Interaction

Data Visualization

Interaction

- It is often required to interact with our visualization to change the appearance of the visual elements or drill down information
  - Hovering, zooming, clicking etc.

- Topics
  - Mouse event
  - Tooltip
  - Drag
  - Brush
  - Zoom and pan

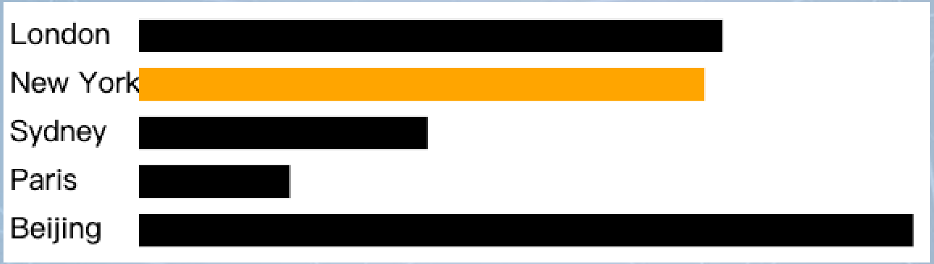
Mouse Events

- Detect the mouse click, mouse down, mouse enter, mouse leave, mouse over event

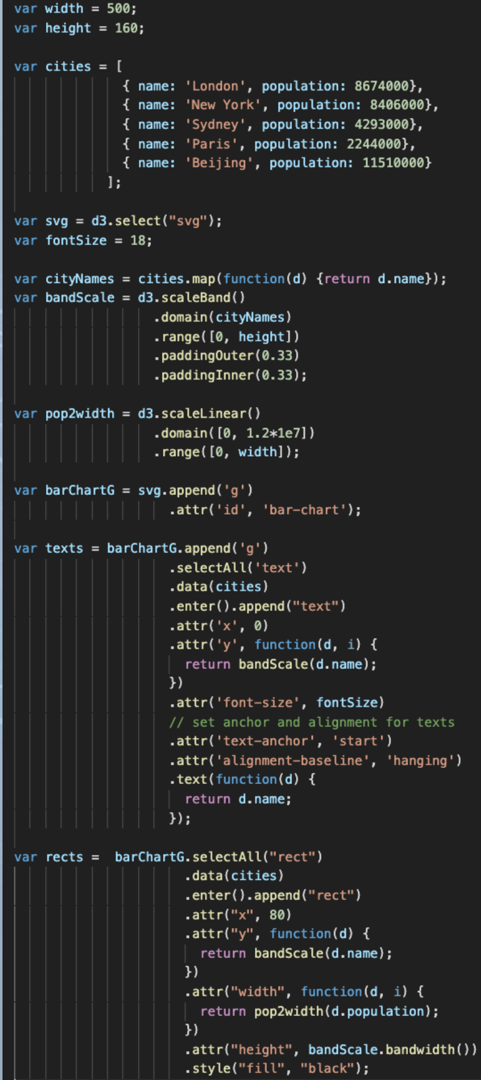
- selection.on(EventType, listenerFunction)
  - e.g. d3.selectAll(“rect”).on(“mouseover”, listenerFunction)
  - EventType: a name(String) of an event type (e.g. “mouseover”)
    - Any DOM event type supported by your browser may be used (not only mouse events)
    - Event list: https://developer.mozilla.org/en-US/docs/Web/Events#Standard_events
  - When a specified event is triggered, the listener function will be invoked

Mouse Events (Ex07-1)

- When the cursor is on a bar, the bar is highlighted (by different color)



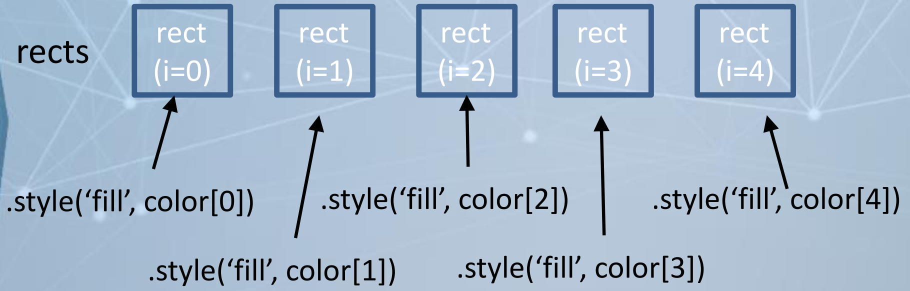
- The screenshot at the right hand side is the first part in main.js
  - It just draws the bars without mouse event



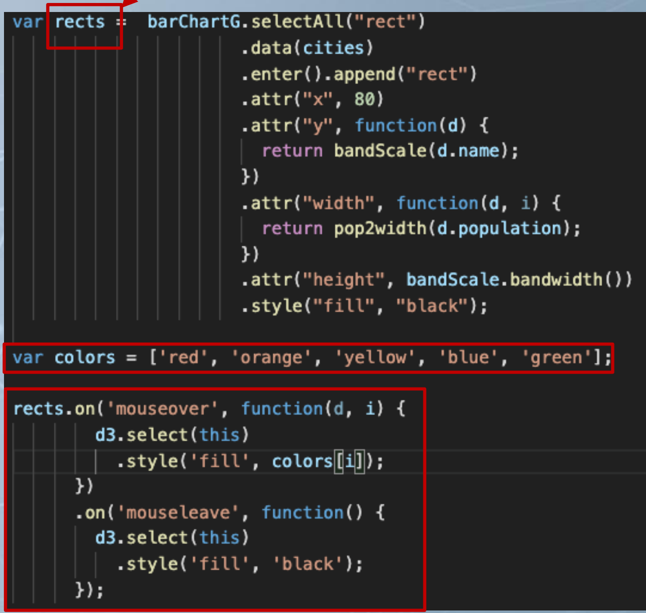
```
var width = 500;
var height = 160;

var cities = [
              { name: 'London', population: 8674000},
              { name: 'New York', population: 8406000},
              { name: 'Sydney', population: 4293000},
              { name: 'Paris', population: 2244000},
              { name: 'Beijing', population: 11510000}
            ];

var svg = d3.select("svg");
var fontSize = 18;

var cityNames = cities.map(function(d) {return d.name});
var bandScale = d3.scaleBand()
                  .domain(cityNames)
                  .range([0, height])
                  .paddingOuter(0.33)
                  .paddingInner(0.33);

var pop2width = d3.scaleLinear()
                  .domain([0, 1.2*1e7])
                  .range([0, width]);

var barChartG = svg.append('g')
                    .attr('id', 'bar-chart');

var texts = barChartG.append('g')
                    .selectAll('text')
                    .data(cities)
                    .enter().append("text")
                    .attr('x', 0)
                    .attr('y', function(d, i) {
                      return bandScale(d.name);
                    })
                    .attr('font-size', fontSize)
                    // set anchor and alignment for texts
                    .attr('text-anchor', 'start')
                    .attr('alignment-baseline', 'hanging')
                    .text(function(d) {
                      return d.name;
                    });

var rects =  barChartG.selectAll("rect")
                      .data(cities)
                      .enter().append("rect")
                      .attr("x", 80)
                      .attr("y", function(d) {
                        return bandScale(d.name);
                      })
                      .attr("width", function(d, i) {
                        return pop2width(d.population);
                      })
                      .attr("height", bandScale.bandwidth())
                      .style("fill", "black");
```

Mouse Events (Ex08-1)

- d3.select(this)
  - rects are all bars
  - rects.on(….) iterate through all bars
  - d3.select(this) indicates we are processing which bar “now”



Select all rects

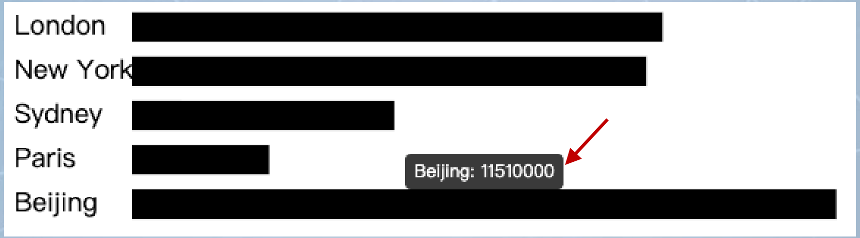


```
var rects =  barChartG.selectAll("rect")
                 .data(cities)
                 .enter().append("rect")
                 .attr("x", 80)
                 .attr("y", function(d) {
                   return bandScale(d.name);
                 })
                 .attr("width", function(d, i) {
                   return pop2width(d.population);
                 })
                 .attr("height", bandScale.bandwidth())
                 .style("fill", "black");

var colors = ['red', 'orange', 'yellow', 'blue', 'green'];

rects.on('mouseover', function(d, i) {
        d3.select(this)
          .style('fill', colors[i]);
      })
      .on('mouseleave', function() {
        d3.select(this)
          .style('fill', 'black');
      });
```

Because “rects” are all rectangles, this statement means that adding these listener functions to all rectangles

Tooltip - d3.tip()

- We can use d3.tip() to pop out a tooltip (small window) and show more details of a items for users



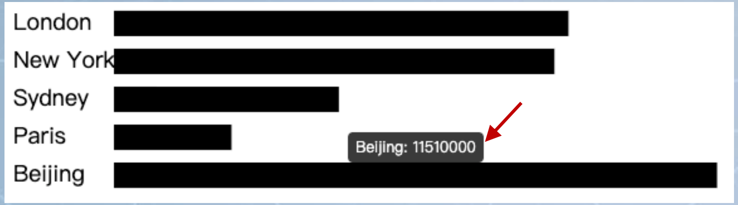
Tooltip - d3.tip()

- You can write tool tip function using d3 from scratch
- We introduce a plugin library to produce tooltip (make your life easily)
  - You have to import “d3-tip.js”
  - I have put “d3-tip.js” in Ex08-02 folder

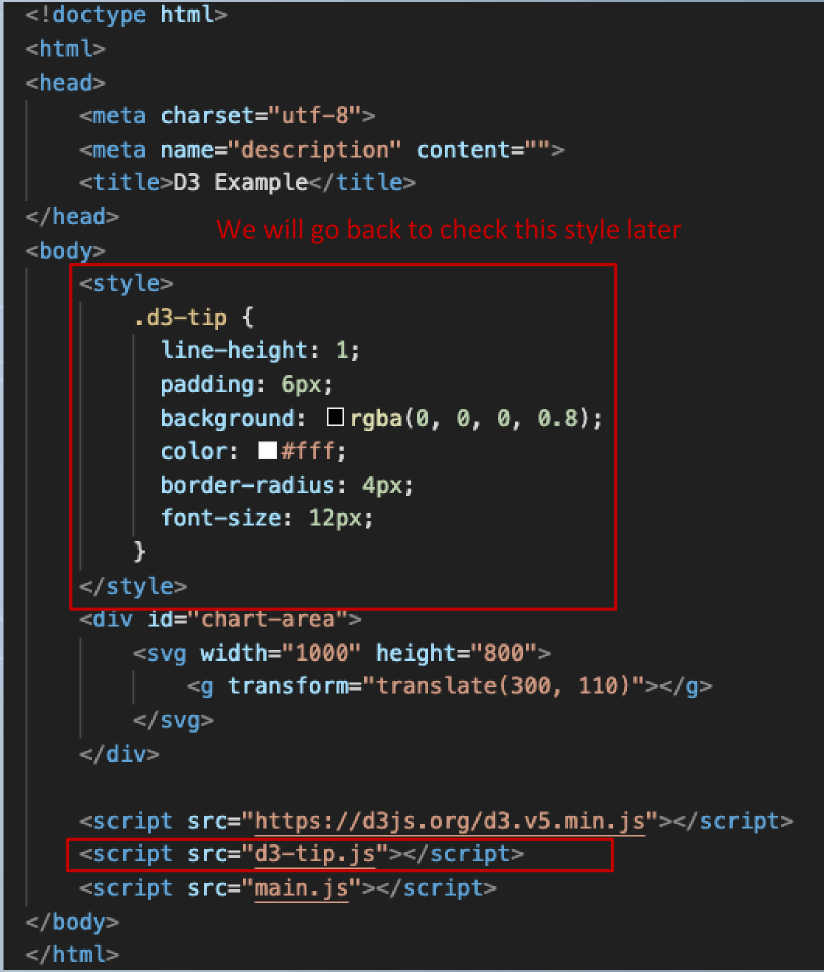
- Steps to use d3.tip() to create a tool tip
  - 1. initializing the tool tip
  - 2. calling the tip in the context of the visualization
  - 3. adding event listener

d3.tip() (Ex08-02)

What we want



Index.html



```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="description" content="">
    <title>D3 Example</title>
</head>
<body>
    <style>
        .d3-tip {
          line-height: 1;
          padding: 6px;
          background: rgba(0, 0, 0, 0.8);
          color: #fff;
          border-radius: 4px;
          font-size: 12px;
        }
    </style>
    <div id="chart-area">
        <svg width="1000" height="800">
            <g transform="translate(300, 110)"></g>
        </svg>
    </div>

    <script src="https://d3js.org/d3.v5.min.js"></script>
    <script src="d3-tip.js"></script>
    <script src="main.js"></script>
</body>
</html>
```

We will go back to check this style later

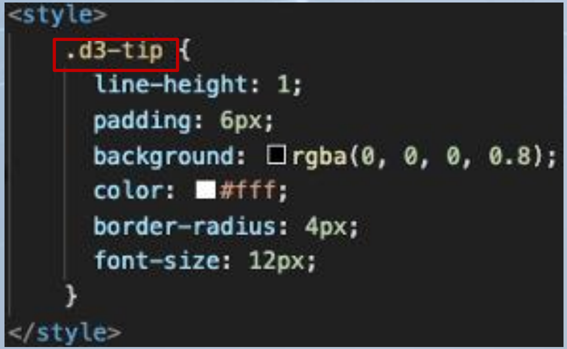
Import d3-tip.js before main.js
Because we will use it in main.js

d3.tip() (Ex08-02)

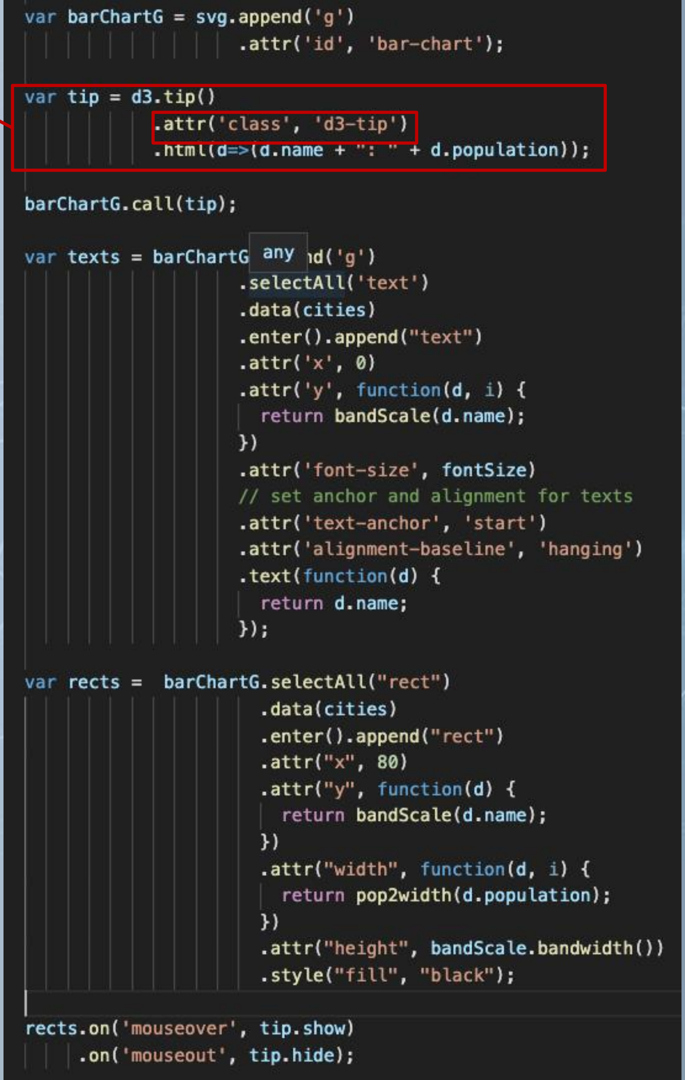
1. initializing the tool tip

- Ex08-02 is an extension of Ex08-01. Most of the code in main.js are the same.
  - We do not introduce the code to create the bar chart again

- **.attr('class', 'd3-tip')** determines the appearance of the tooltip
  - **d3-tip**? in index.html



```
<style>
    .d3-tip {
      line-height: 1;
      padding: 6px;
      background: rgba(0, 0, 0, 0.8);
      color: #fff;
      border-radius: 4px;
      font-size: 12px;
    }
</style>
```



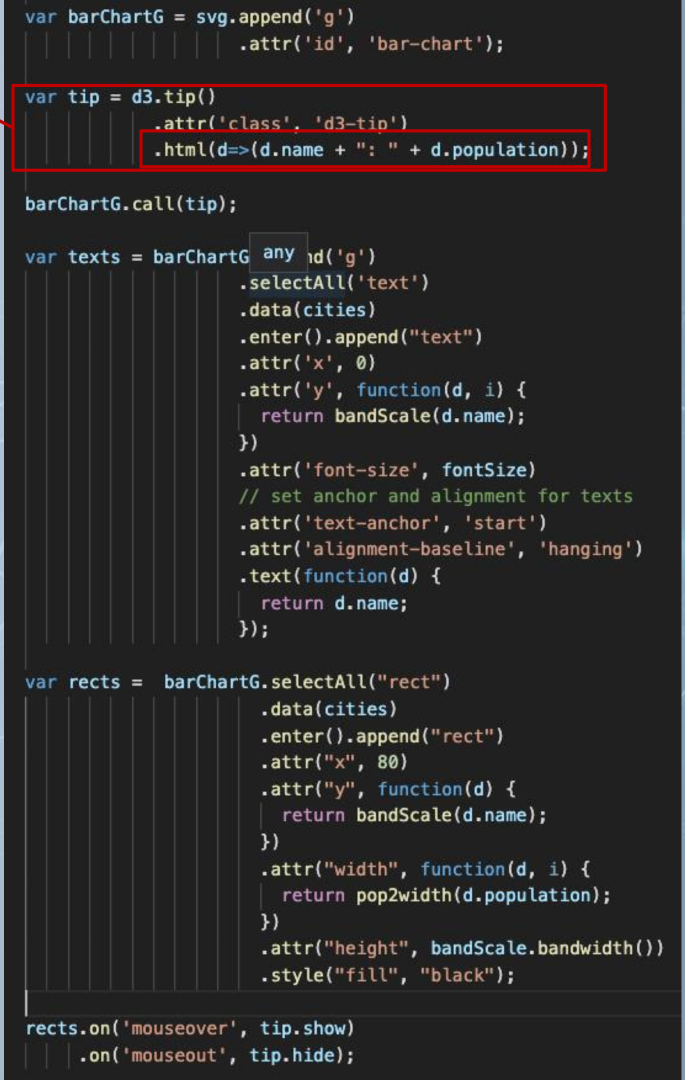
```
var barChartG = svg.append('g')
                    .attr('id', 'bar-chart');

var tip = d3.tip()
            .attr('class', 'd3-tip')
            .html(d=>(d.name + ": " + d.population));

barChartG.call(tip);

var texts = barChartG any nd('g')
                    .selectAll('text')
                    .data(cities)
                    .enter().append("text")
                    .attr('x', 0)
                    .attr('y', function(d, i) {
                      return bandScale(d.name);
                    })
                    .attr('font-size', fontSize)
                    // set anchor and alignment for texts
                    .attr('text-anchor', 'start')
                    .attr('alignment-baseline', 'hanging')
                    .text(function(d) {
                      return d.name;
                    });

var rects =  barChartG.selectAll("rect")
                      .data(cities)
                      .enter().append("rect")
                      .attr("x", 80)
                      .attr("y", function(d) {
                        return bandScale(d.name);
                      })
                      .attr("width", function(d, i) {
                        return pop2width(d.population);
                      })
                      .attr("height", bandScale.bandwidth())
                      .style("fill", "black");

rects.on('mouseover', tip.show)
    .on('mouseout', tip.hide);
```

1. initializing the tool tip

d3.tip() (Ex08-02)

- Ex08-02 is an extension of Ex08-01. Most of the code in main.js are the same.
  - We do not introduce the code to create the bar chart again

- **.html(d=>(d.name + “: ” + d.population))** determines what the content of the tooltip is
  - d is the data item sent to the tooltip



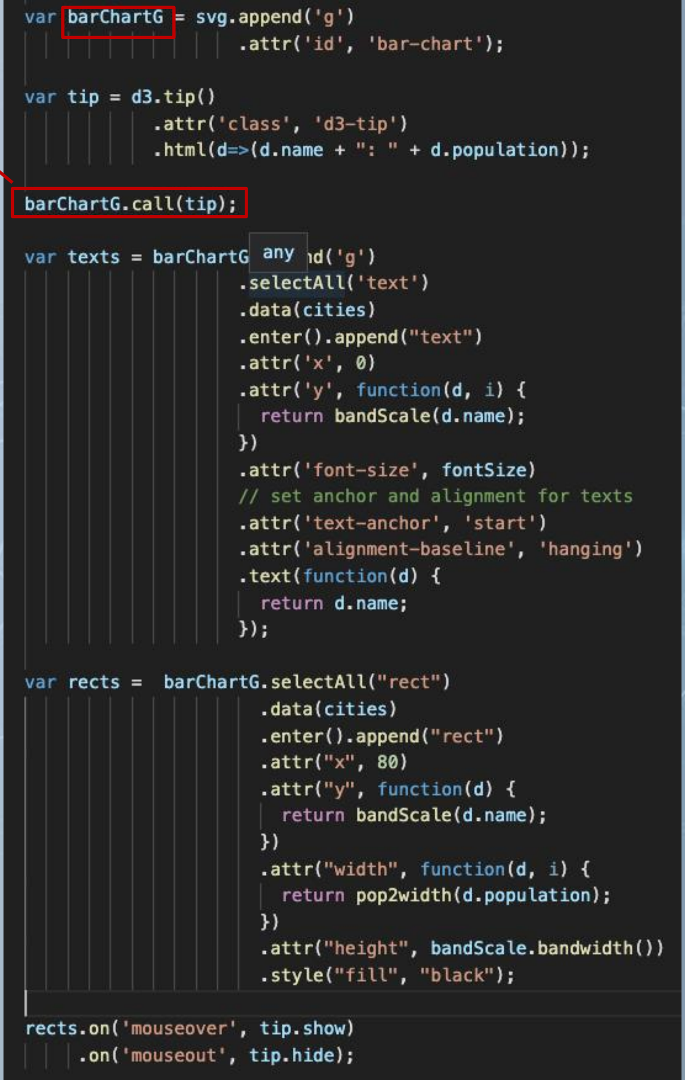
```
var barChartG = svg.append('g')
                    .attr('id', 'bar-chart');

var tip = d3.tip()
                .attr('class', 'd3-tip')
                .html(d=>(d.name + ": " + d.population));

barChartG.call(tip);

var texts = barChartG any nd('g')
                    .selectAll('text')
                    .data(cities)
                    .enter().append("text")
                    .attr('x', 0)
                    .attr('y', function(d, i) {
                      return bandScale(d.name);
                    })
                    .attr('font-size', fontSize)
                    // set anchor and alignment for texts
                    .attr('text-anchor', 'start')
                    .attr('alignment-baseline', 'hanging')
                    .text(function(d) {
                      return d.name;
                    });

var rects =  barChartG.selectAll("rect")
                      .data(cities)
                      .enter().append("rect")
                      .attr("x", 80)
                      .attr("y", function(d) {
                        return bandScale(d.name);
                      })
                      .attr("width", function(d, i) {
                        return pop2width(d.population);
                      })
                      .attr("height", bandScale.bandwidth())
                      .style("fill", "black");

rects.on('mouseover', tip.show)
      .on('mouseout', tip.hide);
```

2. calling the tip in the context of the visualization

d3.tip() (Ex08-02)

- Ex08-02 is an extension of Ex08-01. Most of the code in main.js are the same.
  - We do not introduce the code to create the bar chart again

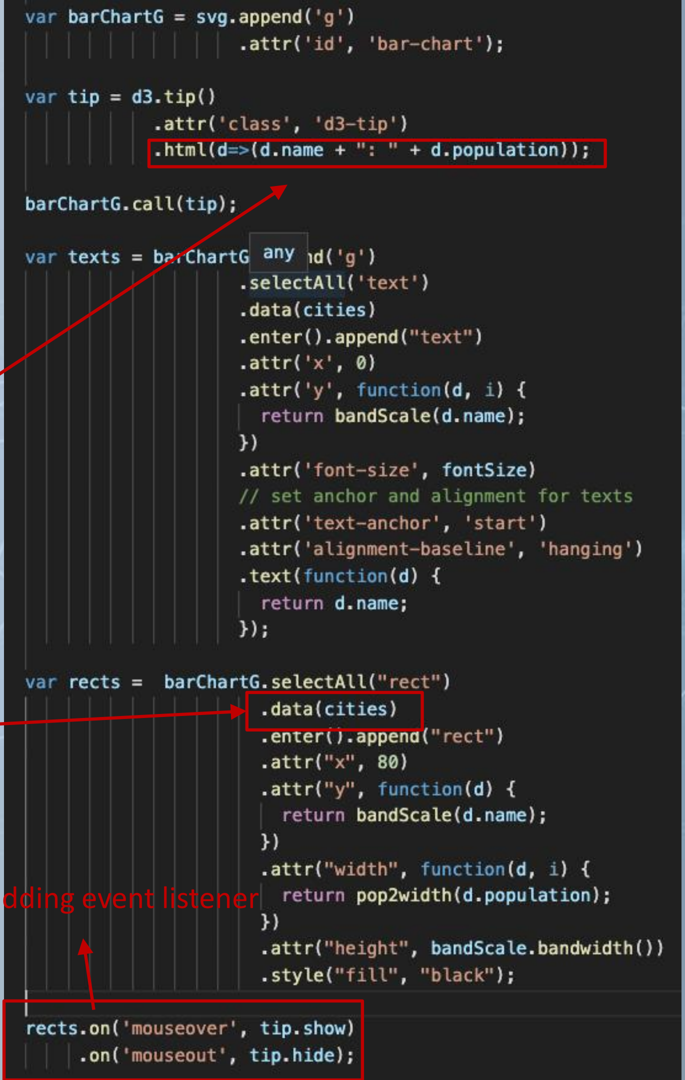
- barChartG.call(tip): add/enable the tool tip in barChartG



```
var barChartG = svg.append('g')
                   .attr('id', 'bar-chart');

var tip = d3.tip()
            .attr('class', 'd3-tip')
            .html(d=>(d.name + ": " + d.population));

barChartG.call(tip);

var texts = barChartG any nd('g')
                   .selectAll('text')
                   .data(cities)
                   .enter().append("text")
                   .attr('x', 0)
                   .attr('y', function(d, i) {
                     return bandScale(d.name);
                   })
                   .attr('font-size', fontSize)
                   // set anchor and alignment for texts
                   .attr('text-anchor', 'start')
                   .attr('alignment-baseline', 'hanging')
                   .text(function(d) {
                     return d.name;
                   });

var rects =  barChartG.selectAll("rect")
                    .data(cities)
                    .enter().append("rect")
                    .attr("x", 80)
                    .attr("y", function(d) {
                      return bandScale(d.name);
                    })
                    .attr("width", function(d, i) {
                      return pop2width(d.population);
                    })
                    .attr("height", bandScale.bandwidth())
                    .style("fill", "black");

rects.on('mouseover', tip.show)
     .on('mouseout', tip.hide);
```

d3.tip() (Ex08-02)

- Ex08-02 is an extension of Ex08-01. Most of the code in main.js are the same.
  - We do not introduce the code to create the bar chart again
- If ’mouseover’, show the tooltip for the item
  - What to show?
  - d? the data attached to each rect
- If ‘mouseout’, hide the tooltip



```
var barChartG = svg.append('g')
                   .attr('id', 'bar-chart');

var tip = d3.tip()
            .attr('class', 'd3-tip')
            .html(d=>(d.name + ": " + d.population));

barChartG.call(tip);

var texts = barChartG any nd('g')
                   .selectAll('text')
                   .data(cities)
                   .enter().append("text")
                   .attr('x', 0)
                   .attr('y', function(d, i) {
                     return bandScale(d.name);
                   })
                   .attr('font-size', fontSize)
                   // set anchor and alignment for texts
                   .attr('text-anchor', 'start')
                   .attr('alignment-baseline', 'hanging')
                   .text(function(d) {
                     return d.name;
                   });

var rects =  barChartG.selectAll("rect")
                    .data(cities)
                    .enter().append("rect")
                    .attr("x", 80)
                    .attr("y", function(d) {
                      return bandScale(d.name);
                    })
                    .attr("width", function(d, i) {
                      return pop2width(d.population);
                    })
                    .attr("height", bandScale.bandwidth())
                    .style("fill", "black");

rects.on('mouseover', tip.show)
     .on('mouseout', tip.hide);
```

3. adding event listener

Mouse Drag Behavior (d3.drag())

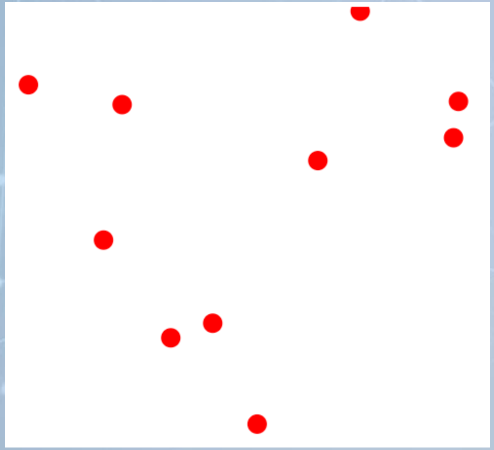
- Work on an item (such as a rect, a circle…)
  - Similar to d3.tip()
  - But you do not need extra plug-in library

- 2 steps to setup mouse drag event listener
  - 1. initialization and adding event listener
  - 2. calling the “drag” on each item

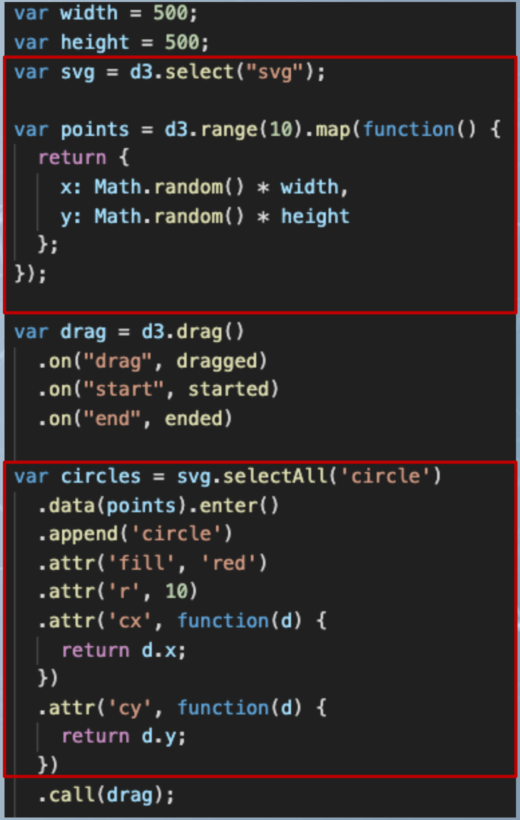
- Three type of events (d3.drag.on(EventType, listener))
  - “start”: when you click on the item
  - “drag”: when you drag the item
  - “end”: when you release the mouse

d3.drag() (Ex08-03)

- We will drag to move these circles



main.js



```
var width = 500;
var height = 500;
var svg = d3.select("svg");

var points = d3.range(10).map(function() {
  return {
    x: Math.random() * width,
    y: Math.random() * height
  };
});

var drag = d3.drag()
  .on("drag", dragged)
  .on("start", started)
  .on("end", ended)

var circles = svg.selectAll('circle')
  .data(points).enter()
  .append('circle')
  .attr('fill', 'red')
  .attr('r', 10)
  .attr('cx', function(d) {
    return d.x;
  })
  .attr('cy', function(d) {
    return d.y;
  })
  .call(drag);
```

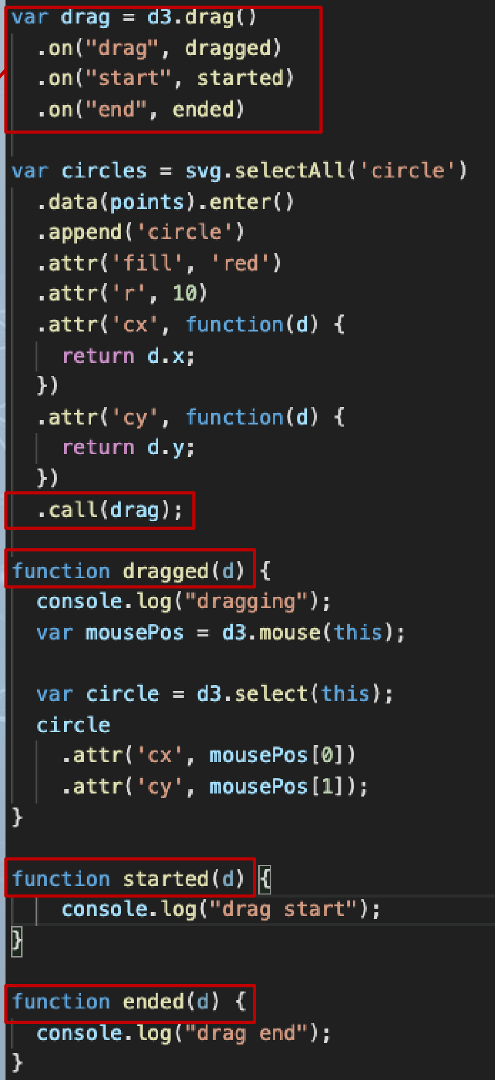
Create 10 red circles on the svg with random positions

d3.drag() (Ex08-03)

- If the drag starts, call “started” function
- If dragging, call “dragged” function
- If the drag end (release the mouse), call “ended” function

1. initialization and adding event listener

2. calling the “drag” on each item

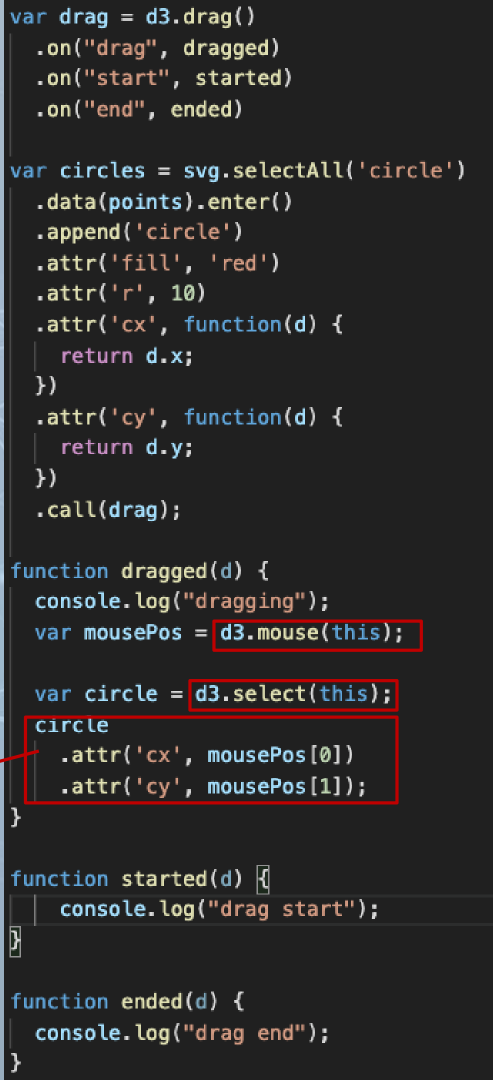


```
var drag = d3.drag()
  .on("drag", dragged)
  .on("start", started)
  .on("end", ended)

var circles = svg.selectAll('circle')
  .data(points).enter()
  .append('circle')
  .attr('fill', 'red')
  .attr('r', 10)
  .attr('cx', function(d) {
    return d.x;
  })
  .attr('cy', function(d) {
    return d.y;
  })
  .call(drag);

function dragged(d) {
  console.log("dragging");
  var mousePos = d3.mouse(this);

  var circle = d3.select(this);
  circle
    .attr('cx', mousePos[0])
    .attr('cy', mousePos[1]);
}

function started(d) {
    console.log("drag start");
}

function ended(d) {
  console.log("drag end");
}
```

d3.drag() (Ex08-03)

- d3.mouse(this)
  - Return the x and y coordinates (as an array [x,y]) of the current event relative to the specified container
    - svg or g

- d3.select(this)
  - Return the item which triggers current event

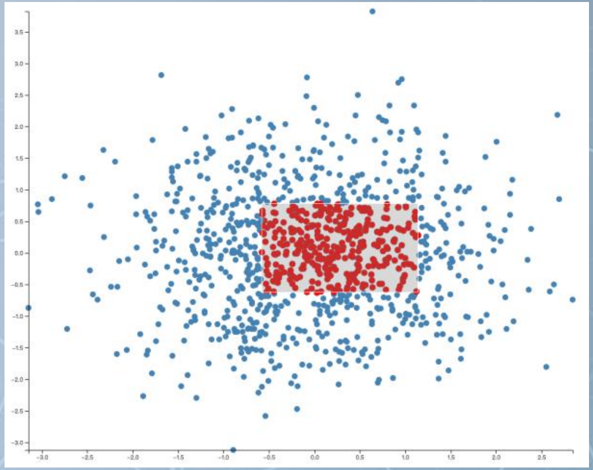


```
var drag = d3.drag()
  .on("drag", dragged)
  .on("start", started)
  .on("end", ended)

var circles = svg.selectAll('circle')
  .data(points).enter()
  .append('circle')
  .attr('fill', 'red')
  .attr('r', 10)
  .attr('cx', function(d) {
    return d.x;
  })
  .attr('cy', function(d) {
    return d.y;
  })
  .call(drag);

function dragged(d) {
  console.log("dragging");
  var mousePos = d3.mouse(this);

  var circle = d3.select(this);
  circle
    .attr('cx', mousePos[0])
    .attr('cy', mousePos[1]);
}

function started(d) {
    console.log("drag start");
}

function ended(d) {
  console.log("drag end");
}
```

Change the circle position

Brushing

- Brushing is the interactive specification a one- or two-dimensional selected region using a pointing gesture
  - Such as by clicking and dragging the mouse
- Brushing is often used to select discrete elements, such as dots in a scatterplot or files on a desktop
- We creates a scatterplot which supports brushing (Ex08-04)

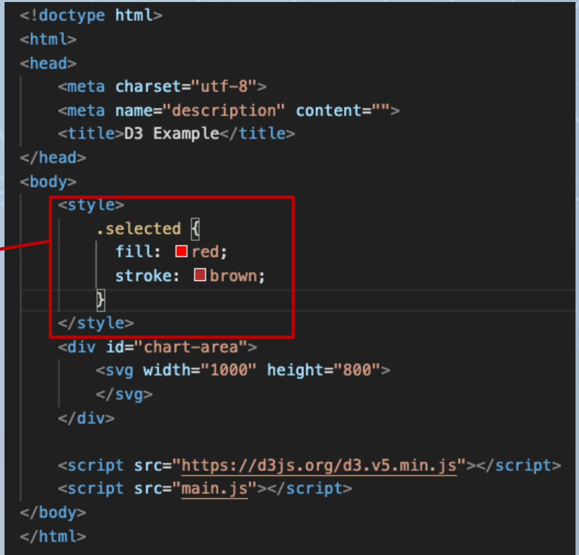


- d3.brush()
- Steps (again, similar to mouse drag)
  1. initialization, set the brushing area and adding event listener
  2. calling the “brush” on the visualization container

d3.brush() (Ex08-04)

- The first part in main.js is going to draw a scatterplot with axis

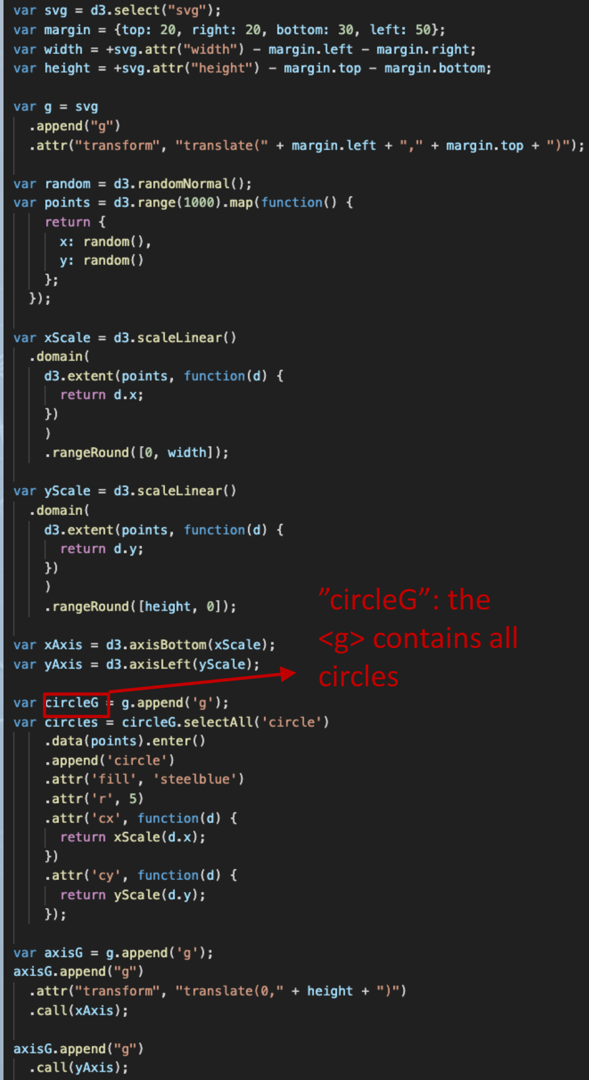
index.js



```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="description" content="">
    <title>D3 Example</title>
</head>
<body>
    <style>
        .selected {
          fill: 🟥red;
          stroke: 🟫brown;
        }
    </style>
    <div id="chart-area">
        <svg width="1000" height="800">
        </svg>
    </div>

    <script src="https://d3js.org/d3.v5.min.js"></script>
    <script src="main.js"></script>
</body>
</html>
```

The style to show the selected circle

main.js



```javascript
var svg = d3.select("svg");
var margin = {top: 20, right: 20, bottom: 30, left: 50};
var width = +svg.attr("width") - margin.left - margin.right;
var height = +svg.attr("height") - margin.top - margin.bottom;

var g = svg
  .append("g")
  .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

var random = d3.randomNormal();
var points = d3.range(1000).map(function() {
    return {
      x: random(),
      y: random()
    };
  });

var xScale = d3.scaleLinear()
  .domain(
    d3.extent(points, function(d) {
      return d.x;
    })
    )
    .rangeRound([0, width]);

var yScale = d3.scaleLinear()
  .domain(
    d3.extent(points, function(d) {
      return d.y;
    })
    )
    .rangeRound([height, 0]);

var xAxis = d3.axisBottom(xScale);
var yAxis = d3.axisLeft(yScale);

var circleG = g.append('g');
var circles = circleG.selectAll('circle')
    .data(points).enter()
    .append('circle')
    .attr('fill', 'steelblue')
    .attr('r', 5)
    .attr('cx', function(d) {
      return xScale(d.x);
    })
    .attr('cy', function(d) {
      return yScale(d.y);
    });

var axisG = g.append('g');
axisG.append("g")
  .attr("transform", "translate(0," + height + ")")
  .call(xAxis);

axisG.append("g")
  .call(yAxis);
```

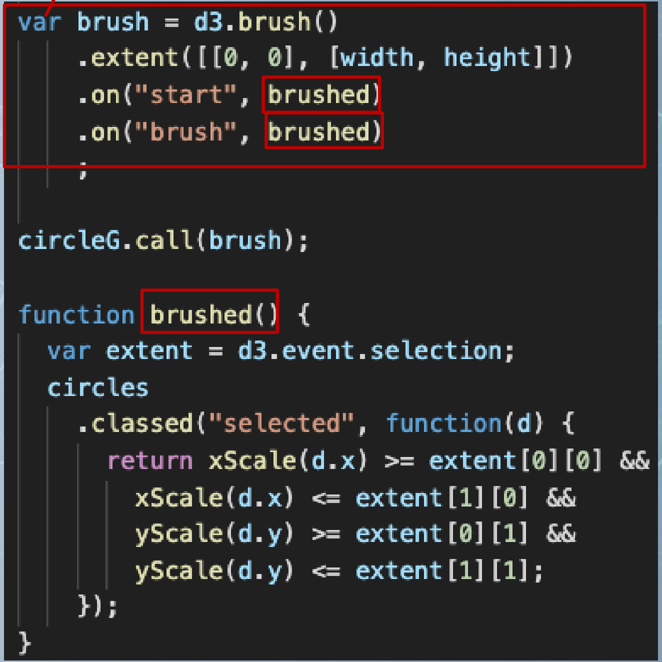
"circleG": the <g> contains all circles

d3.brush() (Ex08-04)

- d3.brush()
  - .extent()
    - Set the brushable area
    - We limit the brushing behavior within the screen
  - .on(“start”), .on(“brush”), .on(“end)”
    - If the brushing “start”/”brushing”/”end”, what should we do?

1. initialization, set the brushing area and adding event listener

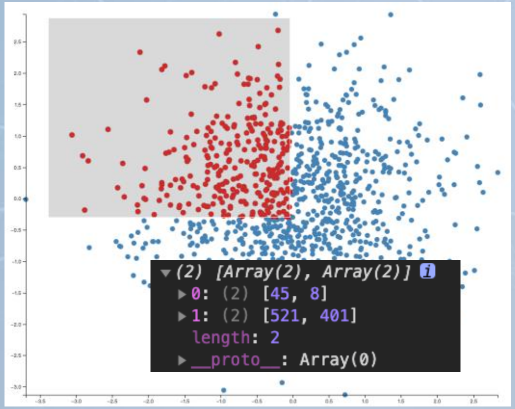
main.js



```
var brush = d3.brush()
    .extent([[0, 0], [width, height]])
    .on("start", brushed)
    .on("brush", brushed)
    ;

circleG.call(brush);

function brushed() {
  var extent = d3.event.selection;
  circles
    .classed("selected", function(d) {
      return xScale(d.x) >= extent[0][0] &&
        xScale(d.x) <= extent[1][0] &&
        yScale(d.y) >= extent[0][1] &&
        yScale(d.y) <= extent[1][1];
    });
}
```

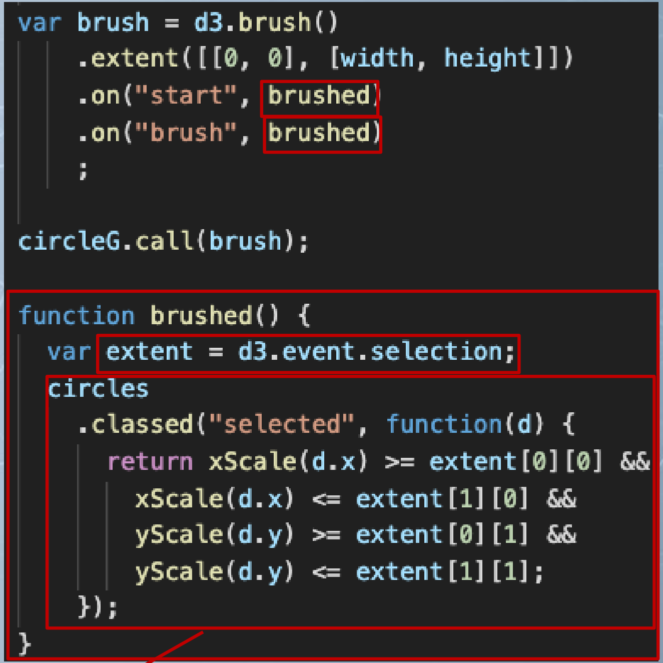
d3.brush() (Ex08-04)

- d3.event.selection
  - The coordinates of the selected region (top left and bottom right x-y)



```
▼(2) [Array(2), Array(2)]
 ▶0: (2) [45, 8]
 ▶1: (2) [521, 401]
  length: 2
 ▶__proto__: Array(0)
```

main.js

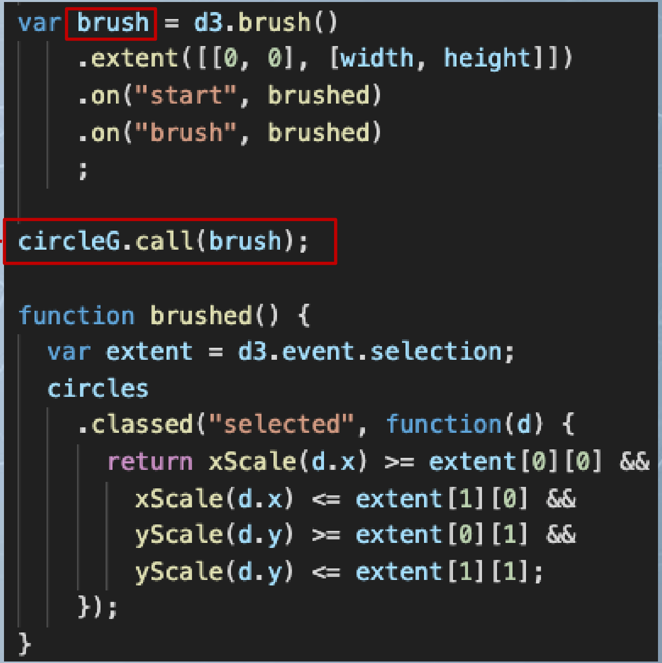


```
var brush = d3.brush()
    .extent([[0, 0], [width, height]])
    .on("start", brushed)
    .on("brush", brushed)
    ;

circleG.call(brush);

function brushed() {
  var extent = d3.event.selection;
  circles
    .classed("selected", function(d) {
      return xScale(d.x) >= extent[0][0] &&
        xScale(d.x) <= extent[1][0] &&
        yScale(d.y) >= extent[0][1] &&
        yScale(d.y) <= extent[1][1];
    });
}
```

If a circle is within the selected area, assign the style “selected” to it

d3.brush() (Ex08-04)

main.js



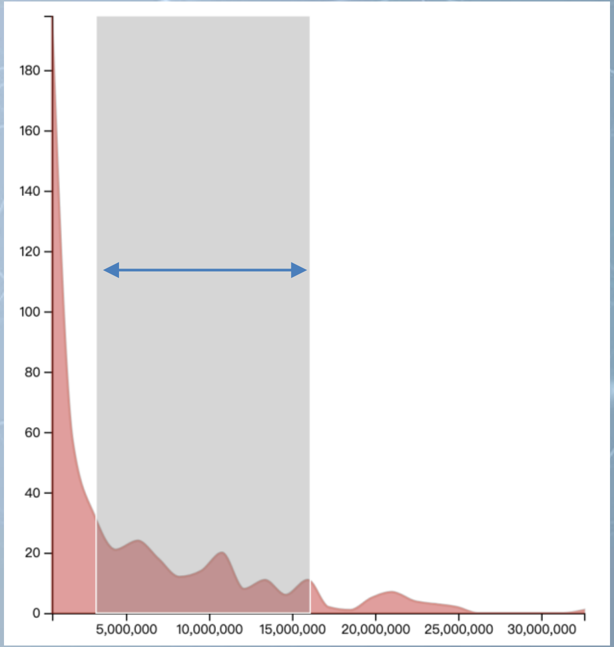
```
var brush = d3.brush()
    .extent([[0, 0], [width, height]])
    .on("start", brushed)
    .on("brush", brushed)
    ;

circleG.call(brush);

function brushed() {
  var extent = d3.event.selection;
  circles
    .classed("selected", function(d) {
      return xScale(d.x) >= extent[0][0] &&
        xScale(d.x) <= extent[1][0] &&
        yScale(d.y) >= extent[0][1] &&
        yScale(d.y) <= extent[1][1];
    });
}
```

2. calling the “brush” on the visualization container

Brush along X or Y-axis only

- d3.brushX()
- d3.brushY()

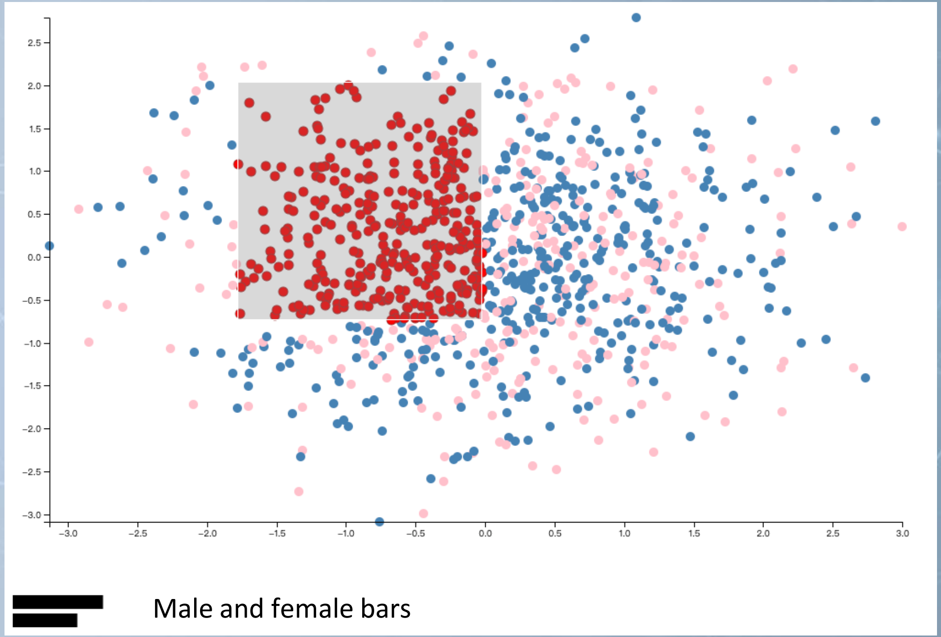
Ex: d3.brushX() only allow brushing horizontally



Ex08-05 (Link Views)

Blue: male
Pink: female

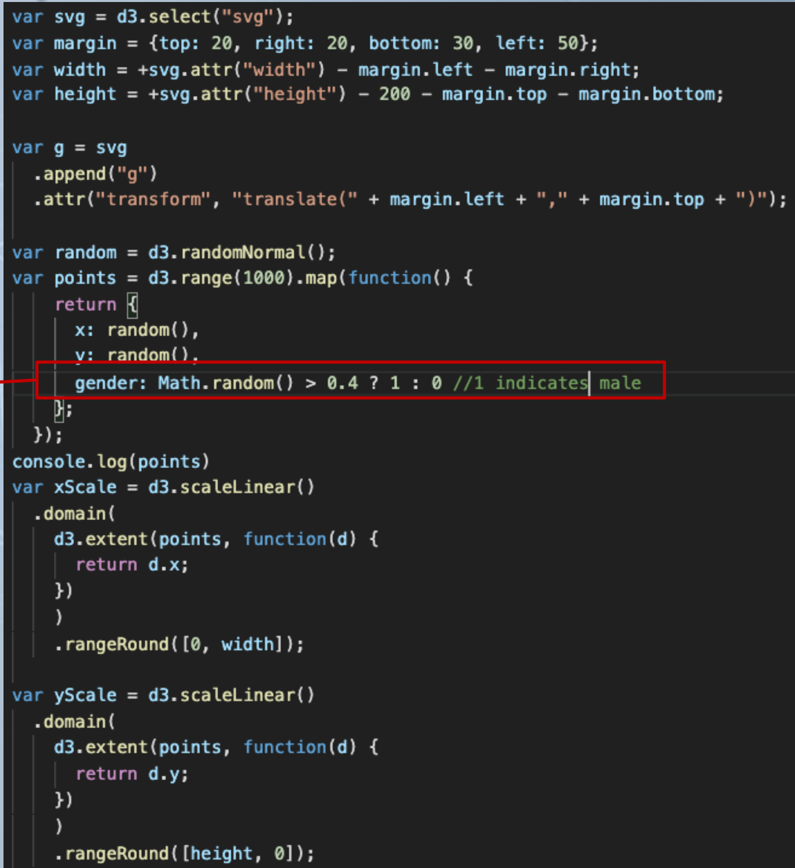
- Show the selected male and female counts by bar chart
  - Modify from Ex08-04
- File
  - index.html
  - main.js



Male and female bars

Ex08-05 (Link Views)

Add one more attribute to our data (Math.random() returns a random number between 0 and 1)



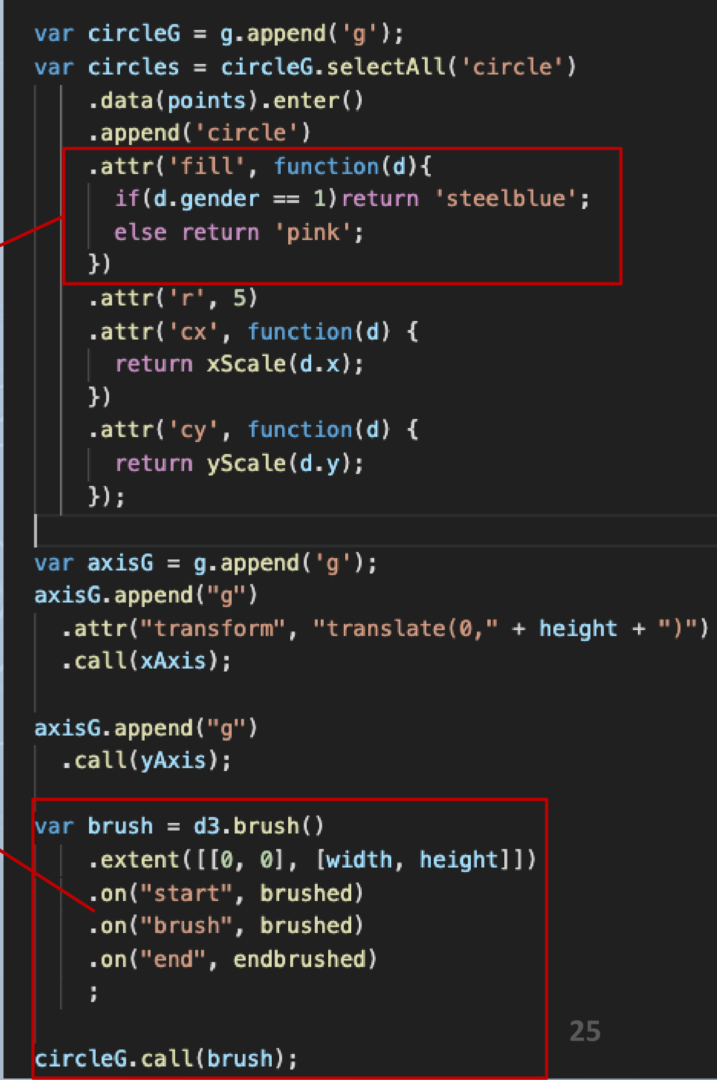
```
var svg = d3.select("svg");
var margin = {top: 20, right: 20, bottom: 30, left: 50};
var width = +svg.attr("width") - margin.left - margin.right;
var height = +svg.attr("height") - 200 - margin.top - margin.bottom;

var g = svg
  .append("g")
  .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

var random = d3.randomNormal();
var points = d3.range(1000).map(function() {
    return {
      x: random(),
      y: random(),
      gender: Math.random() > 0.4 ? 1 : 0 //1 indicates male
    };
  });
console.log(points)
var xScale = d3.scaleLinear()
  .domain(
    d3.extent(points, function(d) {
      return d.x;
    })
    )
    .rangeRound([0, width]);

var yScale = d3.scaleLinear()
  .domain(
    d3.extent(points, function(d) {
      return d.y;
    })
    )
    .rangeRound([height, 0]);
```

Ex08-05 (Link Views)

If the gender is 1, color the point by steelblue, otherwise, color it by pink

Run “brushed()” when we start to brush and are brushing. (we are going to calculate the number of selected male and female in brushed())

Run “endbrushed()” when brushing is done. (we are going to update the bar chart in endbrushed())

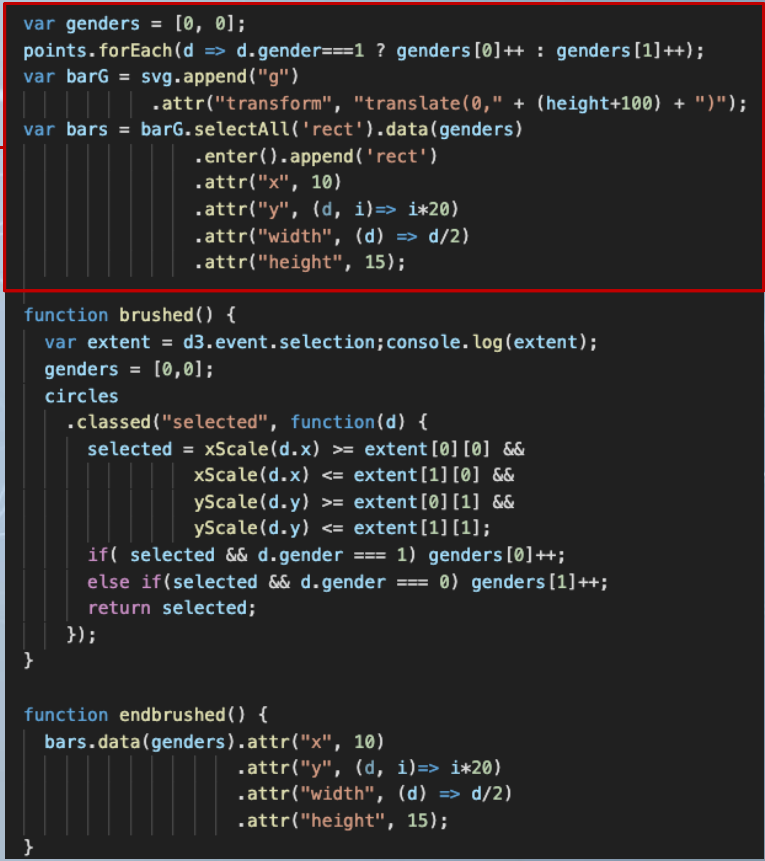


```
var circleG = g.append('g');
var circles = circleG.selectAll('circle')
    .data(points).enter()
    .append('circle')
    .attr('fill', function(d){
      if(d.gender == 1)return 'steelblue';
      else return 'pink';
    })
    .attr('r', 5)
    .attr('cx', function(d) {
      return xScale(d.x);
    })
    .attr('cy', function(d) {
      return yScale(d.y);
    });

var axisG = g.append('g');
axisG.append("g")
  .attr("transform", "translate(0," + height + ")")
  .call(xAxis);

axisG.append("g")
  .call(yAxis);

var brush = d3.brush()
    .extent([[0, 0], [width, height]])
    .on("start", brushed)
    .on("brush", brushed)
    .on("end", endbrushed)
    ;

circleG.call(brush);
```

Ex08-05 (Link Views)

Initially, calculate the total number of male and female, and plot the bars

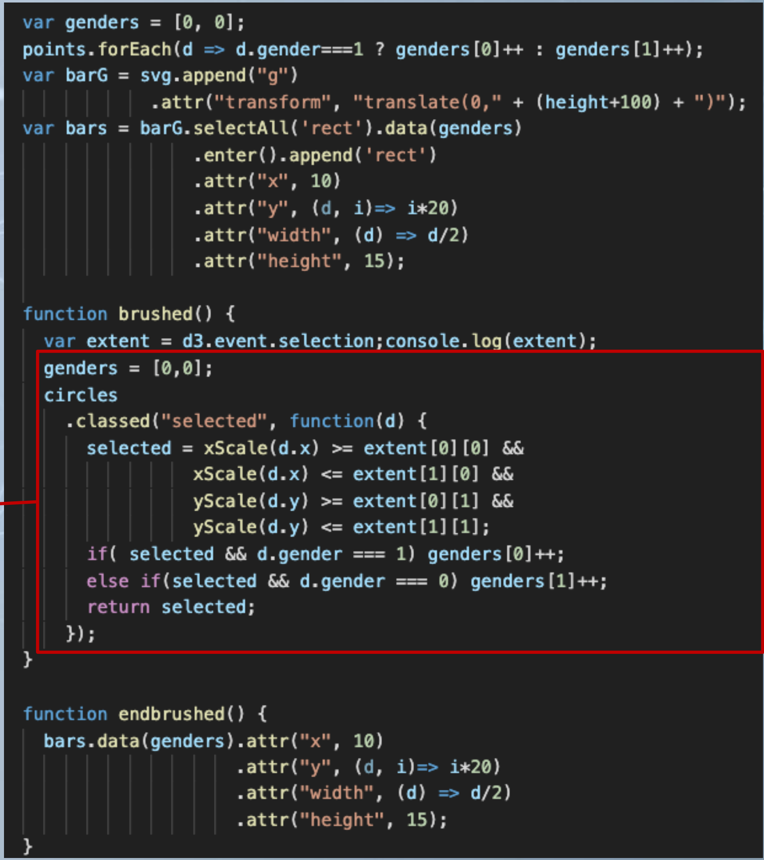
Only for the purpose of link view demonstration, we do not add axis and scale function (just make it short and simple)



```
var genders = [0, 0];
points.forEach(d => d.gender===1 ? genders[0]++ : genders[1]++);
var barG = svg.append("g")
             .attr("transform", "translate(0," + (height+100) + ")");
var bars = barG.selectAll('rect').data(genders)
                 .enter().append('rect')
                 .attr("x", 10)
                 .attr("y", (d, i)=> i*20)
                 .attr("width", (d) => d/2)
                 .attr("height", 15);

function brushed() {
  var extent = d3.event.selection;console.log(extent);
  genders = [0,0];
  circles
    .classed("selected", function(d) {
      selected = xScale(d.x) >= extent[0][0] &&
                 xScale(d.x) <= extent[1][0] &&
                 yScale(d.y) >= extent[0][1] &&
                 yScale(d.y) <= extent[1][1];
      if( selected && d.gender === 1) genders[0]++;
      else if(selected && d.gender === 0) genders[1]++;
      return selected;
    });
}

function endbrushed() {
  bars.data(genders).attr("x", 10)
                    .attr("y", (d, i)=> i*20)
                    .attr("width", (d) => d/2)
                    .attr("height", 15);
}
```

Ex08-05 (Link Views)

When the user is brushing, Recalculate the number of selected male and female, and store the result in "genders"

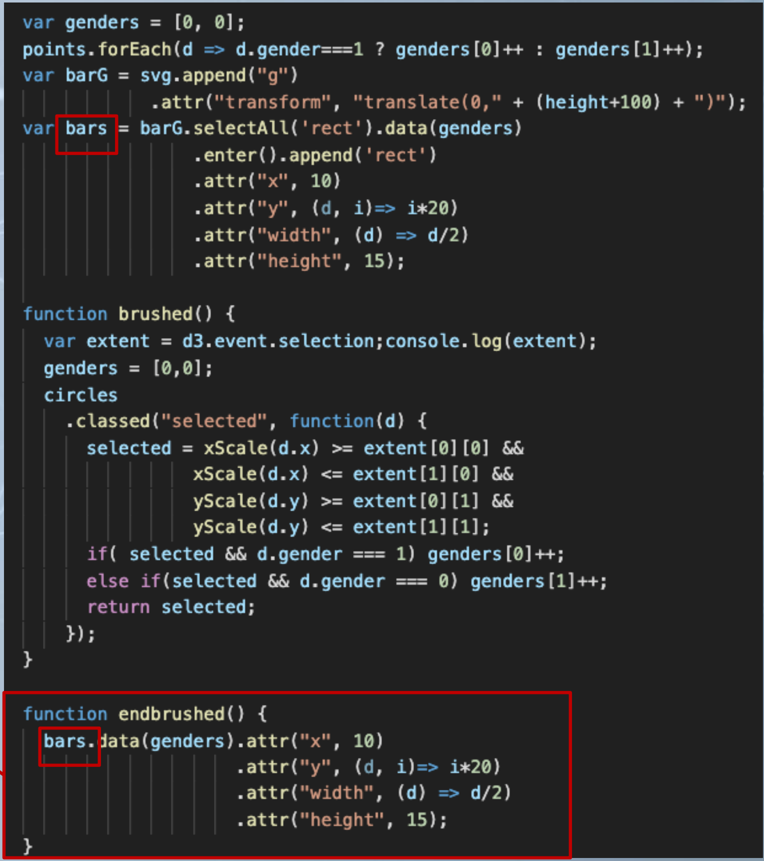


```
var genders = [0, 0];
points.forEach(d => d.gender===1 ? genders[0]++ : genders[1]++);
var barG = svg.append("g")
            .attr("transform", "translate(0," + (height+100) + ")");
var bars = barG.selectAll('rect').data(genders)
                .enter().append('rect')
                .attr("x", 10)
                .attr("y", (d, i)=> i*20)
                .attr("width", (d) => d/2)
                .attr("height", 15);

function brushed() {
  var extent = d3.event.selection;console.log(extent);
  genders = [0,0];
  circles
    .classed("selected", function(d) {
      selected = xScale(d.x) >= extent[0][0] &&
                 xScale(d.x) <= extent[1][0] &&
                 yScale(d.y) >= extent[0][1] &&
                 yScale(d.y) <= extent[1][1];
      if( selected && d.gender === 1) genders[0]++;
      else if(selected && d.gender === 0) genders[1]++;
      return selected;
    });
}

function endbrushed() {
  bars.data(genders).attr("x", 10)
                    .attr("y", (d, i)=> i*20)
                    .attr("width", (d) => d/2)
                    .attr("height", 15);
}
```

Ex08-05 (Link Views)

When the brushing is done, update the bar chart.

Bind the data (genders) to bars (rect) again and update attributes of rects

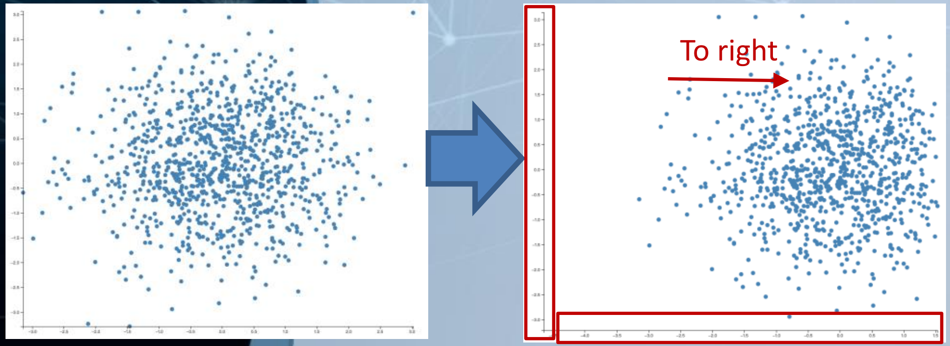


```
var genders = [0, 0];
points.forEach(d => d.gender===1 ? genders[0]++ : genders[1]++);
var barG = svg.append("g")
            .attr("transform", "translate(0," + (height+100) + ")");
var bars = barG.selectAll('rect').data(genders)
                .enter().append('rect')
                .attr("x", 10)
                .attr("y", (d, i)=> i*20)
                .attr("width", (d) => d/2)
                .attr("height", 15);

function brushed() {
  var extent = d3.event.selection;console.log(extent);
  genders = [0,0];
  circles
    .classed("selected", function(d) {
      selected = xScale(d.x) >= extent[0][0] &&
                 xScale(d.x) <= extent[1][0] &&
                 yScale(d.y) >= extent[0][1] &&
                 yScale(d.y) <= extent[1][1];
      if( selected && d.gender === 1) genders[0]++;
      else if(selected && d.gender === 0) genders[1]++;
      return selected;
    });
}

function endbrushed() {
  bars.data(genders).attr("x", 10)
                    .attr("y", (d, i)=> i*20)
                    .attr("width", (d) => d/2)
                    .attr("height", 15);
}
```

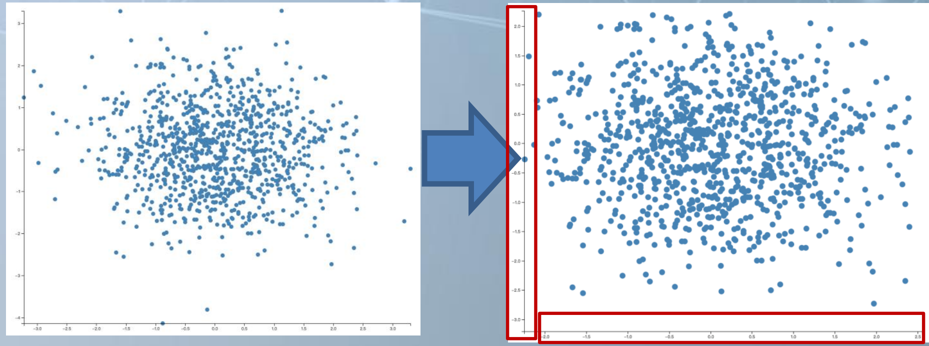
Zooming and Panning

- Zooming and panning are popular interaction techniques which let the user focus on a region of interest by restricting the view
- Zooming and panning are widely used in web-based mapping, but can also be used with visualization such as time-series and scatterplots

Pan by mouse



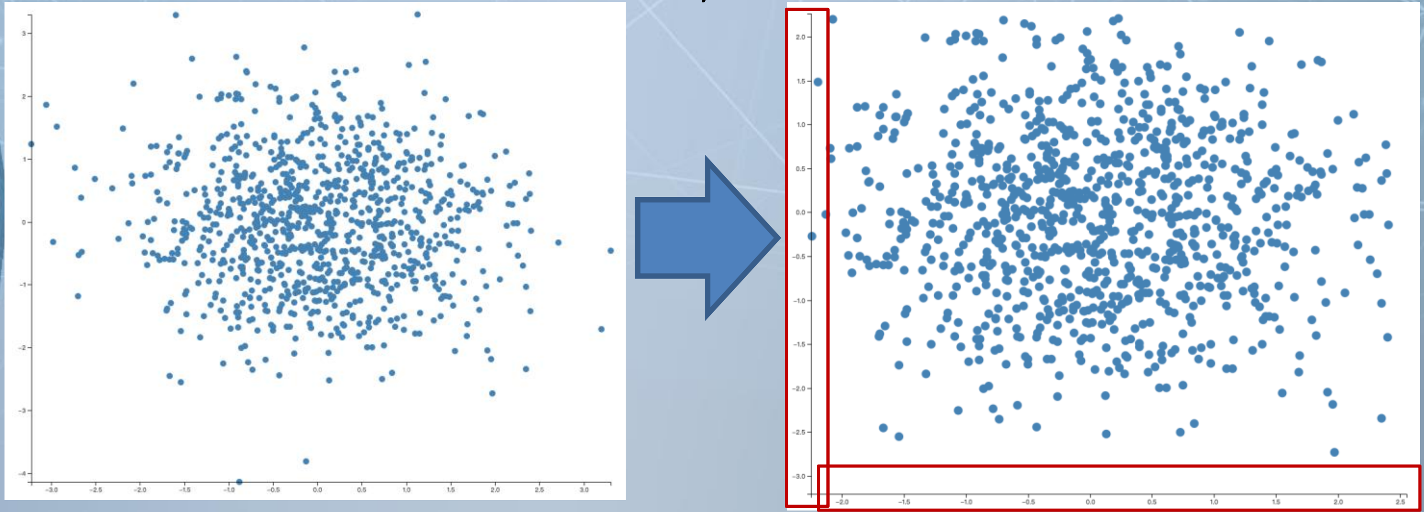
zoom in by mouse



Zooming and Panning (Ex08-05)

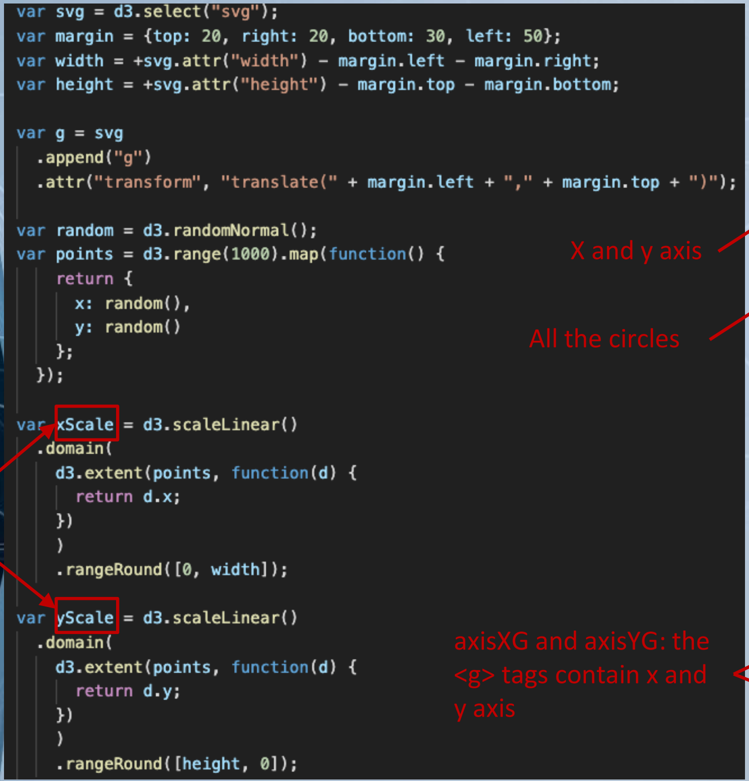
- d3.zoom() for both zooming and panning
- Steps (again, similar to brushing)
  - 1. initialization, set the scale extent and adding event listener
  - 2. calling the “zoom” on the visualization container
  - (you have to handle the x and y axis after panning and zooming)

zoom in by mouse



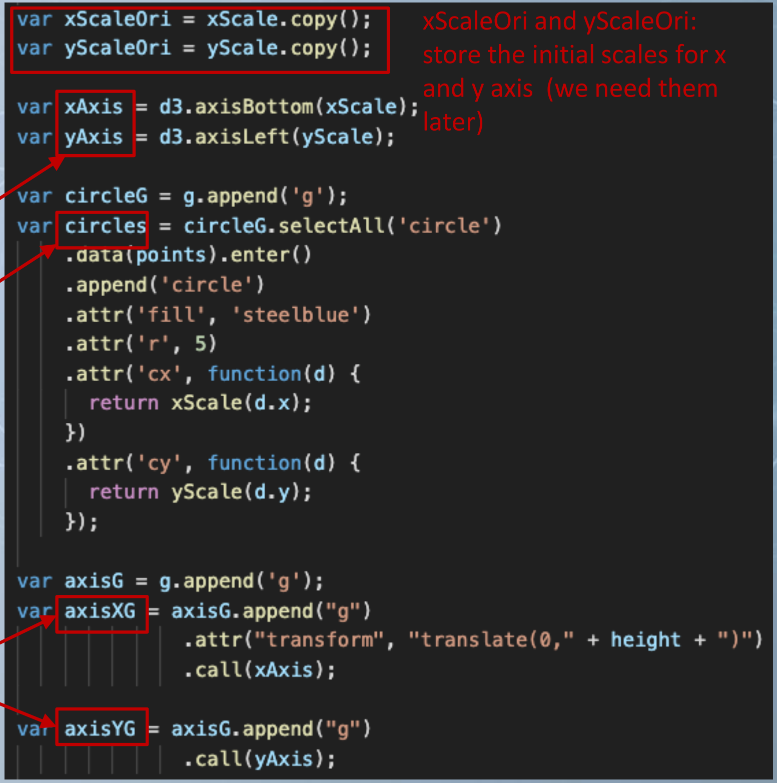
Zooming and Panning (Ex08-05)

main.js (draw the scatterplot with x-y axis first)



```
var svg = d3.select("svg");
var margin = {top: 20, right: 20, bottom: 30, left: 50};
var width = +svg.attr("width") - margin.left - margin.right;
var height = +svg.attr("height") - margin.top - margin.bottom;

var g = svg
  .append("g")
  .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

var random = d3.randomNormal();
var points = d3.range(1000).map(function() {
    return {
      x: random(),
      y: random()
    };
  });

var xScale = d3.scaleLinear()
  .domain(
    d3.extent(points, function(d) {
      return d.x;
    })
    )
    .rangeRound([0, width]);

var yScale = d3.scaleLinear()
  .domain(
    d3.extent(points, function(d) {
      return d.y;
    })
    )
    .rangeRound([height, 0]);
```

xScale and yScale: scale functions to create x-y axis and map circles to the plot



```
var xScaleOri = xScale.copy();
var yScaleOri = yScale.copy();

var xAxis = d3.axisBottom(xScale);
var yAxis = d3.axisLeft(yScale);

var circleG = g.append('g');
var circles = circleG.selectAll('circle')
  .data(points).enter()
  .append('circle')
  .attr('fill', 'steelblue')
  .attr('r', 5)
  .attr('cx', function(d) {
    return xScale(d.x);
  })
  .attr('cy', function(d) {
    return yScale(d.y);
  });

var axisG = g.append('g');
var axisXG = axisG.append("g")
          .attr("transform", "translate(0," + height + ")")
          .call(xAxis);

var axisYG = axisG.append("g")
          .call(yAxis);
```

xScaleOri and yScaleOri: store the initial scales for x and y axis (we need them later)

X and y axis

All the circles

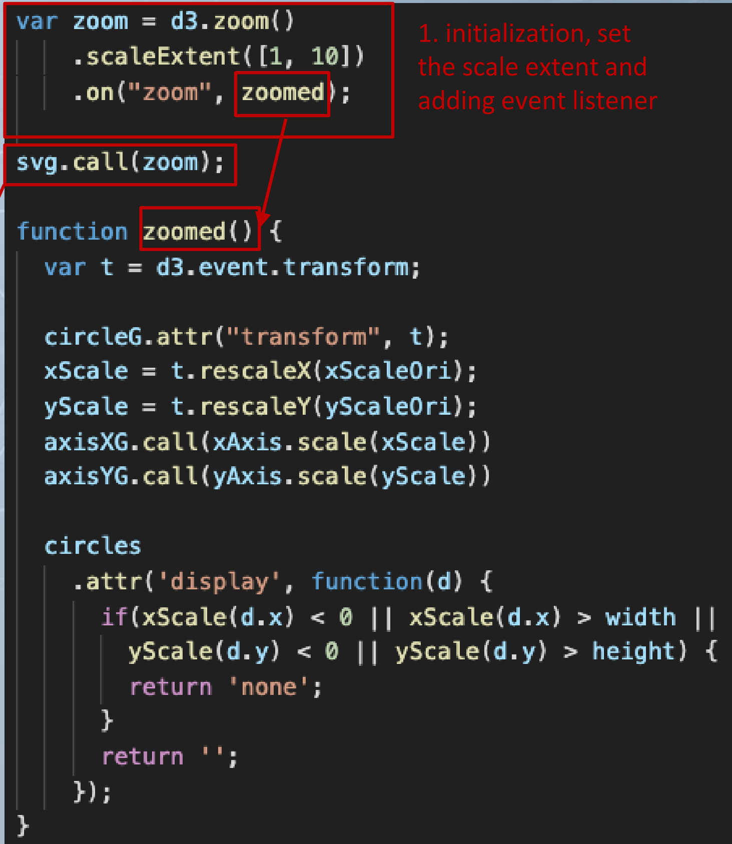
axisXG and axisYG: the <g> tags contain x and y axis

Zooming and Panning (Ex08-05)

- d3.zoom()
  - .on(”zoom”, listener): trigger the listener when zooming or panning
  - .scaleExtent([min, max]): sets the min and max zooming factors
    - E.g.: scaleExtent([1, 10])
    - Minimal scale factor: original scale
    - Maximal scale factor: 10x

2. calling the “zoom” on the visualization container

main.js



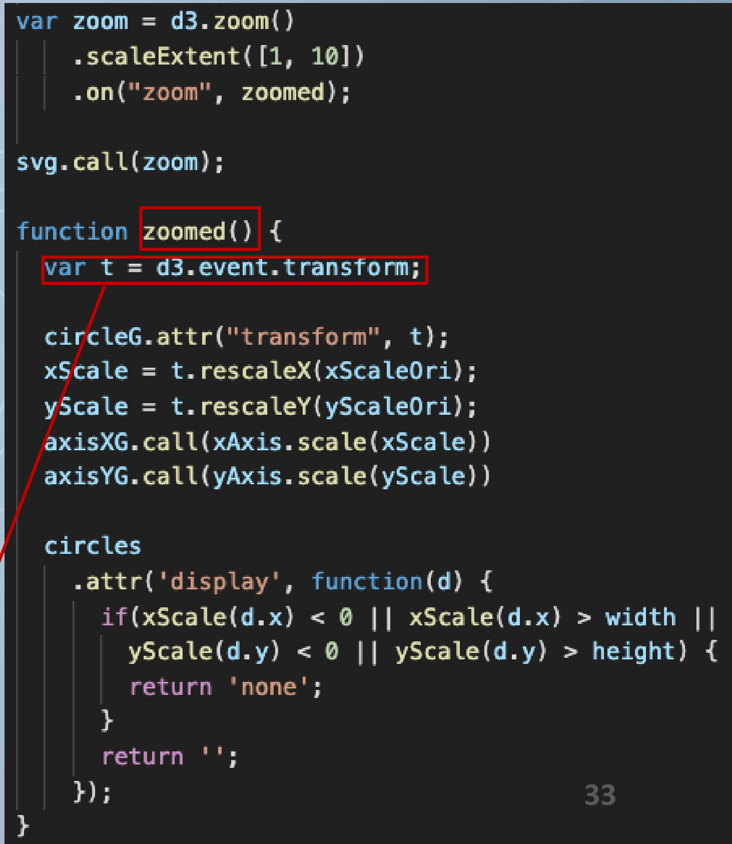
```
var zoom = d3.zoom()
    .scaleExtent([1, 10])
    .on("zoom", zoomed);

svg.call(zoom);

function zoomed() {
  var t = d3.event.transform;

  circleG.attr("transform", t);
  xScale = t.rescaleX(xScaleOri);
  yScale = t.rescaleY(yScaleOri);
  axisXG.call(xAxis.scale(xScale))
  axisYG.call(yAxis.scale(yScale))

  circles
    .attr('display', function(d) {
      if(xScale(d.x) < 0 || xScale(d.x) > width ||
        yScale(d.y) < 0 || yScale(d.y) > height) {
        return 'none';
      }
      return '';
    });
}
```

1. initialization, set the scale extent and adding event listener

Zooming and Panning (Ex08-05)

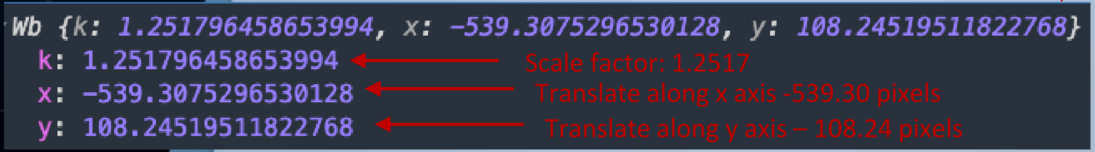
- After we zoom and pan, we have to change the scale(k), and shifting (Δx and Δy) of points
- We can get k, Δx and Δy from d3.event.transform
  - Tell us how to correctly transform the circles

main.js



```
var zoom = d3.zoom()
    .scaleExtent([1, 10])
    .on("zoom", zoomed);

svg.call(zoom);

function zoomed() {
  var t = d3.event.transform;

  circleG.attr("transform", t);
  xScale = t.rescaleX(xScaleOri);
  yScale = t.rescaleY(yScaleOri);
  axisXG.call(xAxis.scale(xScale))
  axisYG.call(yAxis.scale(yScale))

  circles
    .attr('display', function(d) {
      if(xScale(d.x) < 0 || xScale(d.x) > width ||
        yScale(d.y) < 0 || yScale(d.y) > height) {
        return 'none';
      }
      return '';
    });
}
```

Example



```
Wb {k: 1.2517964586539994, x: -539.3075296530128, y: 108.24519511822768}
  k: 1.2517964586539994
  x: -539.3075296530128
  y: 108.24519511822768
```

Scale factor: 1.2517

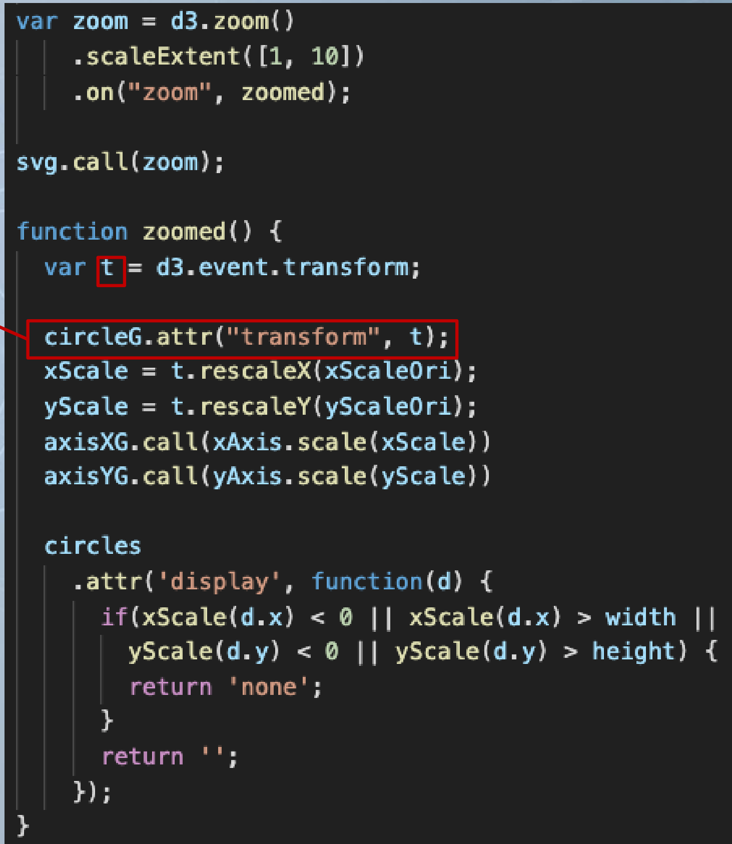
Translate along x axis -539.30 pixels

Translate along y axis – 108.24 pixels

Zooming and Panning (Ex08-05)

main.js

- Update the x and y axes
- circleG: the <g> contains all circles
  - Transform it by “t”



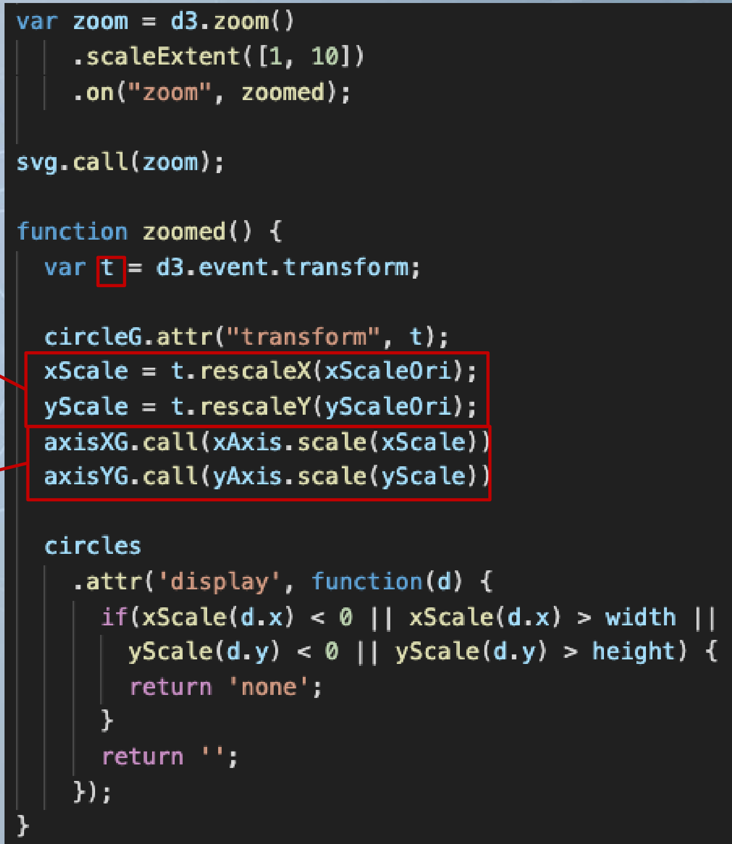
```
var zoom = d3.zoom()
    .scaleExtent([1, 10])
    .on("zoom", zoomed);

svg.call(zoom);

function zoomed() {
  var t = d3.event.transform;

  circleG.attr("transform", t);
  xScale = t.rescaleX(xScaleOri);
  yScale = t.rescaleY(yScaleOri);
  axisXG.call(xAxis.scale(xScale))
  axisYG.call(yAxis.scale(yScale))

  circles
    .attr('display', function(d) {
      if(xScale(d.x) < 0 || xScale(d.x) > width ||
        yScale(d.y) < 0 || yScale(d.y) > height) {
        return 'none';
      }
      return '';
    });
}
```

Zooming and Panning (Ex08-05)

- Update the x and y axes

- circleG: the <g> contains all circles
  - Transform it by “t”
- Use “t” to rescale the original x and y scale functions (xScaleOri and yScaleOri)
- And, set x and y axes using the new x and y scale functions

main.js



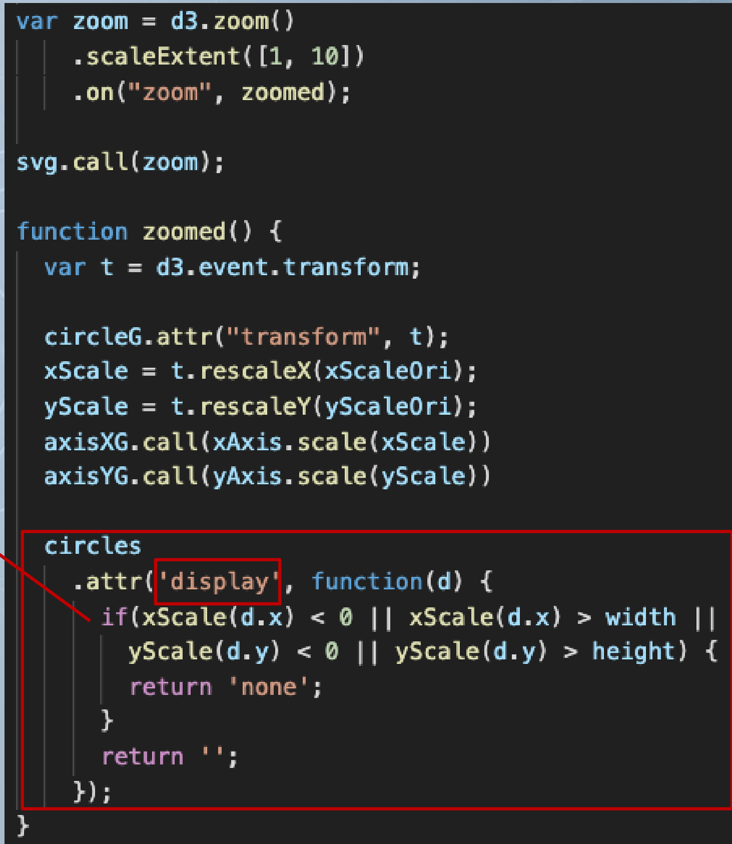
```
var zoom = d3.zoom()
    .scaleExtent([1, 10])
    .on("zoom", zoomed);

svg.call(zoom);

function zoomed() {
  var t = d3.event.transform;

  circleG.attr("transform", t);
  xScale = t.rescaleX(xScaleOri);
  yScale = t.rescaleY(yScaleOri);
  axisXG.call(xAxis.scale(xScale))
  axisYG.call(yAxis.scale(yScale))

  circles
    .attr('display', function(d) {
      if(xScale(d.x) < 0 || xScale(d.x) > width ||
        yScale(d.y) < 0 || yScale(d.y) > height) {
        return 'none';
      }
      return '';
    });
}
```

Zooming and Panning (Ex08-05)

- Update the circles
  - If the circles are still within the plot region, show it. Otherwise, hide it.
  - How to hide it? Set the circle's 'display' attribute to 'none'

- Calculate where to show a circle by sending the the circle's raw data (x, y) to new x- and y- scale function.
  - If the position you should draw it is out of the plot area, hide it

main.js



```
var zoom = d3.zoom()
    .scaleExtent([1, 10])
    .on("zoom", zoomed);

svg.call(zoom);

function zoomed() {
  var t = d3.event.transform;

  circleG.attr("transform", t);
  xScale = t.rescaleX(xScaleOri);
  yScale = t.rescaleY(yScaleOri);
  axisXG.call(xAxis.scale(xScale))
  axisYG.call(yAxis.scale(yScale))

  circles
    .attr('display', function(d) {
      if(xScale(d.x) < 0 || xScale(d.x) > width ||
        yScale(d.y) < 0 || yScale(d.y) > height) {
        return 'none';
      }
      return '';
    });
}
```