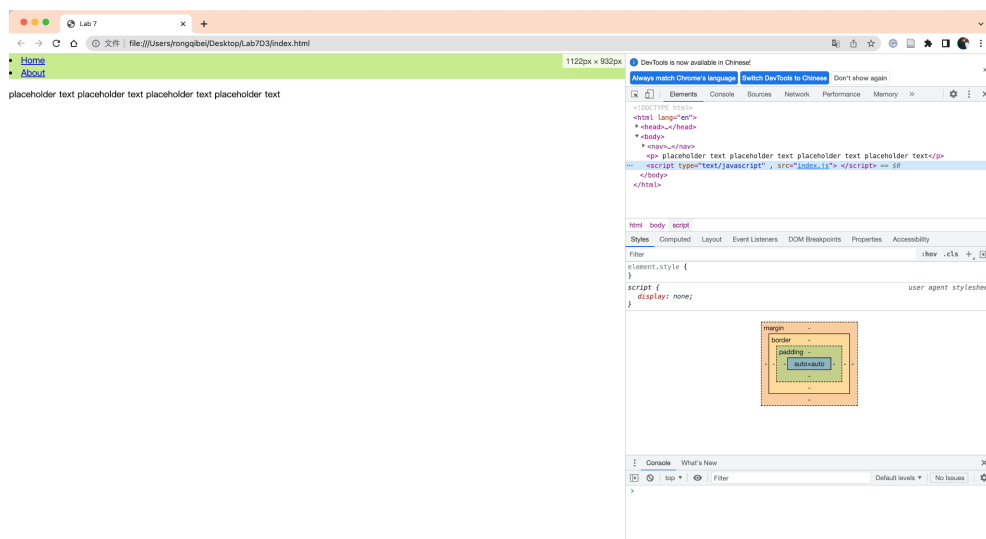
```

nav{
  background-color: #c7ea8d ;
}
html,body{
  margin: 0;
  height: 100%;
}

```

If you read carefully, you must have discovered a new line of code that haven't been introduced - "<script type='text/javascript', src = 'index.js'> </script>". This line provides a reference to the index.js file and allows what we have in that file to be script content here.

Now, run your index.html file and the expected outcome should be similar to figure 6. Most of the spaces are blank because we currently do not have anything in the index.js file.



**Figure 6. Task 0 outcome**

## **Task 1. Visualize Beijing's monthly average temperature in a bar chart**

The sample data we have for this task is the daily average maximum temperature in each month from a public [website](#). From January to December, we have the below data:

```
temperature = [3, 6, 15, 23, 29, 32, 33, 32, 27, 19, 10, 4]
```

Copy and paste the below code to your index.js file to input the sample data and another variable called svg and append the svg component to the body.

```
var temperature = [3, 6, 15, 23, 29, 32, 33, 32, 27, 19, 10, 4];
```

```
var svg = d3.select("body")
    .append("svg")
    .attr('height', '100%')
    .attr('width', '100%');
```

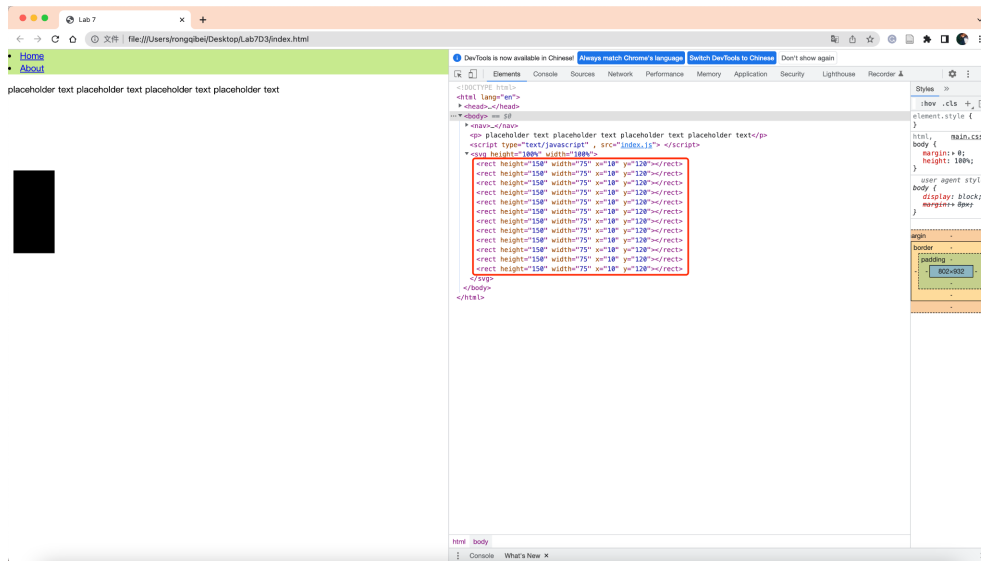
Now, we are going to create rectangles.

```
svg.selectAll('rect')
    .data(temperature)
    .enter()
    .append('rect')
    .attr('height', '150')
    .attr('width', '75')
    .attr('x', 10)
    .attr('y', 120);
```

The rectangles take our temperature list as input and have a width of 75px, height of 150px, and center at the (10, 120) coordinate. If you refresh the webpage after this step, you will only see one rectangle lying on the page (instead of 12 rectangles).

What is wrong?

If you tap F12 and open the inspector, you will see that we do have 12 rectangles. They just overlap with each other.



**Figure 7. Inspect the rectangles**

Also, as you may have already noticed, the attributes of these 12 rectangles do not have any relationship with our temperature data even if we have already input them in the code above.

To separate these 12 rectangles and let the height of the rectangles denote the corresponding monthly temperature, we can revise the above code to

```
svg.selectAll('rect')
  .data(temperature)
  .enter()
  .append('rect')
  .attr('fill', '#96d82a')
  .attr('height', function(d,i){return (d*10);})
  .attr('width', '30')
  .attr('x', function(d,i){return 40*i;})
  .attr('y', function(d,i){return (380-d*10);});
```

To apply the data in the temperature list, we introduce a function with parameters *d* and *i*. Here, parameter *i* represents the looping index for going over the temperature data list. Parameter *d* represents the *i*-th value in the list.

Think about why we have  $380-d*10$  instead of  $d*10$ ? Hint: the coordinate figure.

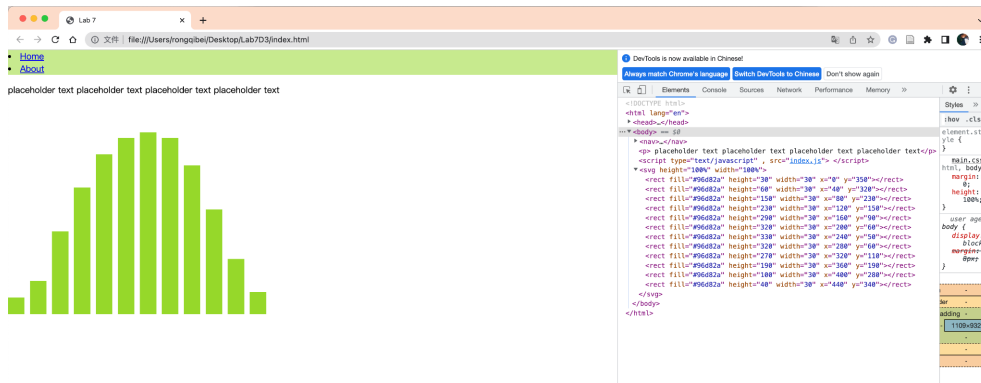


Figure 8. Task 1 outcome

## Task 2. Create two groups of circles that indicate each month's daily average maximum/minimum temperature in Shanghai

From an online data [source](#), we learn that the month's daily average maximum/minimum temperature from January to December in Shanghai are:

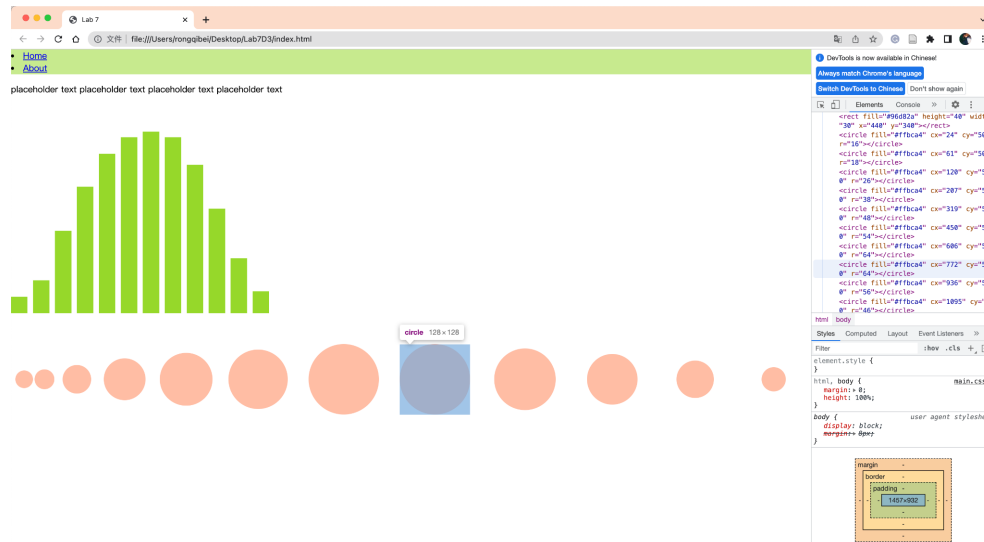
```
var max_temperature = [8, 9, 13, 19, 24, 27, 32, 32, 28, 23, 17, 11]
var min_temperature = [1, 2, 5, 11, 16, 21, 25, 25, 21, 15, 9, 3]
```

Similar to making a bar chart with rectangles, we can also plot 12 circles and relate their radiuses to the temperature. To do so, we have the following code:

```
var fixedX = 0;
svg.selectAll('circle')
  .data(max_temperature)
  .enter().append('circle')
    .attr('fill', '#ffbca4')
    .attr('cx', function(d,i){fixedX += d*3+i*10; return fixedX;})
    .attr('cy', '500')
    .attr('r', function(d,i){return d*2});
```

Calculate some of the `cx` values and check the outcome in the inspector. Do you get it right?

The expected outcome should look like the following (figure 9).



**Figure 9.** 12 circles whose radiuses represent each month's daily average maximum temperature in Shanghai

Now, let's try to add another group of circles. Can we do so by copying and pasting the chunk of code above and changing the data source? To experiment with this idea, let's add the code below (the data source, color, and `cy` value are changed).

```
var fixedX = 0;
svg.selectAll('circle')
  .data(min_temperature)
  .enter().append('circle')
  .attr('fill', '#a4e7ff')
  .attr('cx', function(d,i){fixedX += d*3+i*10; return
fixedX;})
  .attr('cy', '900')
  .attr('r', function(d,i){return d*2});
```

However, when refreshing the webpage, the circle elements appear neither on the webpage nor in the inspector. That is, what we get is still the old maximum temperature version.

Can you guess what is wrong?

This is because of the feature of our `selectAll` function. Since we have already created SVG circles on the canvas for the maximum temperature, calling the `svg.selectAll('circle')` function again will not provide any updated results.

Here is the case where we need to introduce the concept of “group”. We call the the maximum group A (“groupa”) and the minimum group B (“groupb”) with the following code.



```

var fixedX = 0;
svg.selectAll('circle.groupa')
  .data(max_temperature)
  .enter().append('circle')
    .attr('fill', '#ffbc4')
    .attr('class', 'groupa')
    .attr('cx', function(d,i){fixedX += d*3+i*10; return
fixedX;})

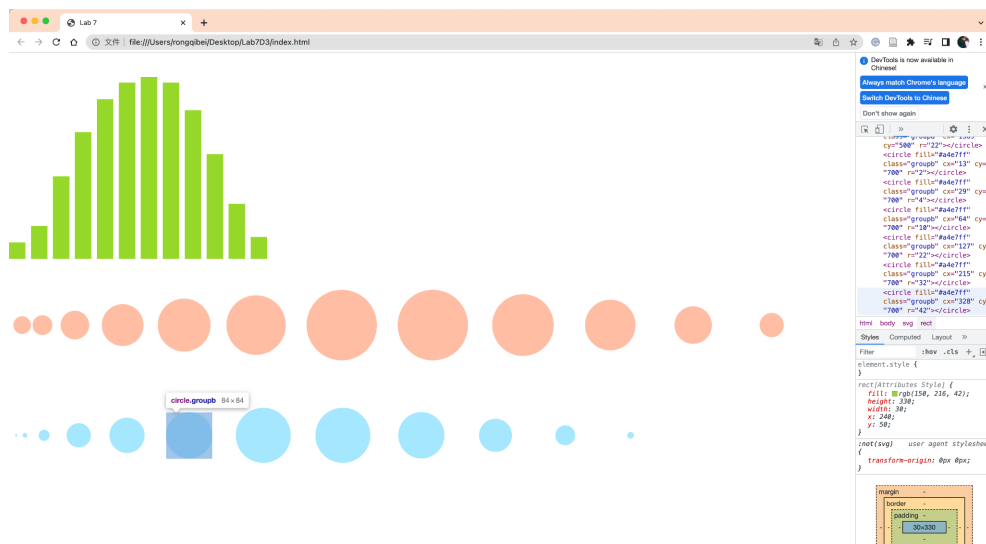
    .attr('cy', '500')
    .attr('r', function(d,i){return d*2});

var fixedX = 10;
svg.selectAll('circle.groupb')
  .data(min_temperature)
  .enter().append('circle')
    .attr('fill', '#a4e7ff')
    .attr('class', 'groupb')
    .attr('cx', function(d,i){fixedX += d*3+i*10; return
fixedX;})

    .attr('cy', '700')
    .attr('r', function(d,i){return d*2});

```

The expected outcome should look like the following (figure 10).



**Figure 10.** Task 2 outcome

**Task 3.** Use the length of the line to denote the number of DKU undergraduate students in each Class

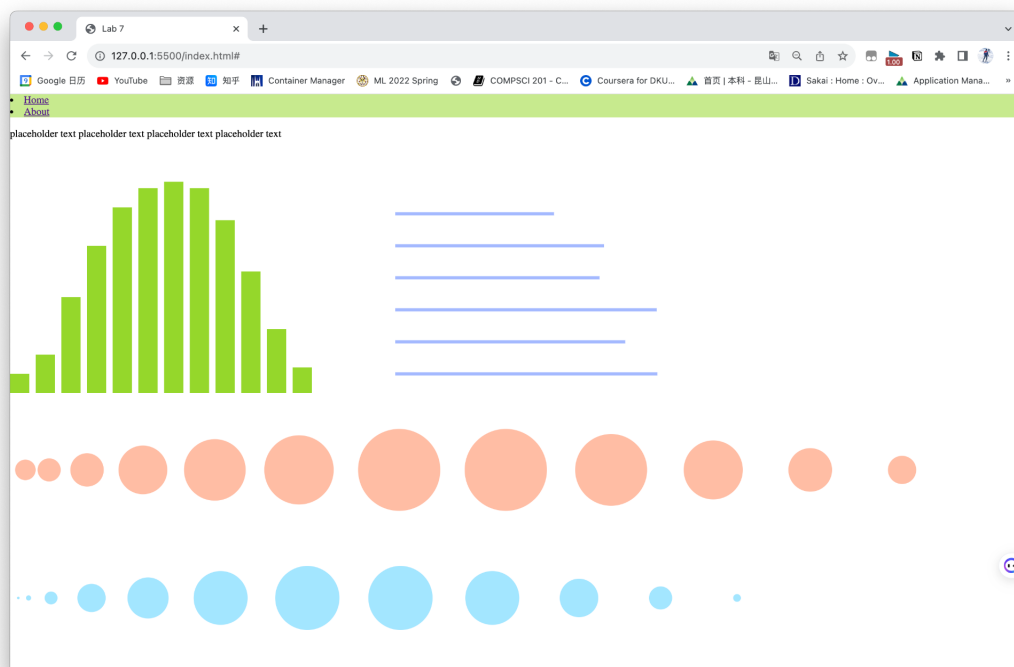
The number of DKU undergraduate students from the Class of 2022 to the Class of 2027 is shown in the below list. I googled it from various news so it might be slightly inaccurate @\_@...

```
var ug_number = [247, 325, 318, 407, 358, 408]
```

From table 1 we know that drawing lines requires the coordinates of the two endpoints, which are  $(x_1, y_1)$ , and  $(x_2, y_2)$ .

```
var fixedX = 600;
svg.selectAll('line')
  .data(ug_number)
  .enter().append('line')
  .attr('stroke', '#a4baff')
  .attr('stroke-width', '5')
  // .style('stroke', 'blue')
  .attr('x1', fixedX)
  .attr('y1', function(d,i){return (100+i*50);})
  .attr('x2', function(d){return fixedX+d;})
  .attr('y2', function(d,i){return (100+i*50);});
```

The expected outcome should look like the following (figure 11).



**Figure 11. Task 3 outcome**

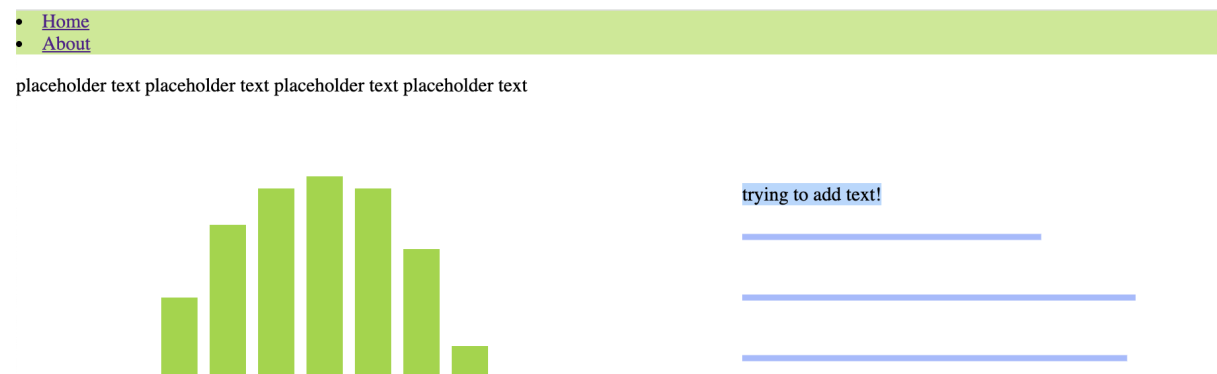
Note: there are two ways to define the color of the lines (attribute & style).  
Uncomment the line `.style('stroke', 'blue')` and refresh the webpage, what do you see?

#### Task 4. Add text

We can add text with the following code

```
svg.append('text')
    .attr('x', 600)
    .attr('y', 70)
    .text('trying to add text!');
```

where we pass in the text content to the `.text()` function. Refresh your browser, you should see that the text appears (figure 12).

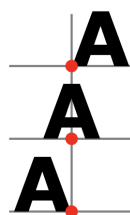


**Figure 12.** Our first added text

We can easily change it's appearance (font size, font color, storke color, etc.) with the following code

```
.style("fill", "black")
.style("stroke", "red")
.attr('font-size', 25);
```

In table 1, there is an optional attribute called “text-anchor” with three types of value: start-alignment, middle-alignment, and end-alignment (see figure 13 cited from the [documentation](#)).



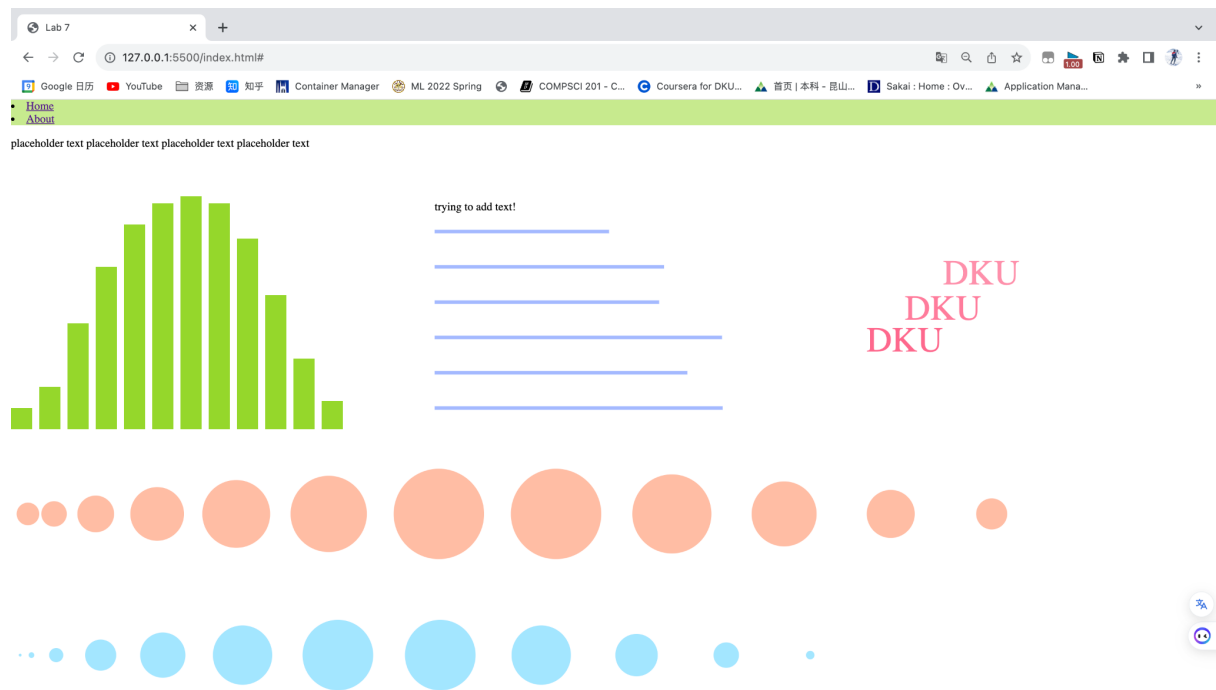
**Figure 13.** Start-, middle- or end-alignment (from top to bottom)

The below chunk of code displays the text “DKU” at the same canvas position but with different alignments. Please copy this to your file and refresh the browser to see if it helps you better understand this concept.

```
var fixedX = 1320;
    svg.append('text')
        .attr('x',fixedX)
        .attr('y',175)
        .attr('text-anchor', 'start')
        .attr('font-size',50)
        .style("fill", "#ff90ab")
        .text('DKU');

    svg.append('text')
        .attr('x',fixedX)
        .attr('y',225)
        .attr('text-anchor', 'middle')
        .attr('font-size',50)
        .style("fill", "#ff7d9c")
        .text('DKU');

    svg.append('text')
        .attr('x',fixedX)
        .attr('y',270)
        .attr('text-anchor', 'end')
        .attr('font-size',50)
        .style("fill", "#ff698d")
        .text('DKU');
```



**Figure 14.** All tasks' outcomes