Information Visualization

Course Module IN6221

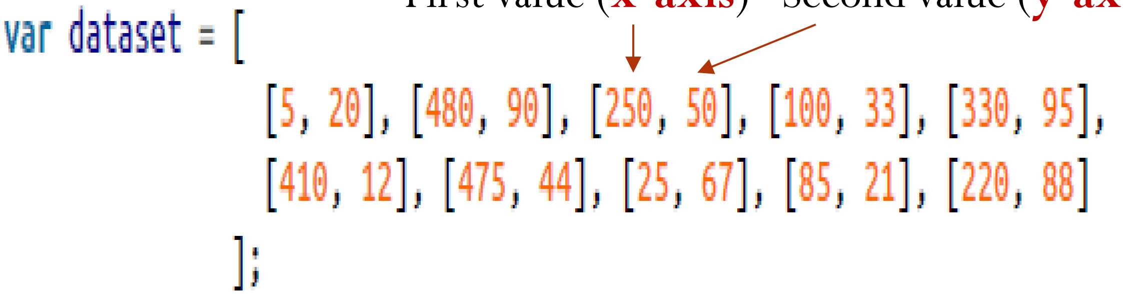
D3 Visualization Tool Part 2

WKW School of Communication and Information,
NTU

Setting Up Dynamic Scales - Recap

Dataset of an array of arrays.

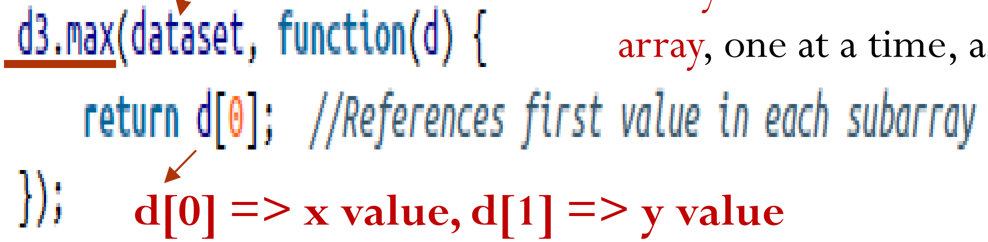
First value (**x-axis**) Second value (**y-axis**)



```
var dataset = [
  [5, 20], [480, 90], [250, 50], [100, 33], [330, 95],
  [410, 12], [475, 44], [25, 67], [85, 21], [220, 88]
];
```

- Map first value in each array onto **x-axis**, second value onto **y-axis.**
- **x values** range from 5 to 480 => instead of specifying input domain as [0, 500] use the functions **d3.min()** and **d3.max()** to analyze the dataset on the fly.

Reference array of values to be evaluated



```
d3.max(dataset, function(d) {
    return d[0]; //References first value in each subarray
});
```

anonymous function hands off each value in the data array, one at a time, as d (position [0]).

d[0] => x value, d[1] => y value

max() function **loops through** each value in the array, and **identifies the largest** (480).

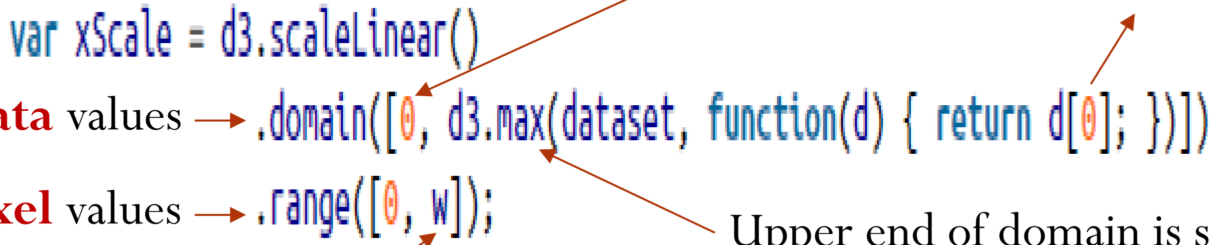
Setting Up Dynamic Scales

Alternatively, could use d3.**min()** to calculate a dynamic value.

References **d[0]**, **x value** of each dataset subarray

Input **data** values →

Output **pixel** values →

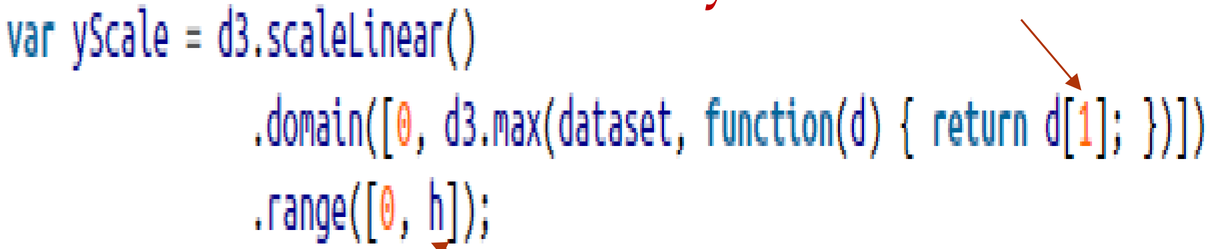


```
var xScale = d3.scaleLinear()
               .domain([0, d3.max(dataset, function(d) { return d[0]; })])
               .range([0, w]);
```

Upper end of domain is set to the **maximum** value in dataset

Output range is set to the SVG's **width**.

max() function references **d[1]**, **y value** of each dataset subarray



```
var yScale = d3.scaleLinear()
               .domain([0, d3.max(dataset, function(d) { return d[1]; })])
               .range([0, h]);
```

Upper end of y scale range() is set to **h** instead of w => range value is moving from top to bottom. (.range([**h, 0**]); to move from bottom to top)

Margin Convention

1) Define the **margin**

```
var margin = {top: 20, right: 10, bottom: 20, left: 10};
```

2) Define **width** and **height** as the **inner dimensions** of the chart area.

```
var width = 960 - margin.left - margin.right,
    height = 500 - margin.top - margin.bottom;
```

3) Define svg **g** element (**container used to group other SVG elements**) that translates origin to top-left corner of chart area

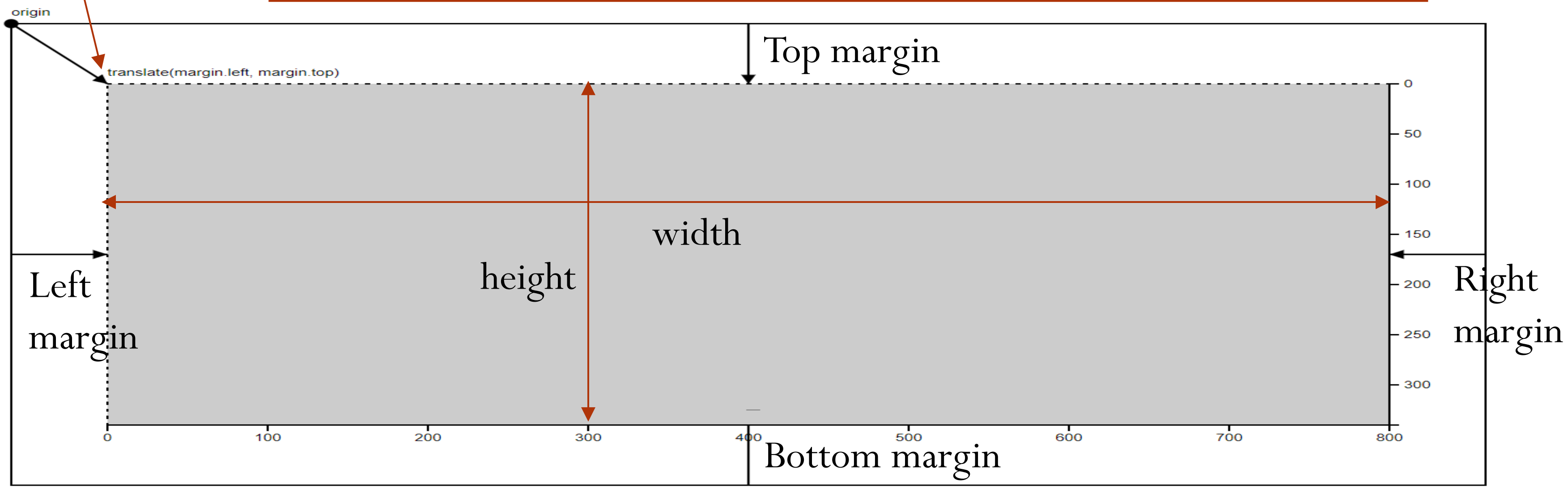
```
var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height)
    .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
```

Define svg

Define width

Define height

(relocate the svg)



Top margin

width

height

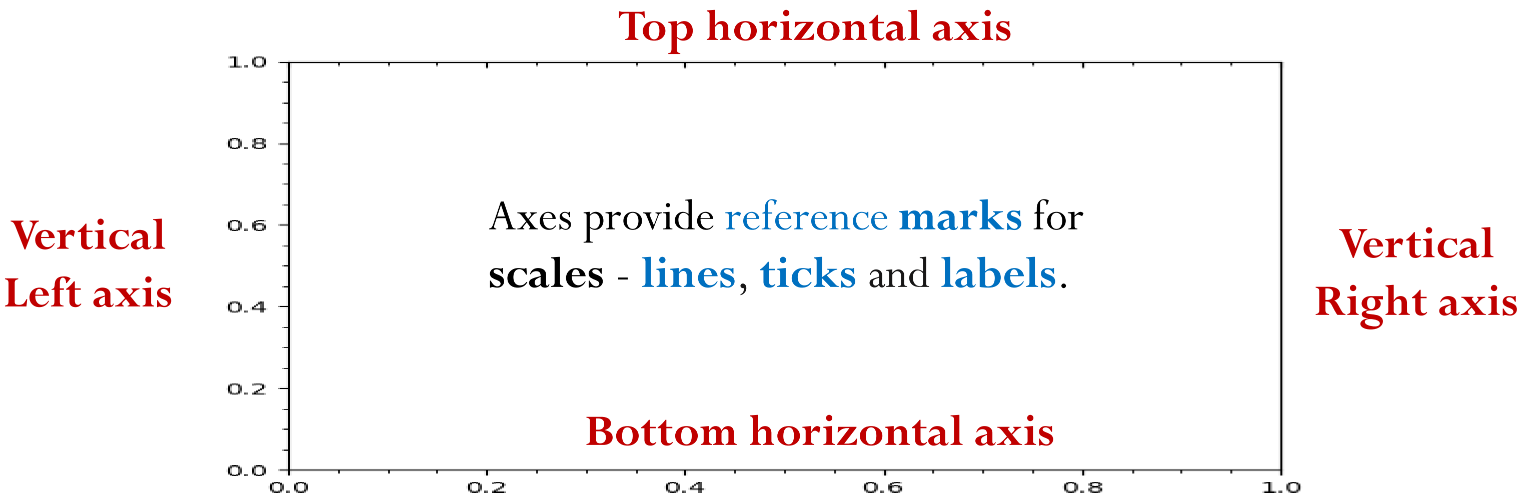
Left margin

Right margin

Bottom margin

D3 - Axes

- 2-D Graphs have **two axes**: the horizontal axis (x-axis) and vertical axis (y-axis). An **axis** uses **scale**, so each axis will need to be given a scale to work with.



Top horizontal axis

Vertical
Left axis

Axes provide reference **marks** for **scales** - **lines**, **ticks** and **labels**.

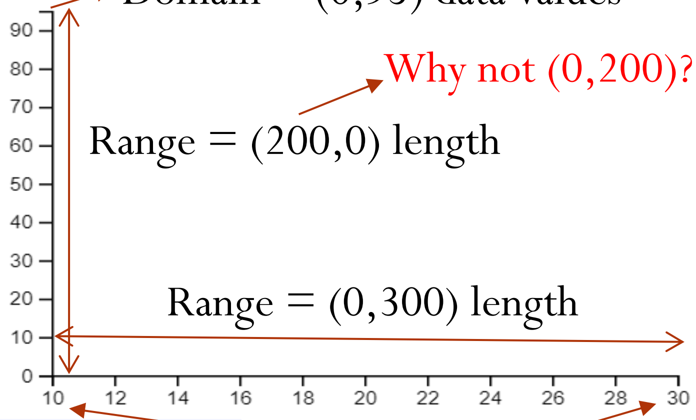
Vertical
Right axis

Bottom horizontal axis

Axis Methods used to create axes

d3.**axisTop()**	Creates top horizontal axis.
d3.**axisRight()**	Creates vertical right-oriented axis.
d3.**axisBottom()**	Creates bottom horizontal axis.
d3.**axisLeft()**	Creates left vertical axis.

D3 – X and Y Axes



Domain = (0,95) data values

Why not (0,200)?

Range = (200,0) length

Range = (0,300) length

Domain = (10,30) data values

```
<script>
    var width = 400, height = 400;
    var xdata = [10, 15, 20, 25, 30];
    var ydata = [10, 55, 70, 85, 95];

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

Create SVG element using D3 object

```
    var xscale = d3.scaleLinear()
        .domain([10, d3.max(xdata)])
        .range([0, width - 100]);
```

(10, 30)

(0, 300)

Create **x-axis** **Linear Scale**

Margin space 50 on each side

```
    var yscale = d3.scaleLinear()
            .domain([0, d3.max(ydata)])
            .range([height/2, 0]);
```

(0, 95)

(200, 0)

Create **y-axis** **Linear Scale**

```
    var x_axis = d3.axisBottom()
            .scale(xscale);
```

Create **bottom x-axis** with defined **x scale**

```
    var y_axis = d3.axisLeft()
            .scale(yscale);
```

Create **left y-axis** with defined **y scale**

```
        svg.append("g")
           .attr("transform", "translate(50, 10)")
           .call(y_axis);
```

Append **group element** - all components of **y-axis** will be grouped under the group element.

Apply translate transformation to align y-axis to **50px right** and **10px bottom of the origin** – margin gives better visual representation on screen.

Call **y_axis** function => **render y-axis**

```
    var xAxisTranslate = height/2 + 10;
```

$\underbrace{\text{height/2 + 10}}_{210}$

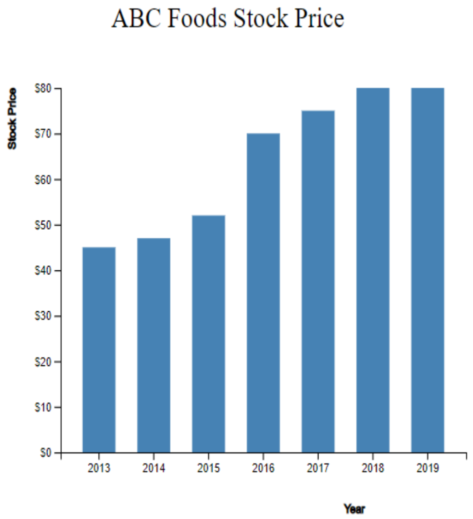
```
        svg.append("g")
                .attr("transform", "translate(50, " + xAxisTranslate  +")")
                .call(x_axis)
</script>
```

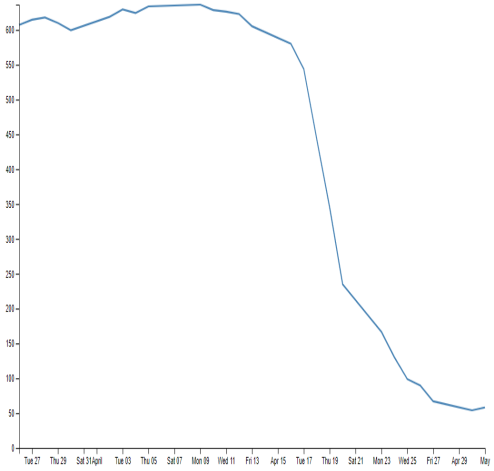
Move down **to y coordinate of x-axis**

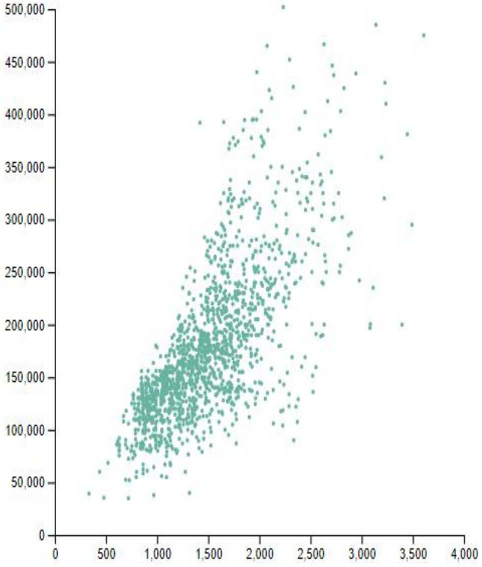
Apply translate transformation to align x-axis with start point of chart (**bottom left** x, y intersection)

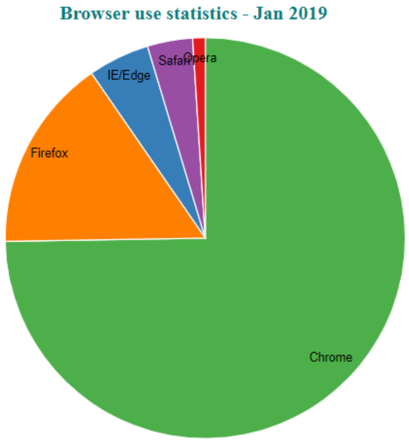
We are going to:

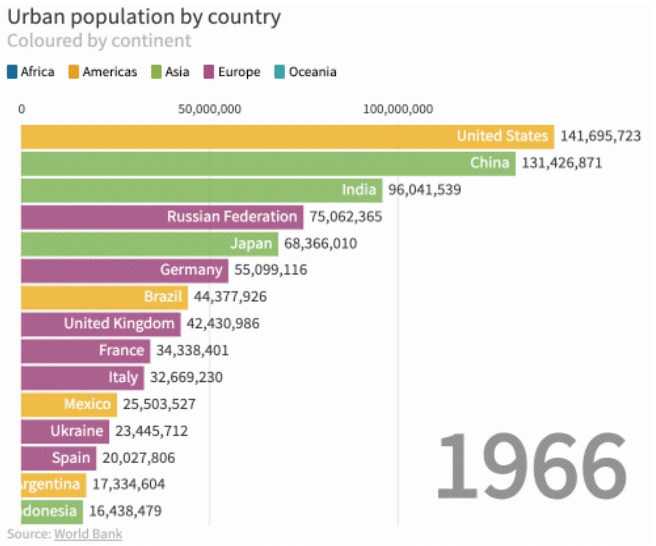
Create the Charts





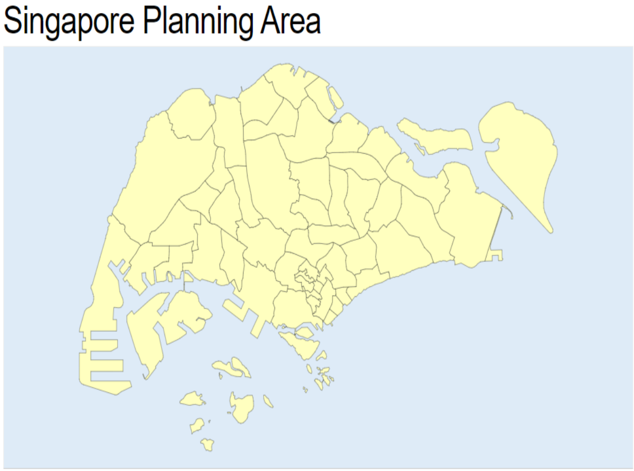






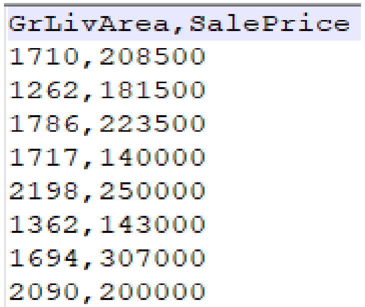
Create Animations

Draw Map Visualization



D3 - Scatterplot

Partial Dataset

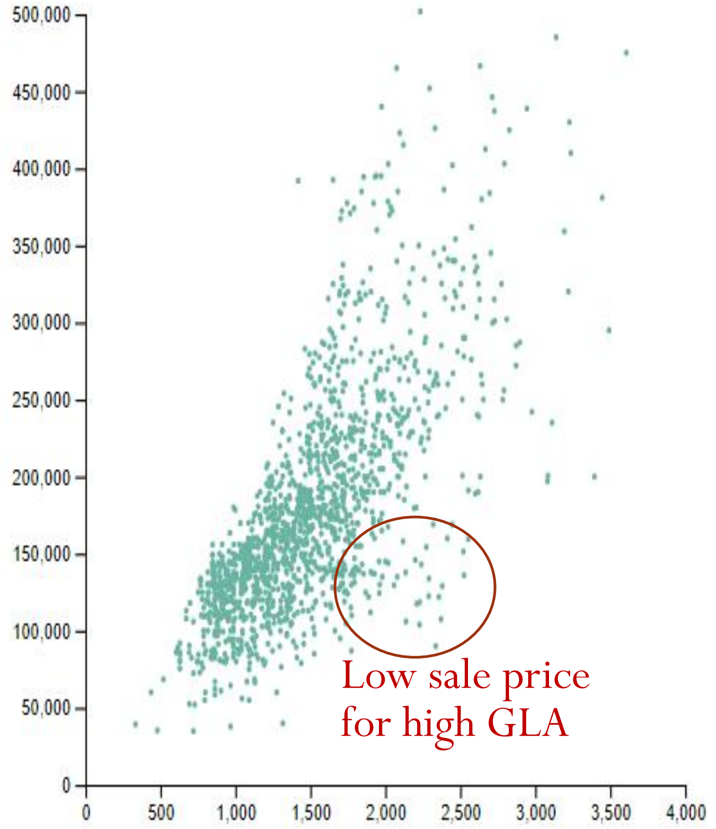


```
GrLivArea,SalePrice
1710,208500
1262,181500
1786,223500
1717,140000
2198,250000
1362,143000
1694,307000
2090,200000
```

Use Scatterplot when

1. There is **paired numerical** data.
2. When the **dependent** variable may have **multiple** values for each value of the **independent** variable.
3. When trying to **determine whether the two variables are related**, such as identify potential root causes of relationship => possible to have low sale price for high GLA

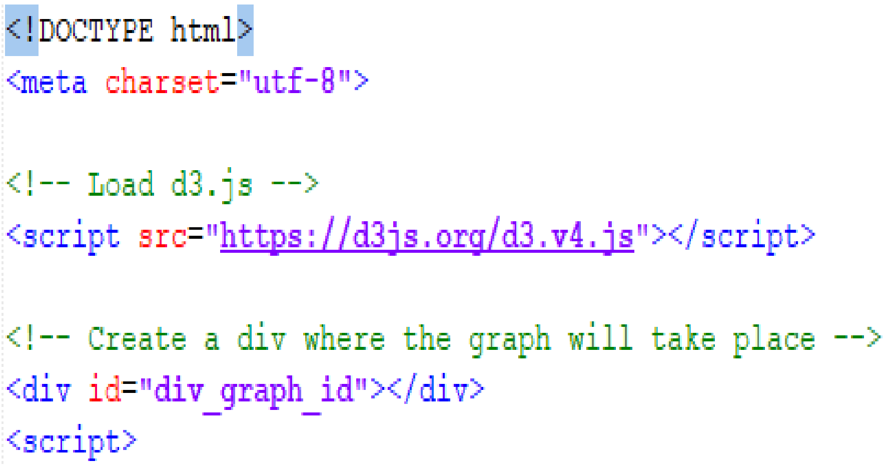
Sale Price
(dependent variable)



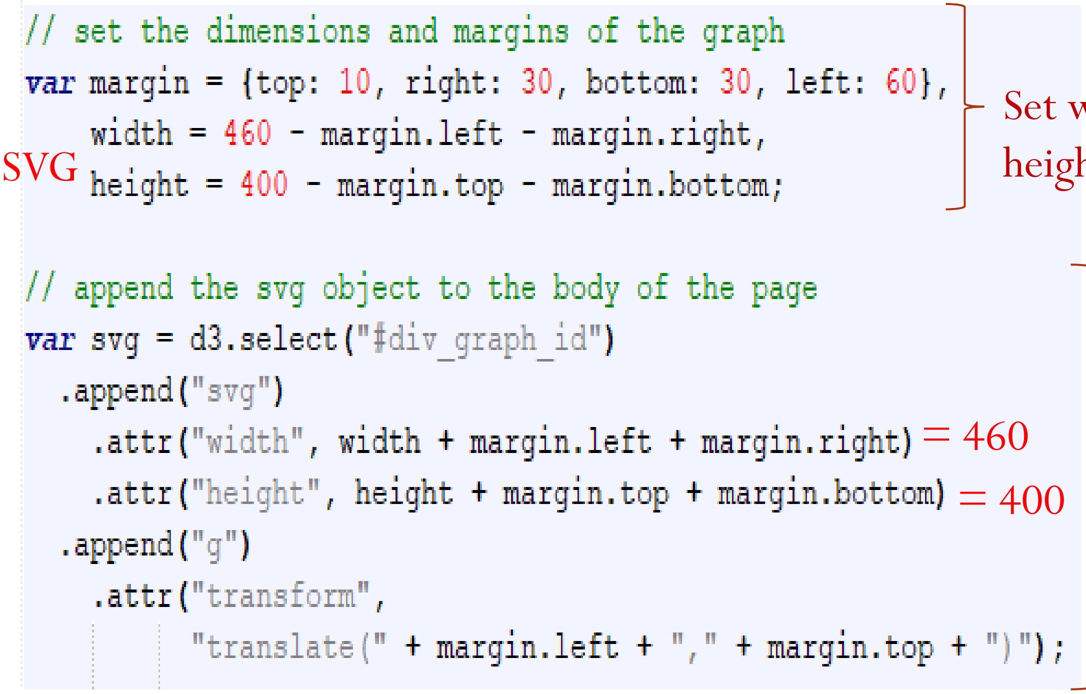
Low sale price for high GLA

Gross Living Area
(independent variable)

D3 - Scatterplot



```
<!DOCTYPE html>
<meta charset="utf-8">

<!-- Load d3.js -->
<script src="https://d3js.org/d3.v4.js"></script>

<!-- Create a div where the graph will take place -->
<div id="div_graph_id"></div>
<script>
```



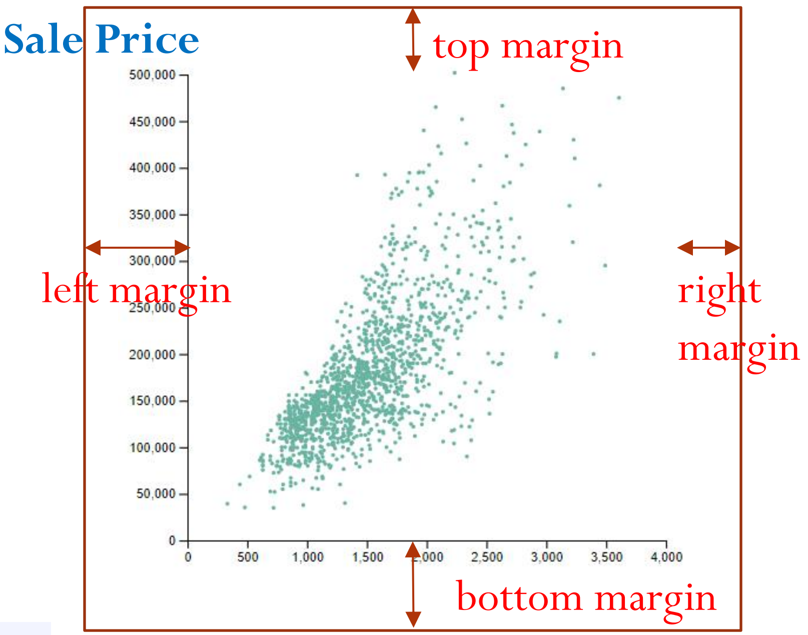
```
// set the dimensions and margins of the graph
var margin = {top: 10, right: 30, bottom: 30, left: 60},
    width = 460 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#div_graph_id")
  .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 + ")");
```

SVG

Set width and height for **graph**

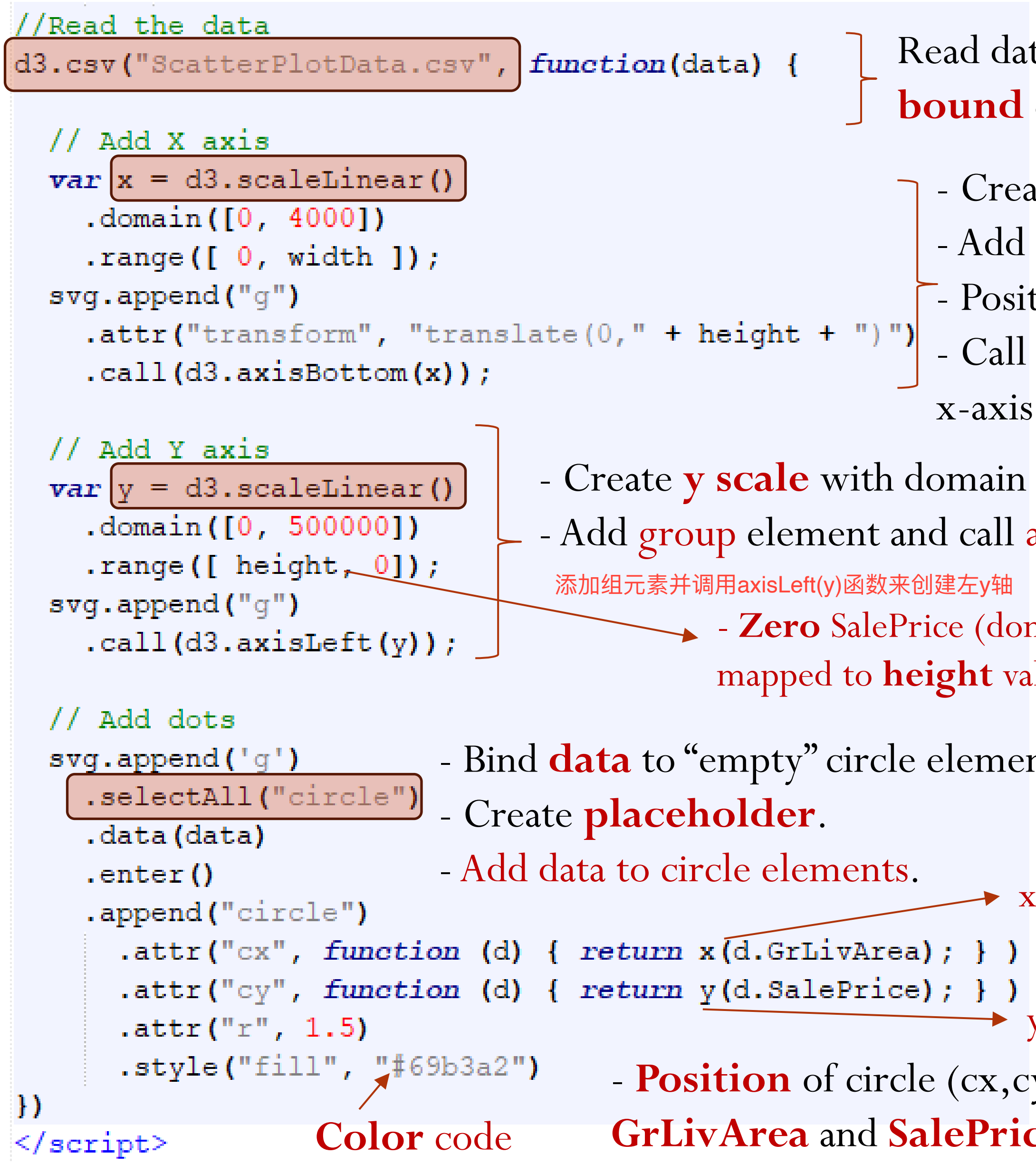
= 460

= 400



- Create **SVG** object for selected div with id #my_dataviz
- Set **width** and **height** attributes
- Add **group element g** to group child elements together
- **Transform** (move) SVG based on given **margins**

D3 - Scatterplot



```
//Read the data
d3.csv("ScatterPlotData.csv", function(data) {

  // Add X axis
  var x = d3.scaleLinear()
    .domain([0, 4000])
    .range([ 0, width ]);
  svg.append("g")
    .attr("transform", "translate(0," + height + ")")
    .call(d3.axisBottom(x));

  // Add Y axis
  var y = d3.scaleLinear()
    .domain([0, 500000])
    .range([ height, 0]);
  svg.append("g")
    .call(d3.axisLeft(y));

  // Add dots
  svg.append('g')
    .selectAll("circle")
    .data(data)
    .enter()
    .append("circle")
      .attr("cx", function (d) { return x(d.GrLivArea); } )
      .attr("cy", function (d) { return y(d.SalePrice); } )
      .attr("r", 1.5)
      .style("fill", "#69b3a2")
})
</script>
```

Read data of file – note to include **data bound** codes within **function(data)**

- Create **x scale** with domain and range values
- Add **group** element and include x-axis in group
- Position x-axis with translate
- Call **axisBottom(x)** function to create bottom x-axis

- Create **y scale** with domain and range values
- Add group element and call axisLeft(y) function to create left y-axis

添加组元素并调用axisLeft(y)函数来创建左y轴

- **Zero** SalePrice (domain) is mapped to **height** value (range)

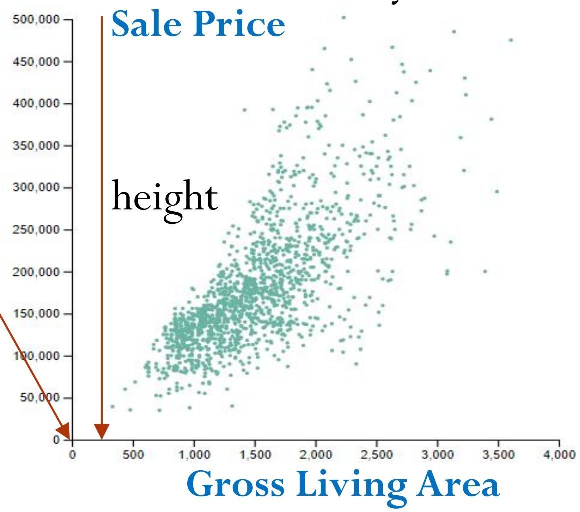
- Bind **data** to "empty" circle element.
- Create **placeholder**.
- Add data to circle elements.

x scale

y scale

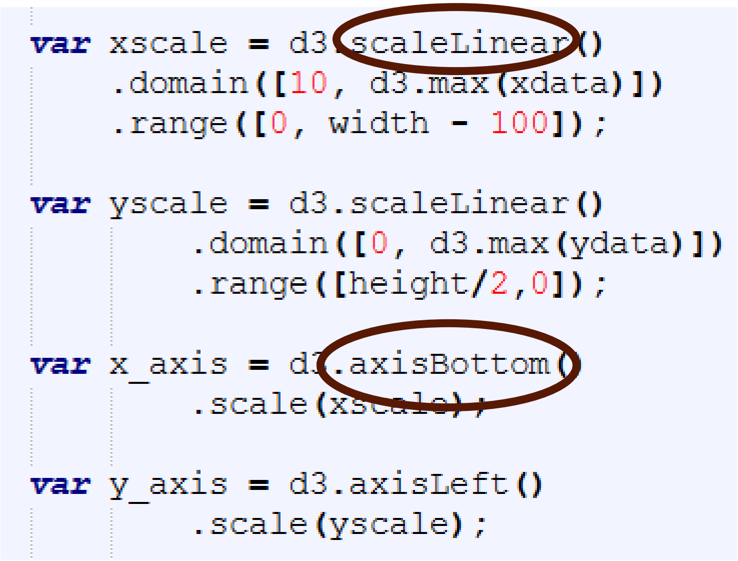
Color code

- **Position** of circle (cx,cy) determined by **GrLivArea** and **SalePrice** values in **data file**

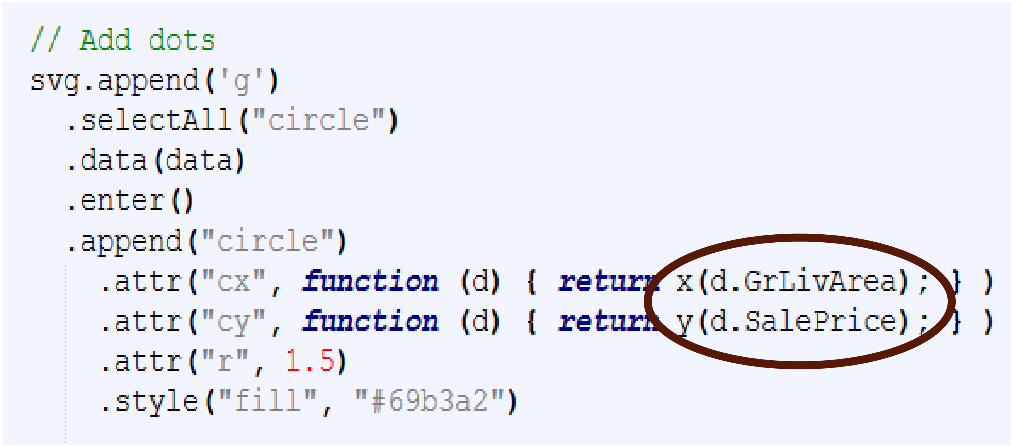


Class Exercise

- Open the following files and make changes to make the visualization work.
- 1_D3_Chart_1_Axes.html
- 1_D3_Chart_2_ScatterPlot.html



```
var xscale = d3.scaleLinear()
    .domain([10, d3.max(xdata)])
    .range([0, width - 100]);

var yscale = d3.scaleLinear()
        .domain([0, d3.max(ydata)])
        .range([height/2,0]);

var x_axis = d3.axisBottom()
        .scale(xscale);

var y_axis = d3.axisLeft()
        .scale(yscale);
```



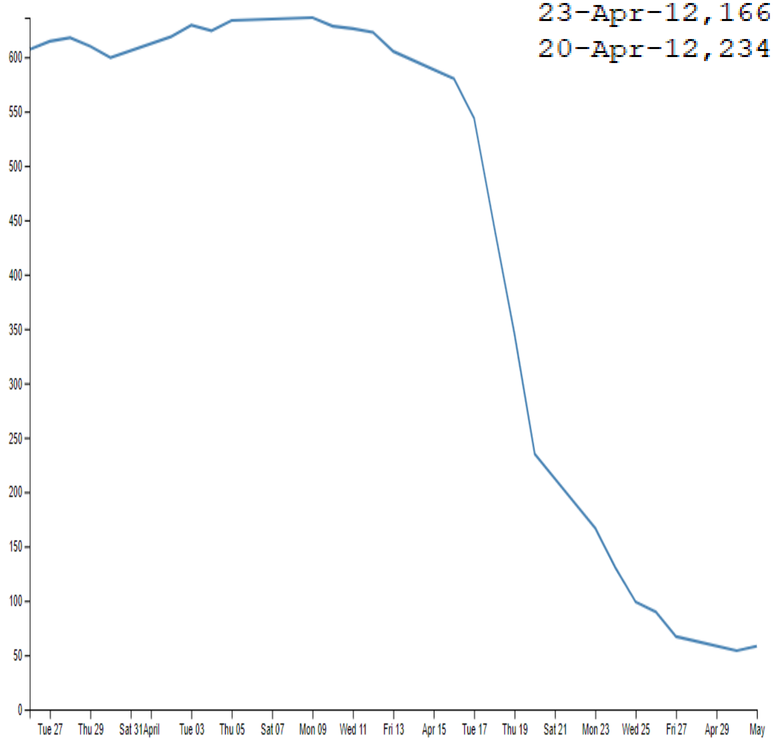
```
// Add dots
svg.append('g')
  .selectAll("circle")
  .data(data)
  .enter()
  .append("circle")
    .attr("cx", function (d) { return x(d.GrLivArea); } )
    .attr("cy", function (d) { return y(d.SalePrice); } )
    .attr("r", 1.5)
    .style("fill", "#69b3a2")
```

D3 – Line Chart

```
date,close
1-May-12,58.13
30-Apr-12,53.98
27-Apr-12,67.00
26-Apr-12,89.70
25-Apr-12,99.00
24-Apr-12,130.28
23-Apr-12,166.70
20-Apr-12,234.98
```

- **Line graphs** are used to **track changes** over short and long periods of time.
- When **smaller changes (differences)** exist, **line graphs** are better to **use** than bar **graphs**.
- Line graphs are used to compare changes over the same **period of time** for **more than one group**.

Stock Closing Price ($)



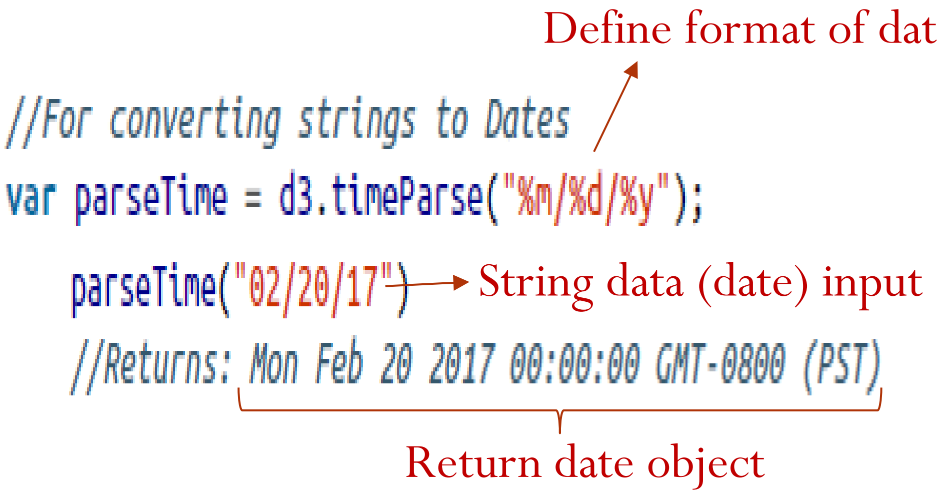
Date

Time Series

D3 – timeParse

- Frequent need to visualize data with a temporal (time) dimension.
- JavaScript and D3 can only perform **time and date calculations** on **Date objects**, not on strings. Working with dates in D3 involves:

1. Converting **strings to Date** objects
2. Using **time scales**, as needed
3. **Formatting Date objects** as human-friendly **strings**, for **display** to the user



Define format of data (date) input

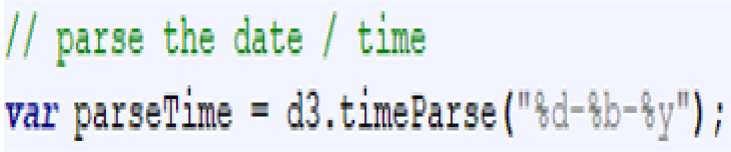
```
//For converting strings to Dates
var parseTime = d3.timeParse("%m/%d/%y");
    parseTime("02/20/17")
    //Returns: Mon Feb 20 2017 00:00:00 GMT-0800 (PST)
```

String data (date) input

Return date object

Look for three values, separated by slashes: **month** with leading zero, **day** of the month with leading zero, and two-digit **year** number.

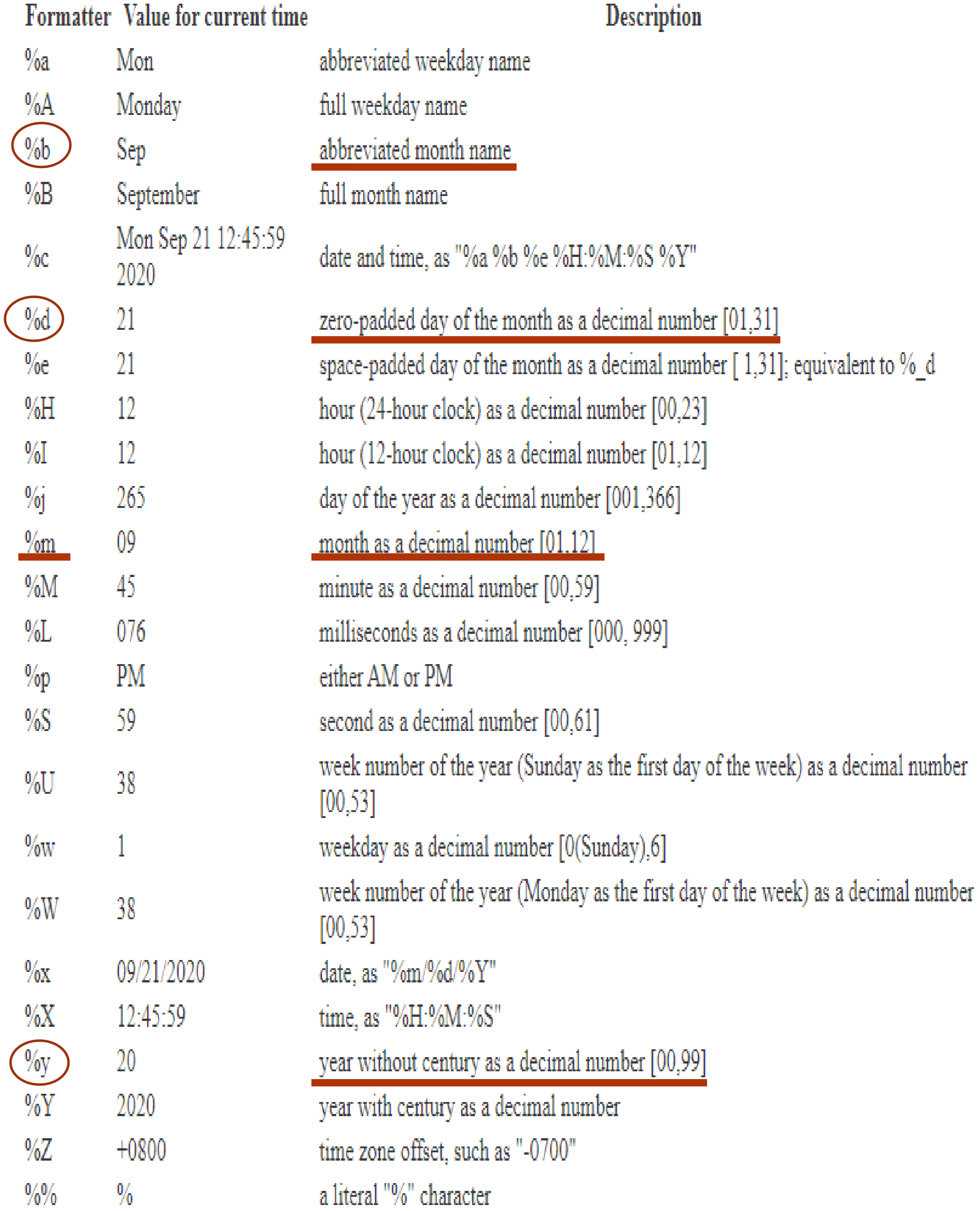
D3 – timeParse☆



```
// parse the date / time
var parseTime = d3.timeParse("%d-%b-%y");
```

e.g., timeParse("01-Jan-22")

Formatter Reference



Formatter	Value for current time	Description
%a	Mon	abbreviated weekday name
%A	Monday	full weekday name
%b	Sep	abbreviated month name
%B	September	full month name
%c	Mon Sep 21 12:45:59 2020	date and time, as "%a %b %e %H:%M:%S %Y"
%d	21	zero-padded day of the month as a decimal number [01,31]
%e	21	space-padded day of the month as a decimal number [1,31]; equivalent to %_d
%H	12	hour (24-hour clock) as a decimal number [00,23]
%I	12	hour (12-hour clock) as a decimal number [01,12]
%j	265	day of the year as a decimal number [001,366]
%m	09	month as a decimal number [01,12]
%M	45	minute as a decimal number [00,59]
%L	076	milliseconds as a decimal number [000, 999]
%p	PM	either AM or PM
%S	59	second as a decimal number [00,61]
%U	38	week number of the year (Sunday as the first day of the week) as a decimal number [00,53]
%w	1	weekday as a decimal number [0(Sunday),6]
%W	38	week number of the year (Monday as the first day of the week) as a decimal number [00,53]
%x	09/21/2020	date, as "%m/%d/%Y"
%X	12:45:59	time, as "%H:%M:%S"
%y	20	year without century as a decimal number [00,99]
%Y	2020	year with century as a decimal number
%Z	+0800	time zone offset, such as "-0700"
%%	%	a literal "%" character

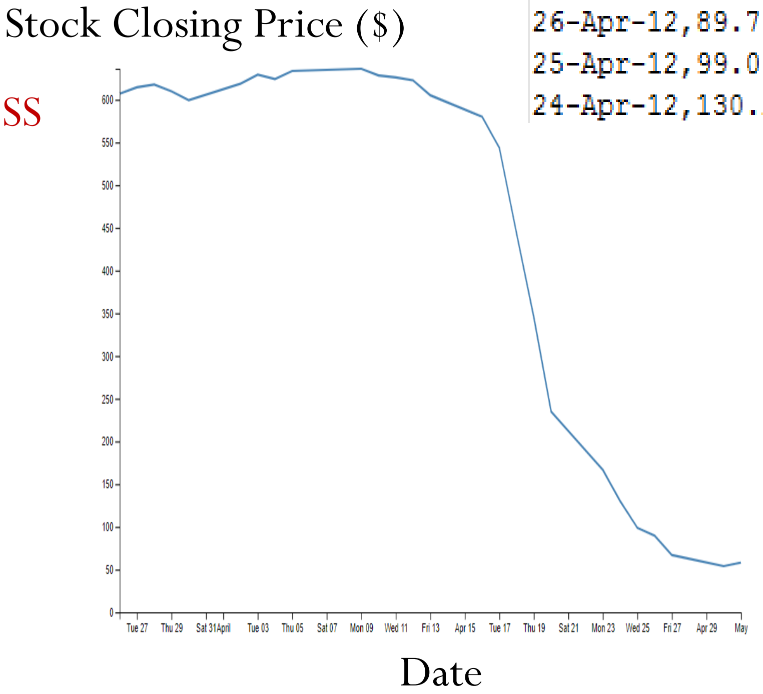
D3 – Line Chart (Time Scale)

```
date,close
1-May-12,58.13
30-Apr-12,53.98
27-Apr-12,67.00
26-Apr-12,89.70
25-Apr-12,99.00
24-Apr-12,130.28
```

```
<style> /* set the CSS */
.line {
  fill: none;
  stroke: steelblue;
  stroke-width: 2px;
}
</style>
```

Define **line class** style in CSS

```
<body>
<!-- load the d3.js library -->
<script src="https://d3js.org/d3.v4.min.js"></script>
<script>
```



```
// set the dimensions and margins of the graph
var margin = {top: 20, right: 20, bottom: 30, left: 50},
    width = 960 - margin.left - margin.right,
    height = 500 - margin.top - margin.bottom;

// parse the date / time
var parseTime = d3.timeParse("%d-%b-%y");
```

Define **format** day-mth-year of **time data** in data file

```
// set the ranges
var x = d3.scaleTime().range([0, width]);
var y = d3.scaleLinear().range([height, 0]);
```

Define **x-axis time scale** and **y-axis linear scale range** values – domain values defined later

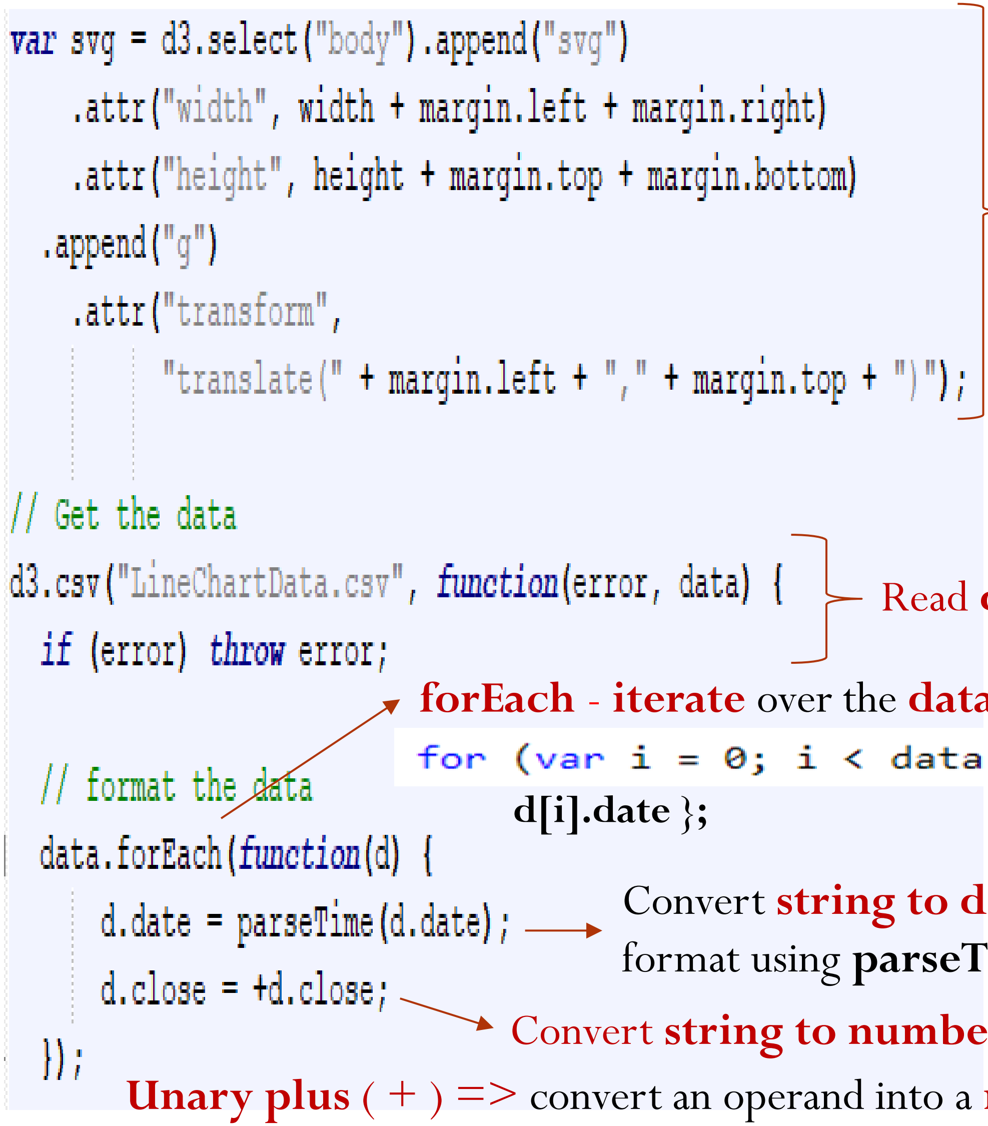
```
// Scale the range of the data
x.domain(d3.extent(data, function(d) { return d.date; }));
y.domain([0, d3.max(data, function(d) { return d.close; })]);
```

extent() - returns **[min, max]** of (**date**) data values.

Assign **domain** values for **defined x, y scales**

max() - returns **max** of (**close**) data values.

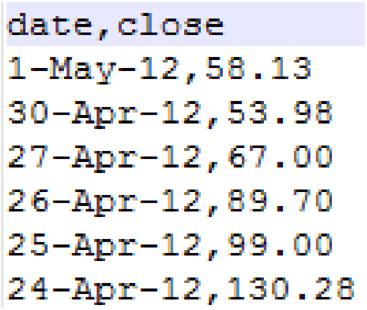
D3 – Line Chart (Time Scale)



```
var svg = d3.select("body").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 + ")");

// Get the data
d3.csv("LineChartData.csv", function(error, data) {
  if (error) throw error;

  // format the data
  data.forEach(function(d) {
      d.date = parseTime(d.date);
      d.close = +d.close;
  });
```

- Create **SVG object** with width and height
- Create **group** element
- **Translate** to **position** based on margins

Read **data** from CSV file



```
date,close
1-May-12,58.13
30-Apr-12,53.98
27-Apr-12,67.00
26-Apr-12,89.70
25-Apr-12,99.00
24-Apr-12,130.28
```

forEach - **iterate** over the **data array** - same as

```
for (var i = 0; i < data.length; i++) {
```

d[i].date };

Convert **string to date** format using **parseTime**

Why do we need to convert strings to date?

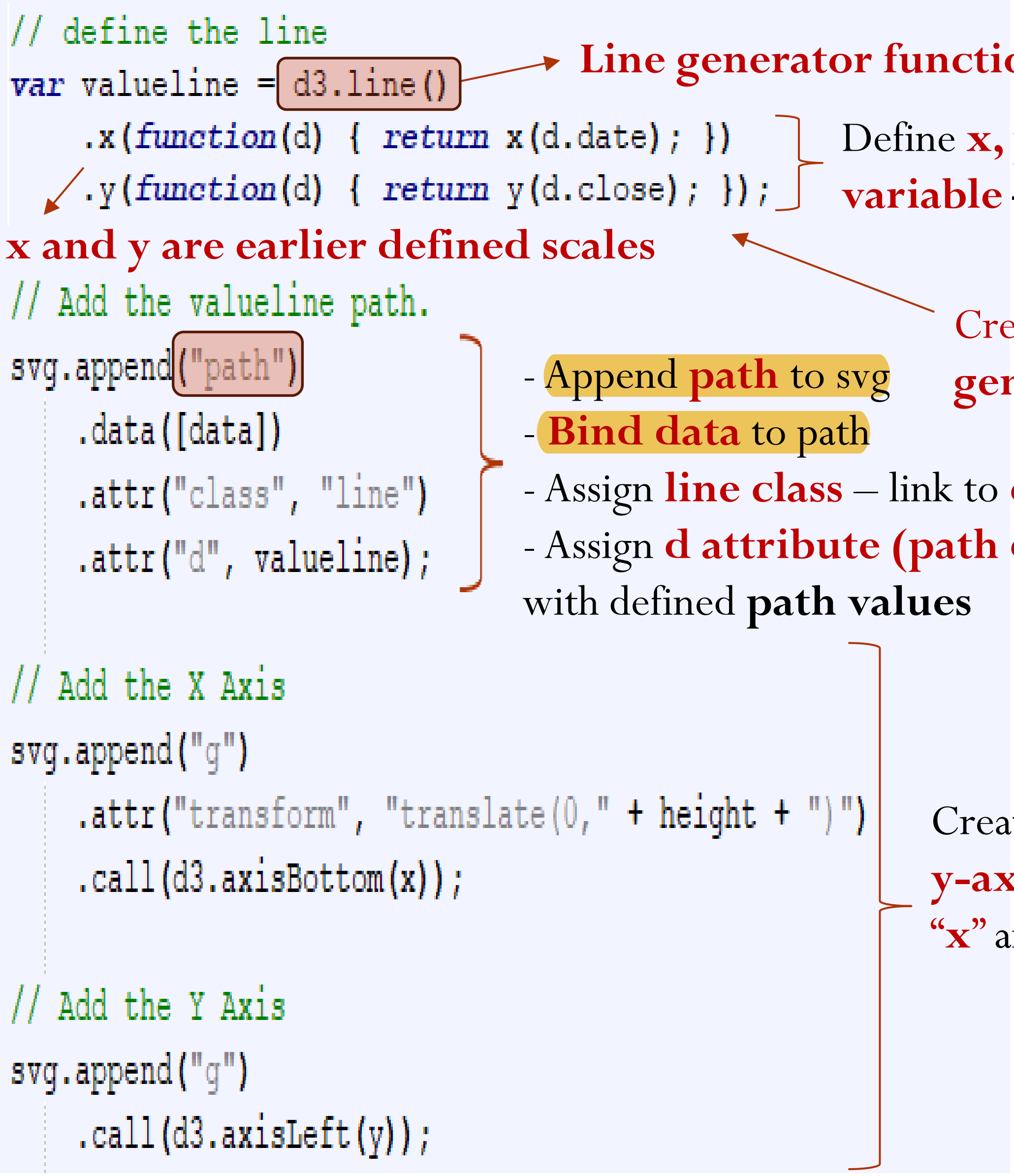
Convert **string to number**

Unary plus (+) => convert an operand into a **number**.

Unary minus (-) => convert an operand into a number and **negate** the value after that.

将操作数转换为数字，然后取 相反的值，即5变-5，-10变10

D3 – Line Chart (Time Scale)



```
// define the line
var valueline = d3.line()
    .x(function(d) { return x(d.date); })
    .y(function(d) { return y(d.close); });
// Add the valueline path.
svg.append("path")
    .data([data])
    .attr("class", "line")
    .attr("d", valueline);

// Add the X Axis
svg.append("g")
    .attr("transform", "translate(0," + height + ")")
    .call(d3.axisBottom(x));

// Add the Y Axis
svg.append("g")
    .call(d3.axisLeft(y));
```

Line generator function – assign **line object** to variable

Define **x, y points** of line (path) based on **data variable –** used later in **Path** for line generation.

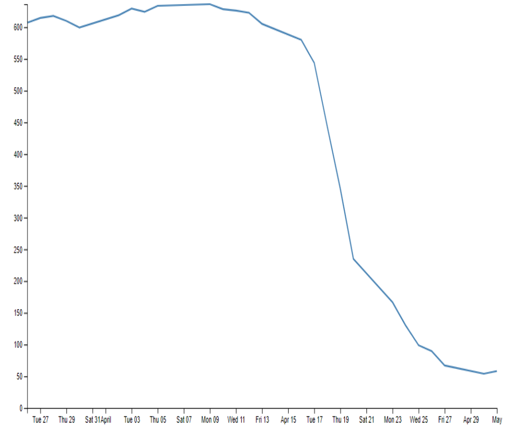
x and y are earlier defined scales

Created from **.line() generator** function

Draw Line

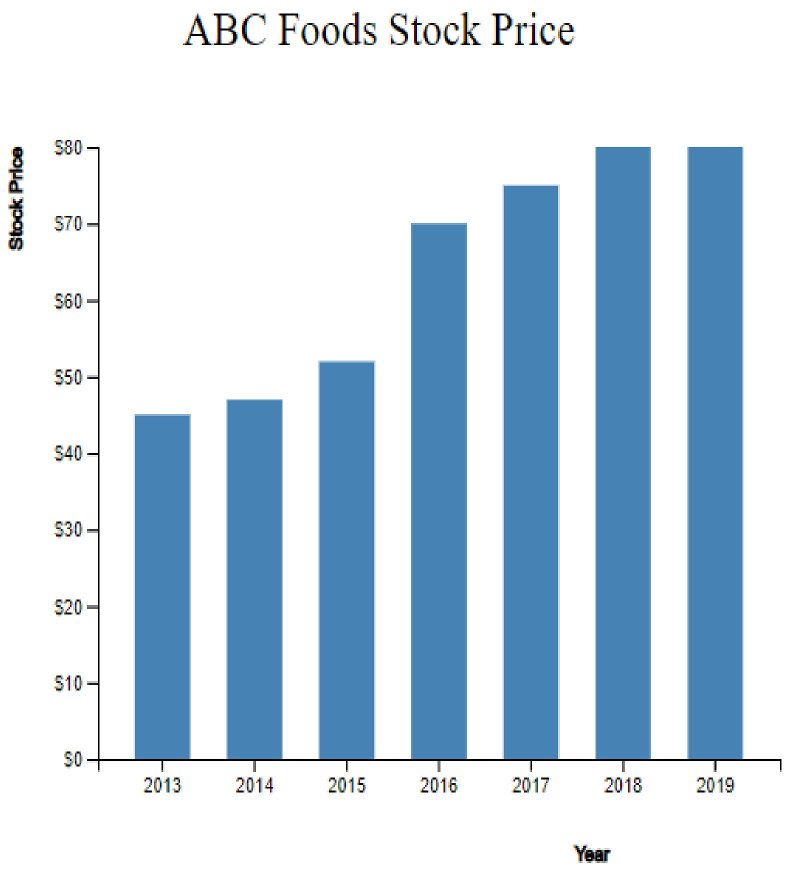
- Append **path** to svg
- **Bind data** to path
- Assign **line class** – link to **css style**
- Assign **d attribute (path definition)** to **line** object **valueline** with defined **path values**

Create **x-axis** and **y-axis** with defined **"x"** and **"y" scales**.



D3 – Bar Chart

- Used for presenting **comparative data** through a chart.
- Bar chart commonly used as it is **easy to interpret**.
- Useful for displaying data that is classified into **nominal** or **ordinal categories**.



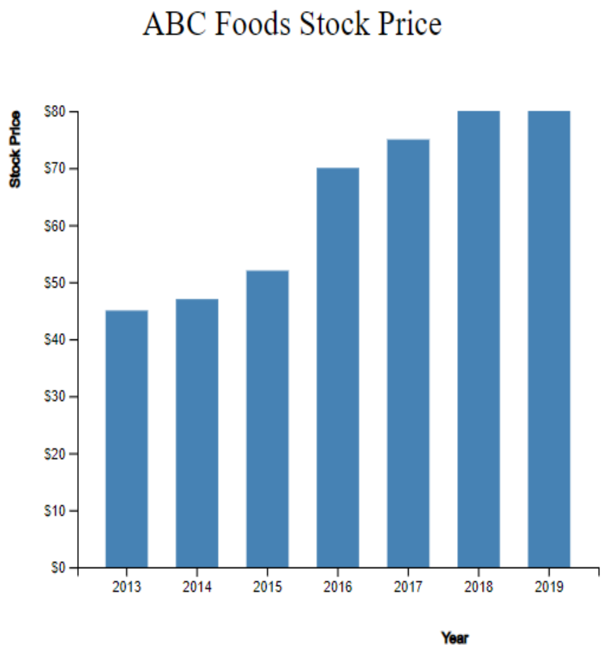
D3 –Bar Chart

```
<body>
<svg width="600" height="500"></svg>
<script>
```

DATA

```
year,value
2011,45
2012,47
2013,52
2014,70
2015,75
2016,78
```

ABC Foods Stock Price



```
    var svg = d3.select("svg"),
        margin = 200,
        width = svg.attr("width") - margin,
        height = svg.attr("height") - margin
```

Chart width and height

Create **SVG object**

```
    svg.append("text")
       .attr("transform", "translate(100,0)")
       .attr("x", 50)
       .attr("y", 50)
       .attr("font-size", "24px")
       .text("ABC Foods Stock Price")
```

transform,是用来提供偏移量的属性

Create **Title text**

Range => Chart width

```
    var xScale = d3.scaleBand().range([0, width]).padding(0.4),
        yScale = d3.scaleLinear().range([height, 0]);
```

scaleBand() – create scale for data with discrete bands. Space between bars – 40% of assigned band width

Create **X and Y scales** – define **range** values

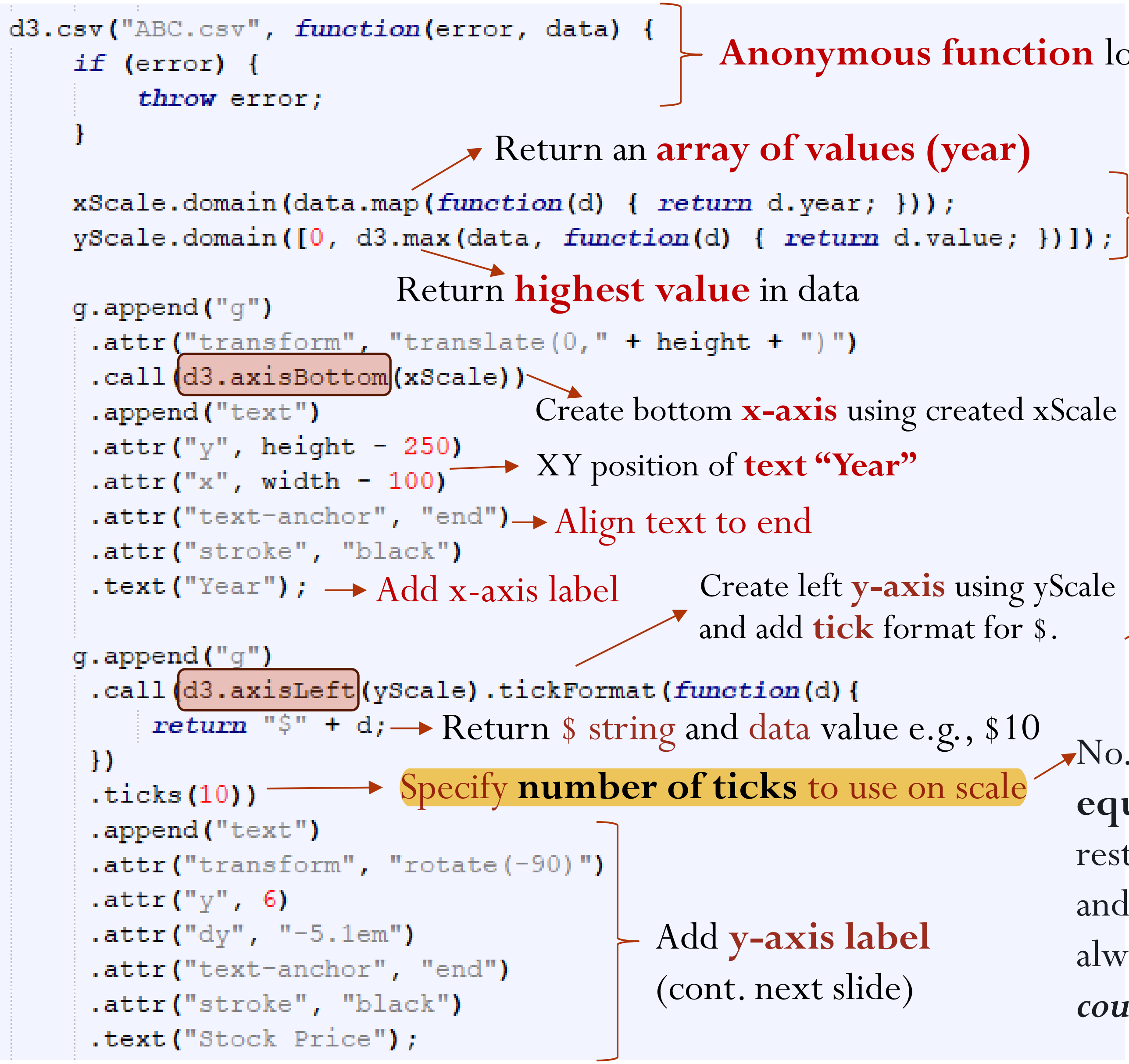
Specified as Chart height

```
    var g = svg.append("g")
               .attr("transform", "translate(" + 100 + "," + 100 + ")");
```

Append **group element** - for adding **axes** and **bars** to the group element

Position graph with top and left **margin** of 100

D3 – Bar Chart – Create Axes



```
d3.csv("ABC.csv", function(error, data) {
    if (error) {
        throw error;
    }

    xScale.domain(data.map(function(d) { return d.year; }));
    yScale.domain([0, d3.max(data, function(d) { return d.value; })]);

    g.append("g")
     .attr("transform", "translate(0," + height + ")")
     .call(d3.axisBottom(xScale))
     .append("text")
     .attr("y", height - 250)
     .attr("x", width - 100)
     .attr("text-anchor", "end")
     .attr("stroke", "black")
     .text("Year");

    g.append("g")
     .call(d3.axisLeft(yScale).tickFormat(function(d){
         return "$" + d;
     })
     .ticks(10))
     .append("text")
     .attr("transform", "rotate(-90)")
     .attr("y", 6)
     .attr("dy", "-5.1em")
     .attr("text-anchor", "end")
     .attr("stroke", "black")
     .text("Stock Price");
```

Anonymous function loads records to **data** keyword

Return an **array of values (year)**

Define Scale **Domain** Values

Return **highest value** in data

Create bottom **x-axis** using created xScale

XY position of **text "Year"**

Align text to end

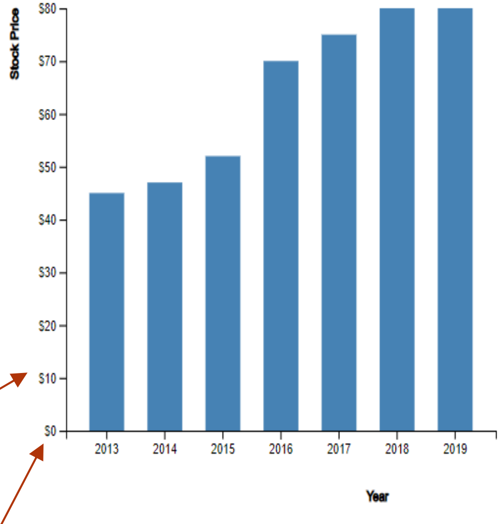
Add x-axis label

Create left **y-axis** using yScale and add **tick** format for $.

Return $ string and data value e.g., $10

Specify **number of ticks** to use on scale

Add **y-axis label** (cont. next slide)



No. of ticks returned **may not equal requested count**. Ticks are restricted to nicely-rounded values and the scale's domain may not always be subdivided in **exact *count*** of such intervals.

D3 – Bar Chart – Create Bar

```
g.append("g")
 .call(d3.axisLeft(yScale).tickFormat(function(d){
     return "$" + d;
 })
 .ticks(10))
 .append("text")
 .attr("transform", "rotate(-90)")
 .attr("y", 6)
 .attr("dy", "-5.1em")
 .attr("text-anchor", "end")
 .attr("stroke", "black")
 .text("Stock Price");
```

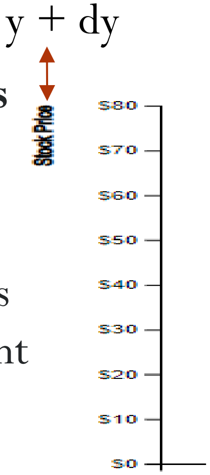
Codes repeated for explanation

Add **y-axis label**

dx and **dy** are **relative coordinates** (relative to the specified **x** and **y**).

(“dy”, “-5.1em”) offsets the text element from **y** a distance of 5.1 times **smaller** (neg) than font size of element => specify **dynamic** (based on font size) **margin** from y

（“Dy”、“-5.1em”）将文本元素从y偏移，距离比元素的字体大小小5.1倍（neg）=>指定动态（基于字体大小）与y的边距

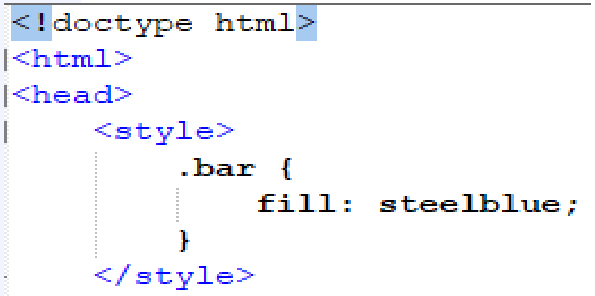


```
g.selectAll(".bar")
 .data(data)
 .enter().append("rect")
 .attr("class", "bar")
 .attr("x", function(d) { return xScale(d.year); })
 .attr("y", function(d) { return yScale(d.value); })
 .attr("width", xScale.bandwidth())
 .attr("height", function(d) { return height - yScale(d.value); });
```

Reference bar class using period (.)

Placeholder

bar class prior defined at CSS style



```
<!doctype html>
<html>
<head>
    <style>
        .bar {
            fill: steelblue;
        }
    </style>
```

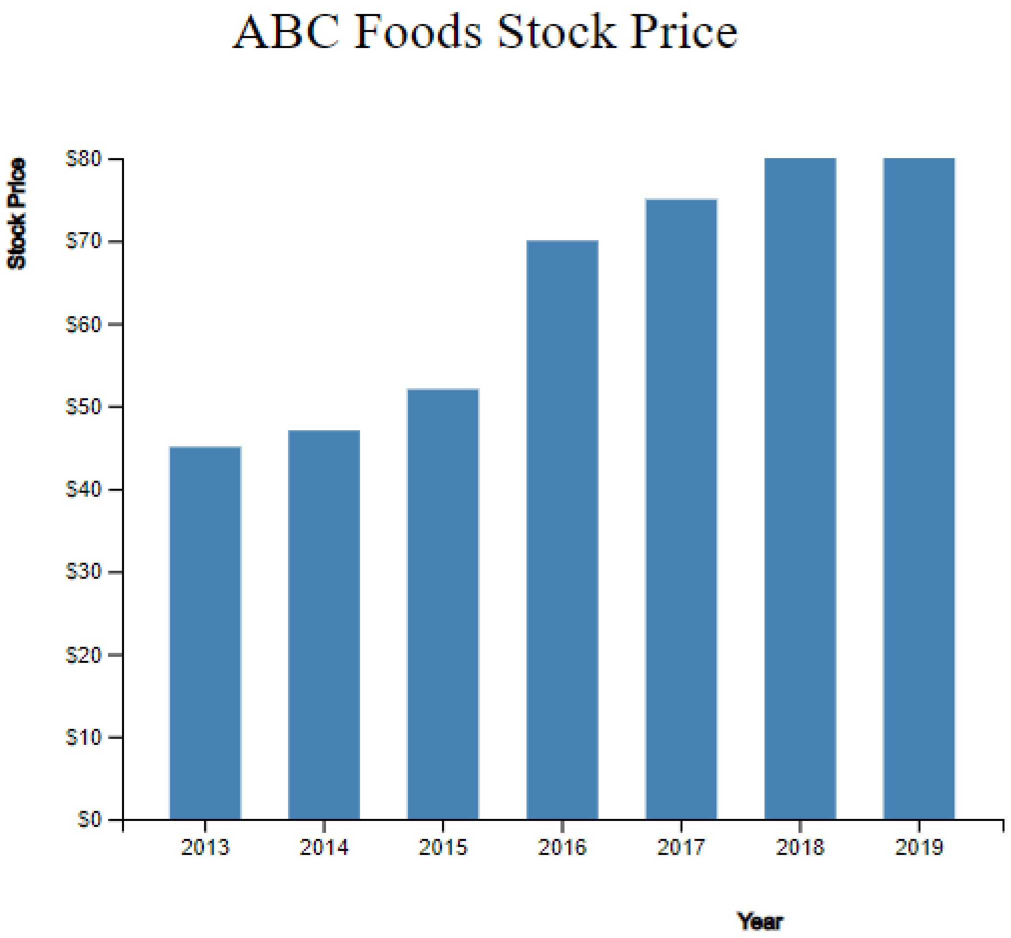
Assign bar class to each rect

Return **x** and **y range** value of each **rect** based on **data** values

Assign rect width based on **width** of **each band** output from xScale (**scaleBand**)

Assign rect height => (500 – data value) measured **downwards from origin**

D3 – Bar Chart

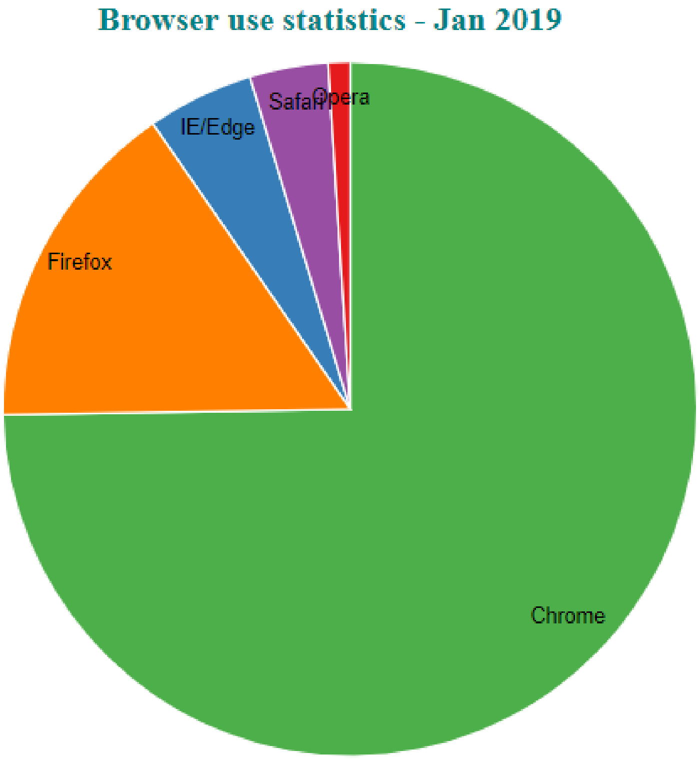


Overview

1) Create **SVG**
2) Create Title
3) Load and bind **data**
4) Create **Scales** – define range and domain values
5) Create **Group** Element – group axes and bars
6) Create **axes** – add **labels** and **ticks**
7) Create **bars** – bind data to rect element with each bar attributes based on scale range values

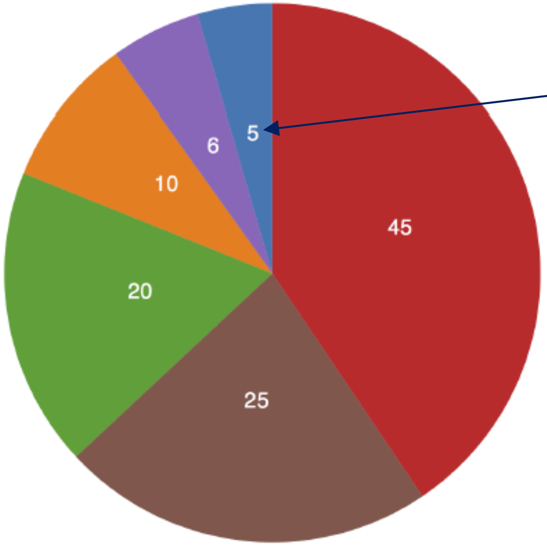
D3 – Pie Chart

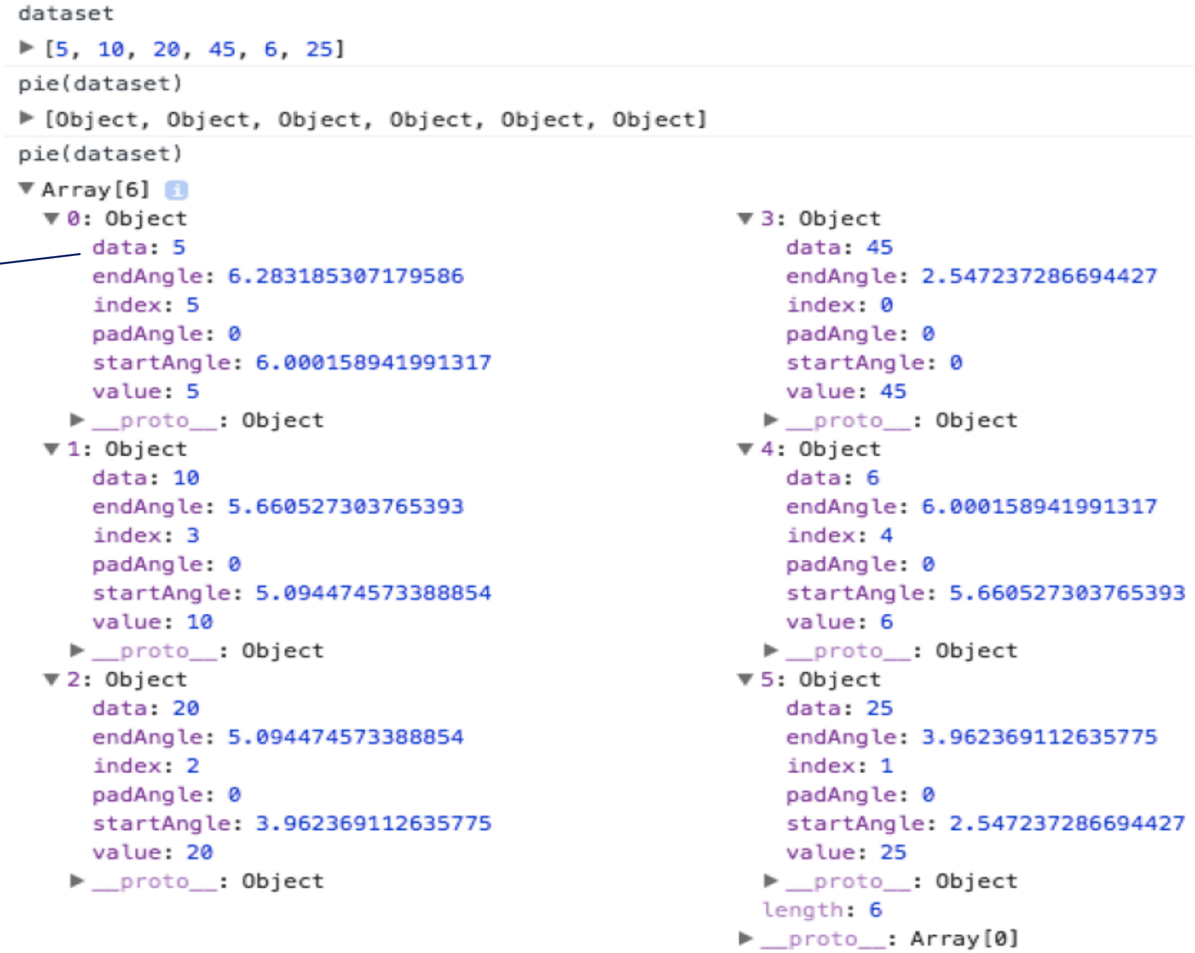
- **Pie Chart**- used for **comparing fractions of a whole**, to **show relative sizes**, and **precision** isn't required.
- A pie chart typically represents numbers in percentages, used to **visualize a part to whole** relationship or a composition.
- Pie charts are **not meant to compare individual sections to each other** or to represent exact values (use a bar chart for that).



Pie Chart

- The **pie(dataset)** function takes the array of numbers (dataset) and generates an **array of objects**, each object with values – notably the **startAngle** and **endAngle** values.



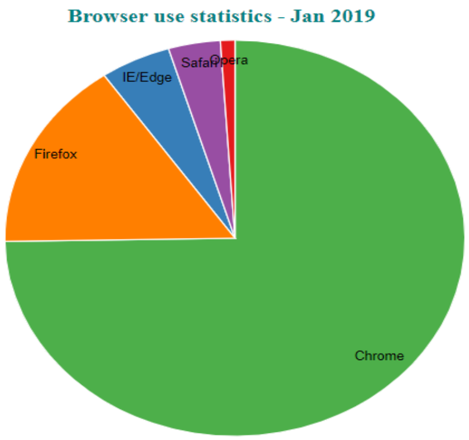


```
dataset
▶ [5, 10, 20, 45, 6, 25]
pie(dataset)
▶ [Object, Object, Object, Object, Object, Object]
pie(dataset)
▼ Array[6]
  ▼ 0: Object
      data: 5
      endAngle: 6.283185307179586
      index: 5
      padAngle: 0
      startAngle: 6.000158941991317
      value: 5
    ▶ __proto__: Object
  ▼ 1: Object
      data: 10
      endAngle: 5.660527303765393
      index: 3
      padAngle: 0
      startAngle: 5.094474573388854
      value: 10
    ▶ __proto__: Object
  ▼ 2: Object
      data: 20
      endAngle: 5.094474573388854
      index: 2
      padAngle: 0
      startAngle: 3.962369112635775
      value: 20
    ▶ __proto__: Object
  ▼ 3: Object
      data: 45
      endAngle: 2.547237286694427
      index: 0
      padAngle: 0
      startAngle: 0
      value: 45
    ▶ __proto__: Object
  ▼ 4: Object
      data: 6
      endAngle: 6.000158941991317
      index: 4
      padAngle: 0
      startAngle: 5.660527303765393
      value: 6
    ▶ __proto__: Object
  ▼ 5: Object
      data: 25
      endAngle: 3.962369112635775
      index: 1
      padAngle: 0
      startAngle: 2.547237286694427
      value: 25
    ▶ __proto__: Object
    length: 6
  ▶ __proto__: Array[0]
```

D3 – Pie Chart

Dataset

```
browser,percent
Chrome,73.70
IE/Edge,4.90
Firefox,15.40
Safari,3.60
Opera,1.00
```



```
<body>
    <svg width="500" height="400"></svg>
    <script>

        var svg = d3.select("svg"),
            width = svg.attr("width"),
            height = svg.attr("height"),
            radius = Math.min(width, height) / 2;

        var g = svg.append("g")
                   .attr("transform", "translate(" + width / 2 + "," + height / 2 + ")");

        var color = d3.scaleOrdinal(['#4daf4a','#377eb8','#ff7f00','#984ea3','#e41a1c']);

        var pie = d3.pie().value(function(d) {
                return d.percent;
            });

        var path = d3.arc()
                     .outerRadius(radius - 10)
                     .innerRadius(0);

        var label = d3.arc()
                      .outerRadius(radius)
                      .innerRadius(radius - 80);
```

Define **svg** element

Radius used for drawing arc – based on min of width or height

javaScript math function – finds min of the width and height values and divide by 2

Move to **center** of svg element

Ordinal Color scale of 5 colors **range**

pie() takes **values** in **dataset** and generates an **array** of **objects** (the **parts**) each with a **startAngle** and an **endAngle** value.

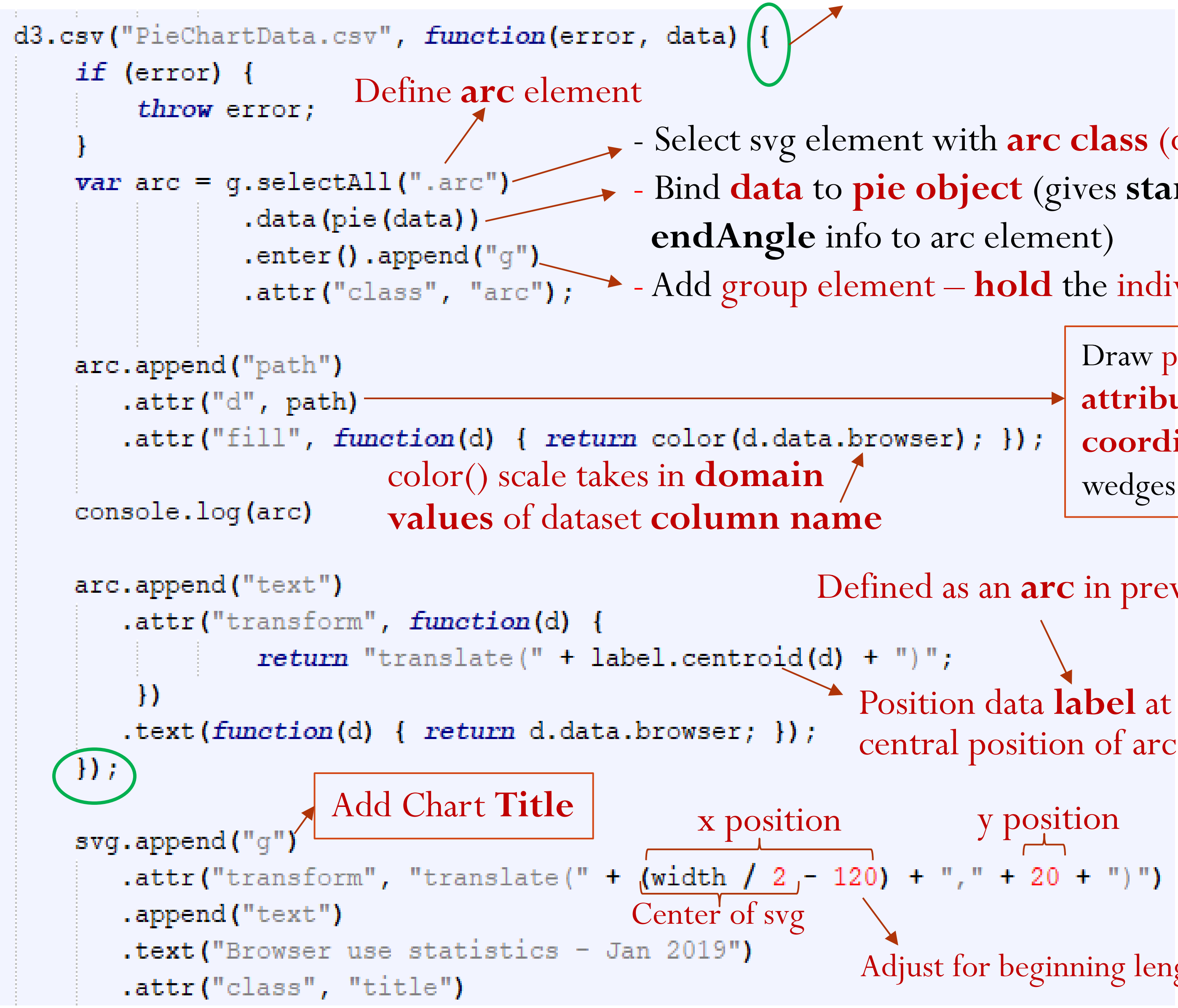
pie（）在数据集中取值，并生成一个具有startAngle和endAngle值的对象（部分）数组。

Arc() – draws the **path** of the pie's wedges using an inner radius and outer radius. If inner **radius = 0**, the result will be a **piechart**, otherwise a **donut** chart

Arc（）-使用内半径和外半径绘制饼楔形的路径。如果内半径=0，结果将是piechart，否则将是甜甜圈图

Define arc for **label** => place at **centroid of arc** late

D3 – Pie Chart



```
d3.csv("PieChartData.csv", function(error, data) {
    if (error) {
        throw error;
    }
    var arc = g.selectAll(".arc")
               .data(pie(data))
               .enter().append("g")
               .attr("class", "arc");

    arc.append("path")
       .attr("d", path)
       .attr("fill", function(d) { return color(d.data.browser); });

    console.log(arc)

    arc.append("text")
       .attr("transform", function(d) {
                return "translate(" + label.centroid(d) + ")";
        })
       .text(function(d) { return d.data.browser; });
    });

    svg.append("g")
       .attr("transform", "translate(" + (width / 2 - 120) + "," + 20 + ")")
       .append("text")
       .text("Browser use statistics - Jan 2019")
       .attr("class", "title")
```

Read and bind **data** within callback function

Define **arc** element

- Select svg element with **arc class** (defined style in codes)
- Bind **data** to **pie object** (gives **startAngle** and **endAngle** info to arc element)
- Add group element – **hold** the individual pie wedges

Draw pie chart using **path d attribute** – define **successive coordinates** for each of the pie wedges in defined **arc** object

color() scale takes in **domain values** of dataset **column name**

Defined as an **arc** in prev slide

Position data **label** at central position of arc

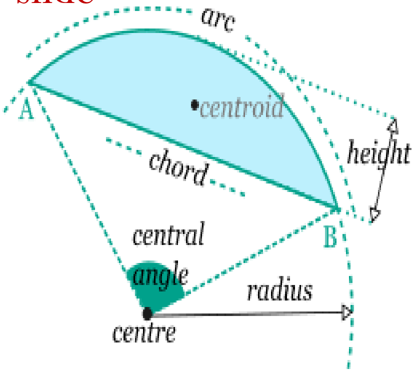
Add Chart **Title**

x position

y position

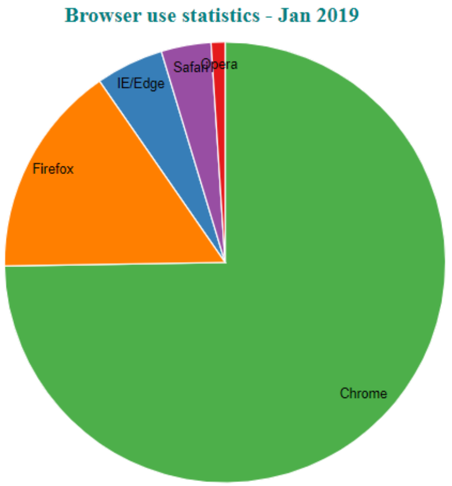
Center of svg

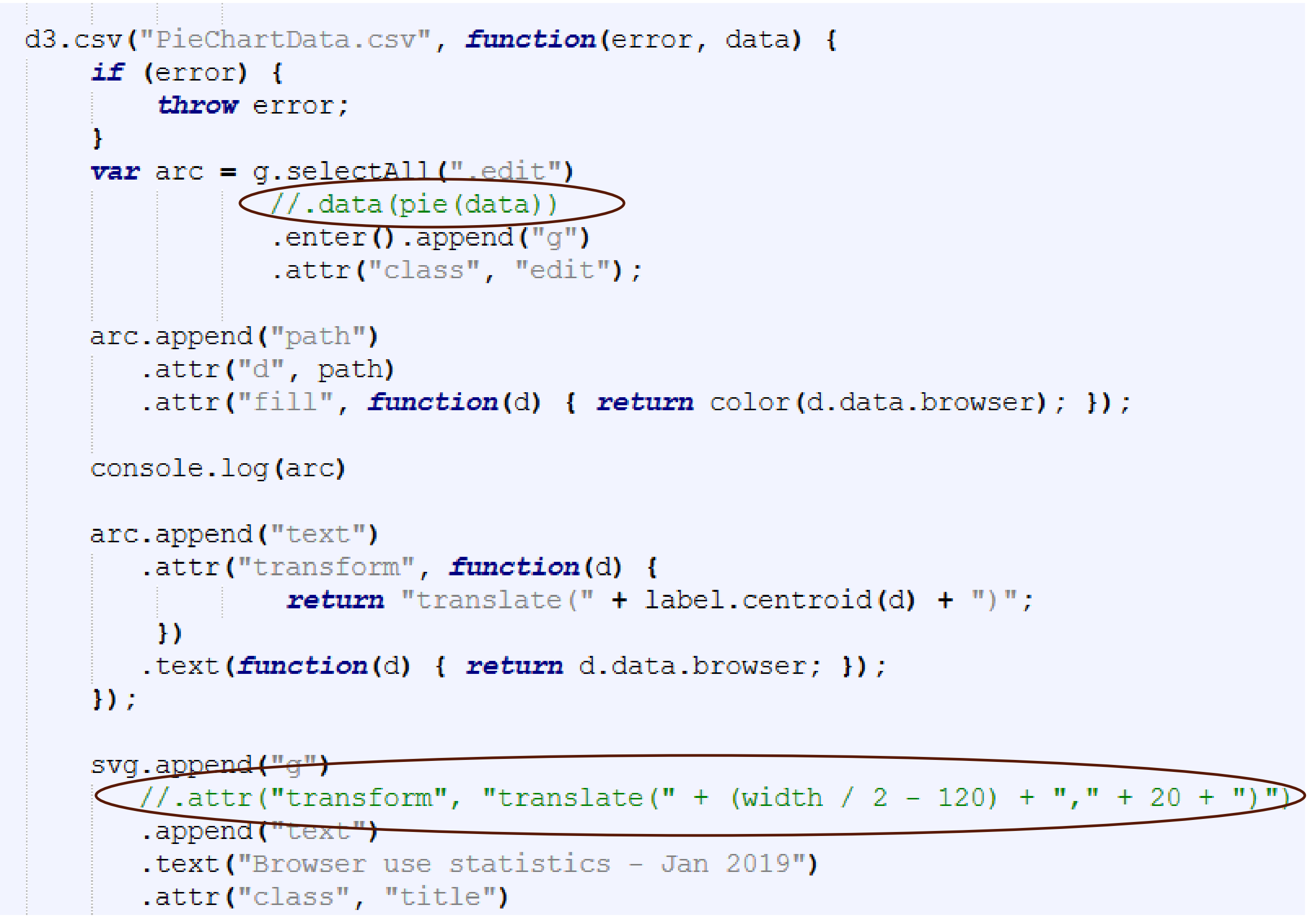
Adjust for beginning length of text



Class Exercise

- Open the file 1_D3_Chart_5_PieChart.html and edit the codes to display the Pie Chart.

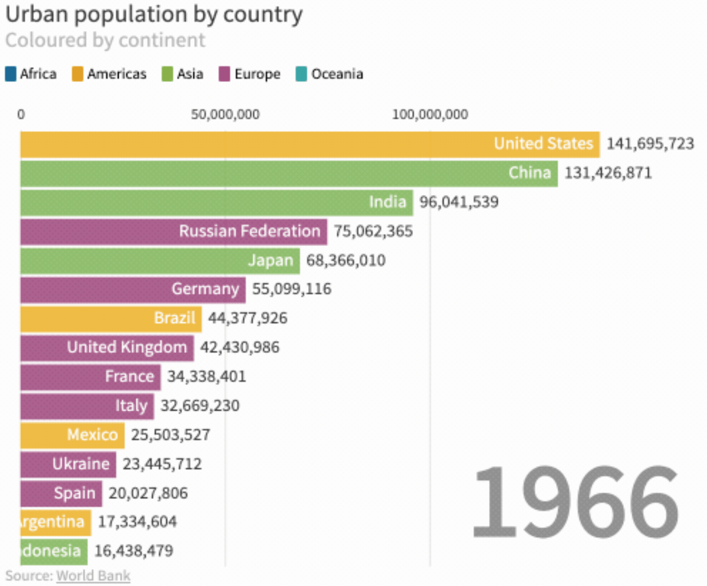




```
d3.csv("PieChartData.csv", function(error, data) {
  if (error) {
    throw error;
  }
  var arc = g.selectAll(".edit")
        //.data(pie(data))
        .enter().append("g")
        .attr("class", "edit");

  arc.append("path")
    .attr("d", path)
    .attr("fill", function(d) { return color(d.data.browser); });

  console.log(arc)

  arc.append("text")
    .attr("transform", function(d) {
          return "translate(" + label.centroid(d) + ")";
     })
    .text(function(d) { return d.data.browser; });
  });

  svg.append("g")
    //.attr("transform", "translate(" + (width / 2 - 120) + "," + 20 + ")")
    .append("text")
    .text("Browser use statistics - Jan 2019")
    .attr("class", "title")
```

Animation

Interaction and animation **not only display** data but **also explain data**. Animation is nothing but a **transition** from one form to another - **changing attributes over time - interval**



- **Start** a new transition with **.transition()** and then define the **final state** of each animated attribute. By **default**, every transition takes **250 milliseconds**; change the timing with **.duration().**

d3.select('rect').transition().**duration(750)**

- New transitions are executed on **all properties simultaneously** unless a **.delay()** is set. Delays are handy when making **transitions happen in sequence**.

Animation - Transitions

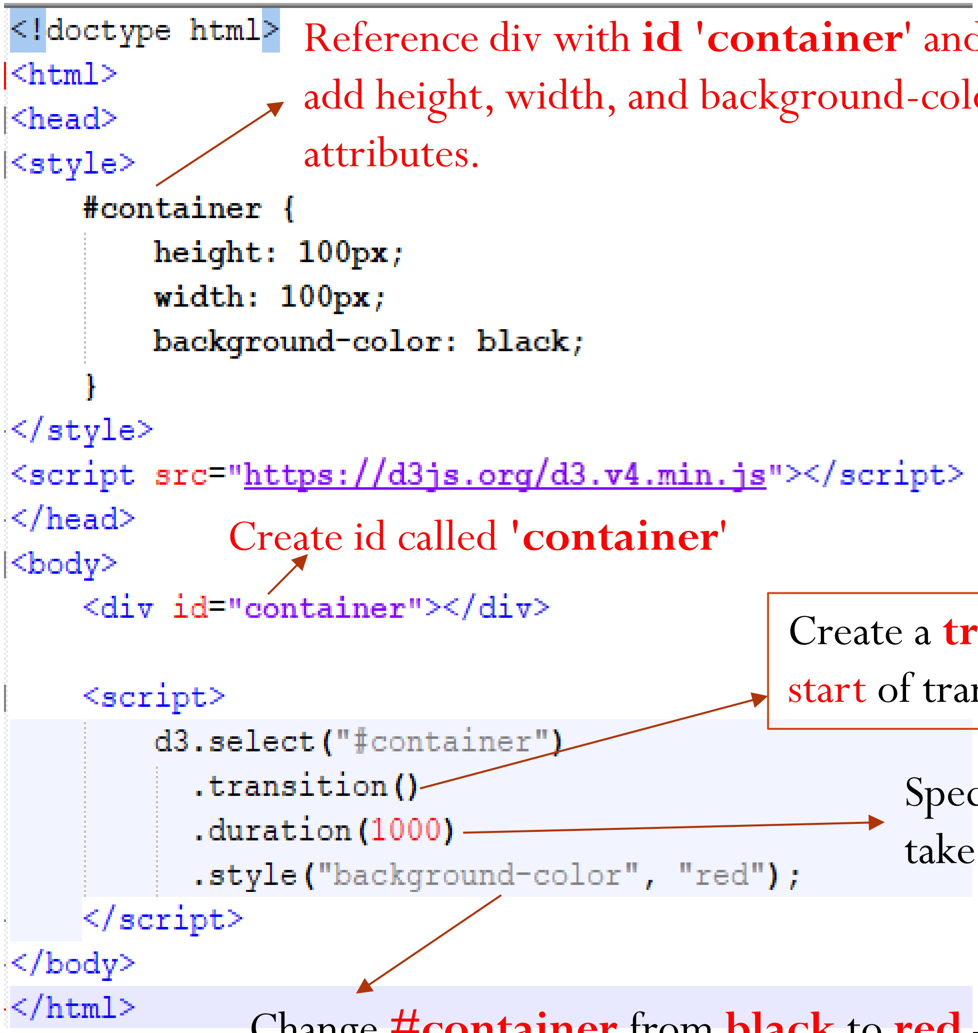
- D3 simplifies the process of animations with transitions. Transitions are made on DOM selections using **<selection>.transition()** method.
- Animation is nothing but a **transition from one form to another over time**.

Method	Description
selection.**transition()**	this **schedules a transition** for the **selected elements**
transition.**duration**()	duration specifies the **animation duration** in **milliseconds** for each element
transition.**ease**()	ease specifies the **easing function**, example: linear, elastic, bounce Ease指定了ease函数，例如：线性、弹性、弹跳
transition.**delay**()	delay specifies the **delay in animation** in **milliseconds** for each element

- The **d3.selection.transition()** method indicates the **start** of transition and then **different transition functions** can be applied to the selected elements.

Animation - Transitions

```
var t = d3.transition()
        .duration(500)
```



```
<!doctype html>
<html>
<head>
<style>
    #container {
        height: 100px;
        width: 100px;
        background-color: black;
    }
</style>
<script src="https://d3js.org/d3.v4.min.js"></script>
</head>
<body>
    <div id="container"></div>

    <script>
        d3.select("#container")
          .transition()
          .duration(1000)
          .style("background-color", "red");
    </script>
</body>
</html>
```

Reference div with **id 'container'** and add height, width, and background-color attributes.

Create id called **'container'**

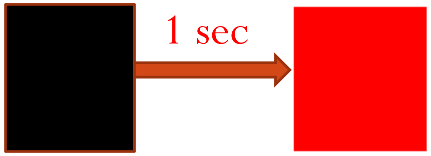
Create a **transition** for selected #container – indicate start of transition – other functions can be applied.

Specify **how long** the transition should take place (**1000 milliseconds**).

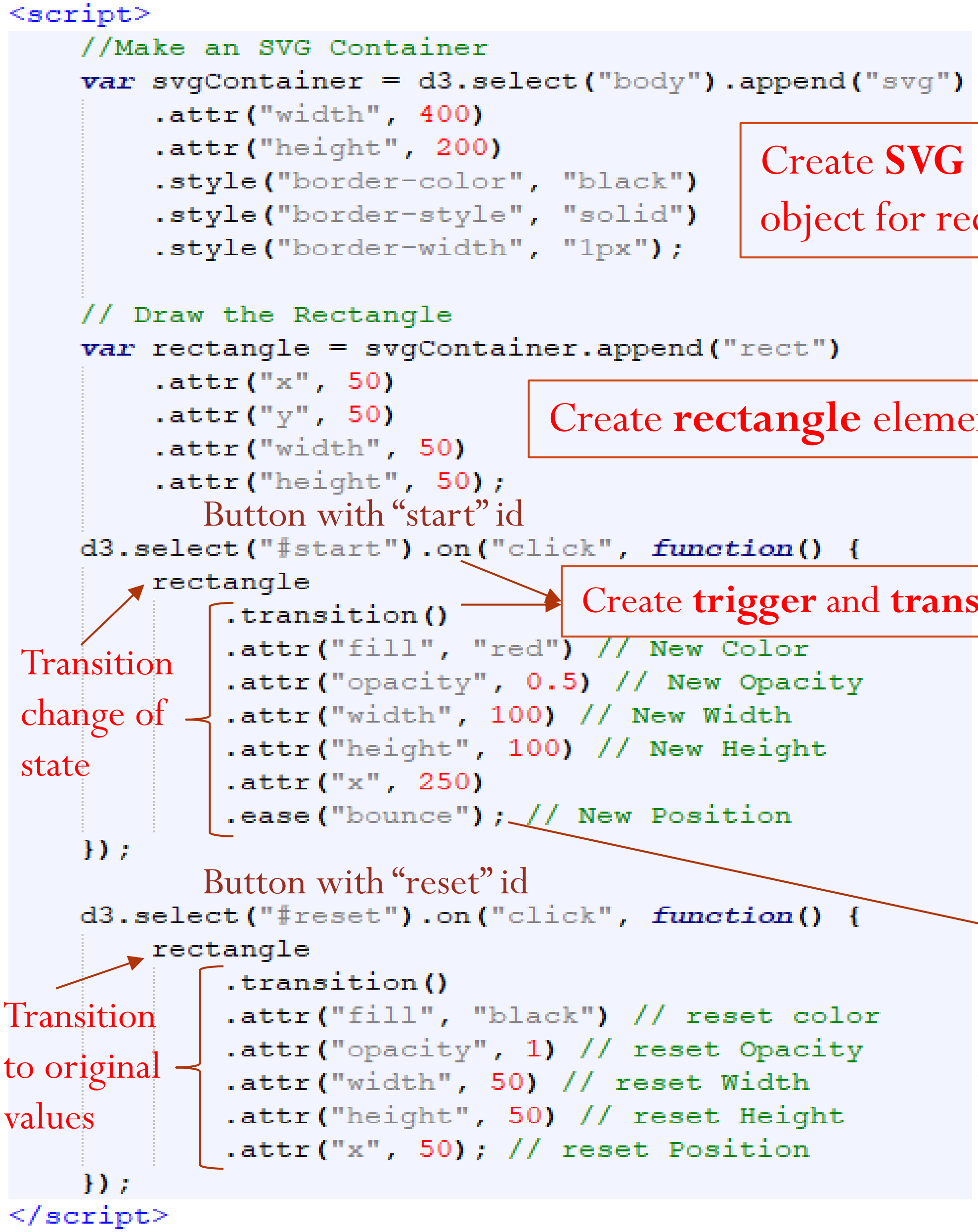
Change **#container** from **black** to **red** – whole transition takes place in **1 sec**

```
d3.select("#container")
   .transition(t)
   .style("background-color", "red");
```

Can also create a transition and store it in a **variable -** to apply animations to different elements.



Animation



```
<script>
    //Make an SVG Container
    var svgContainer = d3.select("body").append("svg")
        .attr("width", 400)
        .attr("height", 200)
        .style("border-color", "black")
        .style("border-style", "solid")
        .style("border-width", "1px");

    // Draw the Rectangle
    var rectangle = svgContainer.append("rect")
        .attr("x", 50)
        .attr("y", 50)
        .attr("width", 50)
        .attr("height", 50);

    d3.select("#start").on("click", function() {
        rectangle
            .transition()
            .attr("fill", "red") // New Color
            .attr("opacity", 0.5) // New Opacity
            .attr("width", 100) // New Width
            .attr("height", 100) // New Height
            .attr("x", 250)
            .ease("bounce"); // New Position
    });

    d3.select("#reset").on("click", function() {
        rectangle
            .transition()
            .attr("fill", "black") // reset color
            .attr("opacity", 1) // reset Opacity
            .attr("width", 50) // reset Width
            .attr("height", 50) // reset Height
            .attr("x", 50); // reset Position
    });
</script>
```

Create **SVG** object for rect

Create **rectangle** element

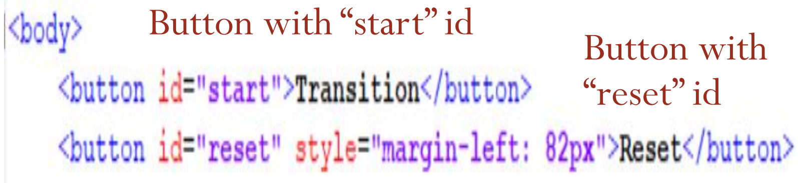
Button with "start" id

Create **trigger** and **transition**

Transition change of state

Button with "reset" id

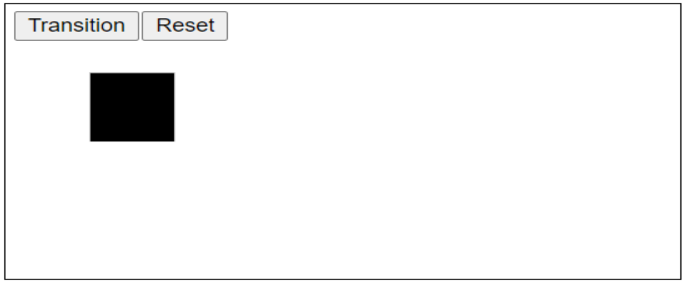
Transition to original values

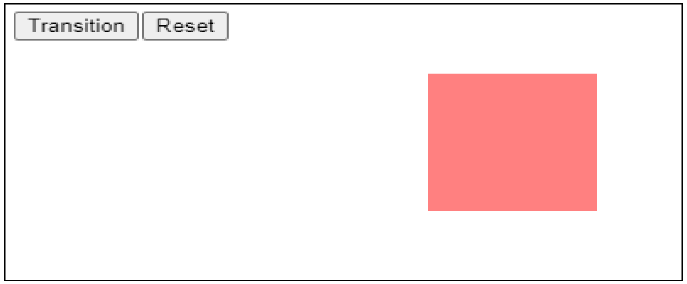


```
<body>
    <button id="start">Transition</button>
    <button id="reset" style="margin-left: 82px">Reset</button>
```

Button with "start" id

Button with "reset" id



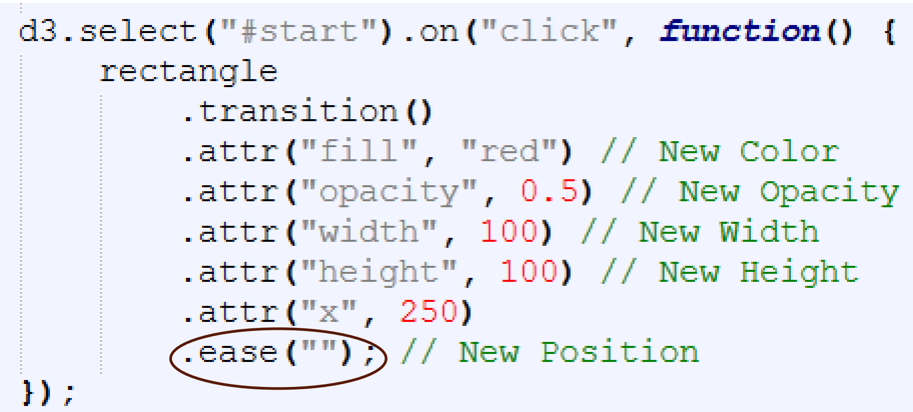


To implement a transition, set up an **event trigger** (button, click or user event) – then within trigger, change respective attributes to modify.

Ease arguments - **cubic-in-out** (fast, slow, fast - default), **linear**, **elastic**, **bounce** => specify and control **motion of transition**

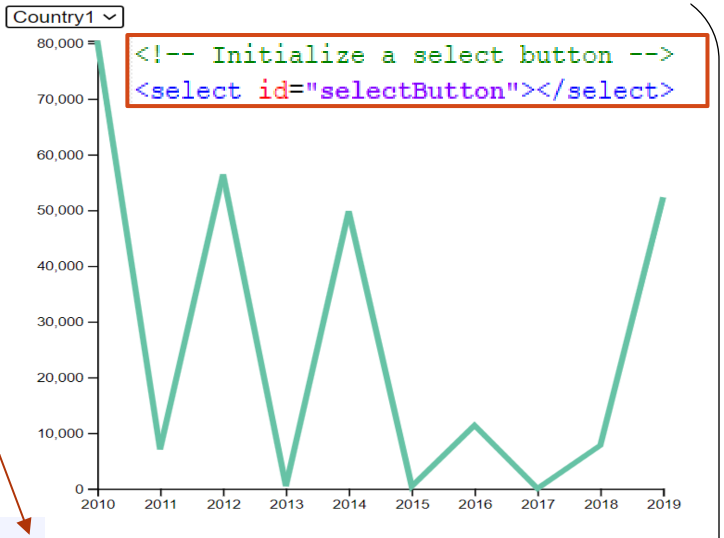
Class Exercise

- Open the 2_D3_Animations_1_Box.html file and change the argument (**cubic-in-out, linear, elastic, bounce**) of the **ease function** to see the different effects.



```
d3.select("#start").on("click", function() {
    rectangle
        .transition()
        .attr("fill", "red") // New Color
        .attr("opacity", 0.5) // New Opacity
        .attr("width", 100) // New Width
        .attr("height", 100) // New Height
        .attr("x", 250)
        .ease(""); // New Position
});
```

Animation – Line Chart

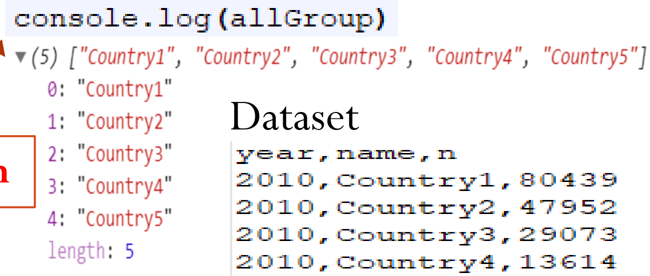


```
<!-- Initialize a select button -->
<select id="selectButton"></select>
```

```
<script>
// set the dimensions and margins of the graph
var margin = {top: 10, right: 30, bottom: 30, left: 60},
    width = 460 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#my_dataviz")
  .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 + ")");

//Read the data
d3.csv("LineAnimationData.csv", function(data) {

    // List of groups (one group per column)
    var allGroup = d3.map(data, function(d){return(d.name)}).keys()

    // add the options to the button
    d3.select("#selectButton")
      .selectAll('myOptions')
        .data(allGroup)
      .enter()
        .append('option')
      .text(function (d) { return d; }) // text showed in the menu
      // corresponding value returned by the button
      .attr("value", function (d) { return d; })
```

Create **svg object**, append group element and transform group with margin

.map() creates **new data array** based on function
- return **unique country names** through **key values**.



```
console.log(allGroup)
▼(5) ["Country1", "Country2", "Country3", "Country4", "Country5"]
   0: "Country1"
   1: "Country2"
   2: "Country3"
   3: "Country4"
   4: "Country5"
   length: 5
```

add **options to selectButton**

Input data array of **country names**

Add option element

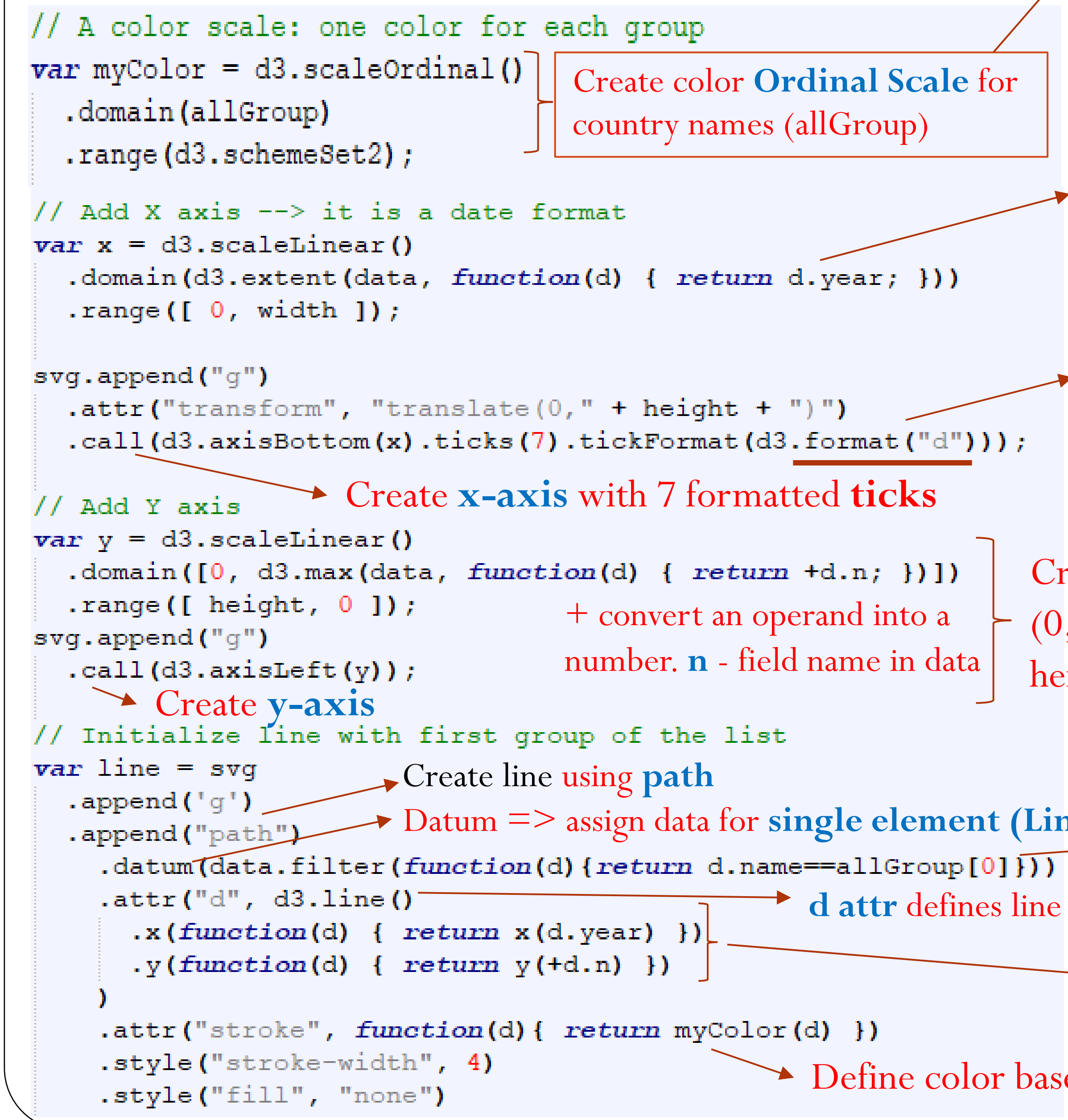
Add d.name to **option text**

Assign **country name value** to **"selectButton"** element => for country selection later

Dataset

```
year,name,n
2010,Country1,80439
2010,Country2,47952
2010,Country3,29073
2010,Country4,13614
2010,Country5,9555
2011,Country1,7060
2011,Country2,402
2011,Country3,379
2011,Country4,15
2011,Country5,73947
```

Animation – Line Chart



```
// A color scale: one color for each group
var myColor = d3.scaleOrdinal()
  .domain(allGroup)
  .range(d3.schemeSet2);

// Add X axis --> it is a date format
var x = d3.scaleLinear()
  .domain(d3.extent(data, function(d) { return d.year; }))
  .range([ 0, width ]);

svg.append("g")
  .attr("transform", "translate(0," + height + ")")
  .call(d3.axisBottom(x).ticks(7).tickFormat(d3.format("d")));

// Add Y axis
var y = d3.scaleLinear()
  .domain([0, d3.max(data, function(d) { return +d.n; })])
  .range([ height, 0 ]);
svg.append("g")
  .call(d3.axisLeft(y));

// Initialize line with first group of the list
var line = svg
  .append('g')
  .append("path")
    .datum(data.filter(function(d){return d.name==allGroup[0]}))
    .attr("d", d3.line()
      .x(function(d) { return x(d.year) })
      .y(function(d) { return y(+d.n) })
    )
    .attr("stroke", function(d){ return myColor(d) })
    .style("stroke-width", 4)
    .style("fill", "none")
```

Create color **Ordinal Scale** for country names (allGroup)

d3.schemeSet2 method - return an array of **eight categorical colors** in RGB hexadecimal strings.

Returns the **year data** and use it to create **x-scale** with **domain** as year data and **range** based on width of svg element.

Format year to "d" - **decimal notation**, rounded to **integer** (else output is 2,010 instead of 2010

Create **x-axis** with 7 formatted **ticks**

Create **y-scale** with **domain** values (0, **max of n)** and **range** based on height of svg element

+ convert an operand into a number. **n** - field name in data

Create **y-axis**

Create line using **path**

Datum => assign data for **single element (Line)**.

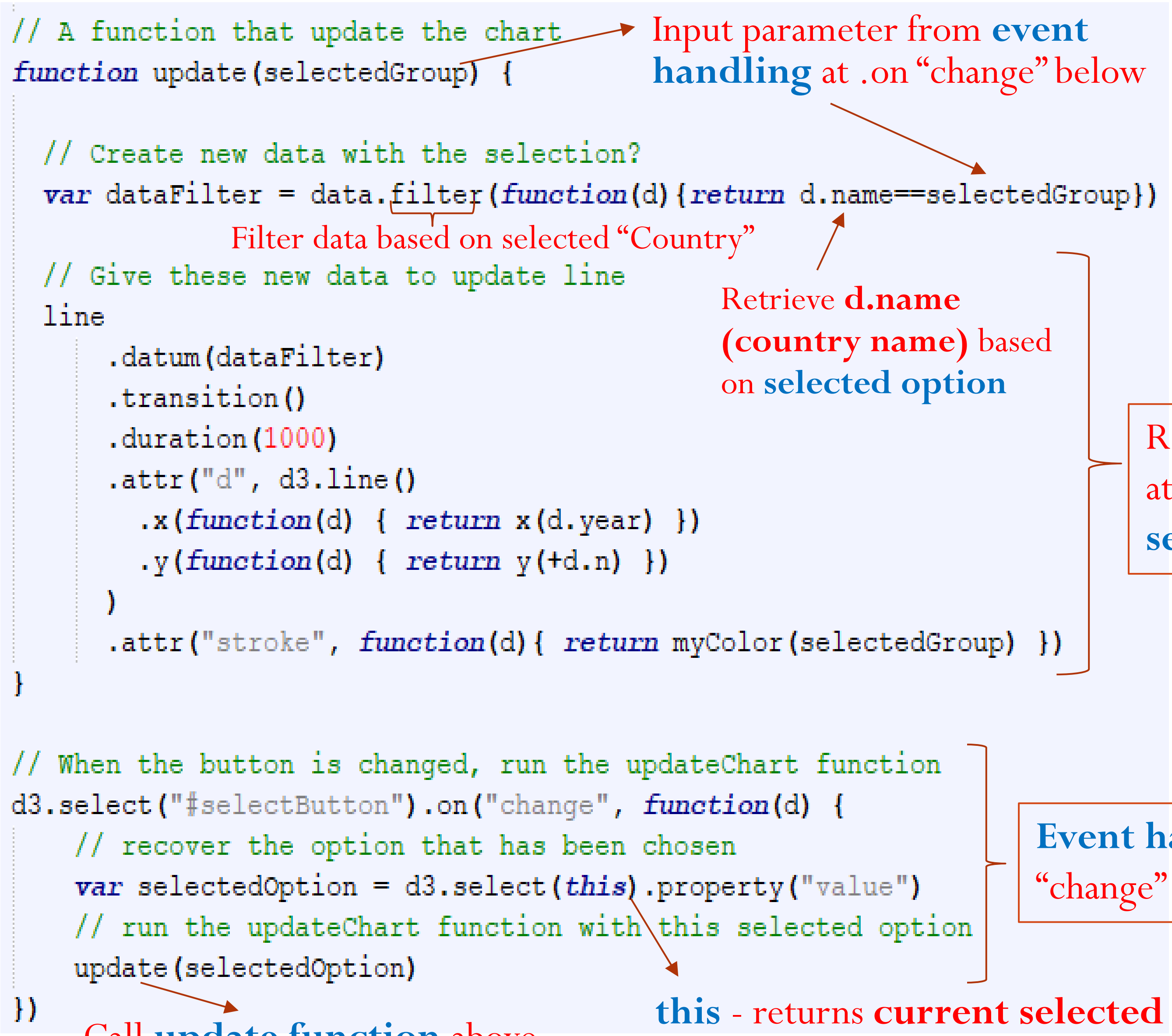
Filter data based on first record – **'Country1'**

d attr defines line path

Here d is the parameter of the anonymous function

Define color based on **myColor** color-scale

Animation – Line Chart



```
// A function that update the chart
function update(selectedGroup) {

  // Create new data with the selection?
  var dataFilter = data.filter(function(d){return d.name==selectedGroup})

  // Give these new data to update line
  line
      .datum(dataFilter)
      .transition()
      .duration(1000)
      .attr("d", d3.line()
        .x(function(d) { return x(d.year) })
        .y(function(d) { return y(+d.n) })
      )
      .attr("stroke", function(d){ return myColor(selectedGroup) })
}

// When the button is changed, run the updateChart function
d3.select("#selectButton").on("change", function(d) {
    // recover the option that has been chosen
    var selectedOption = d3.select(this).property("value")
    // run the updateChart function with this selected option
    update(selectedOption)
})
```

Input parameter from **event handling** at .on “change” below

Filter data based on selected “Country”

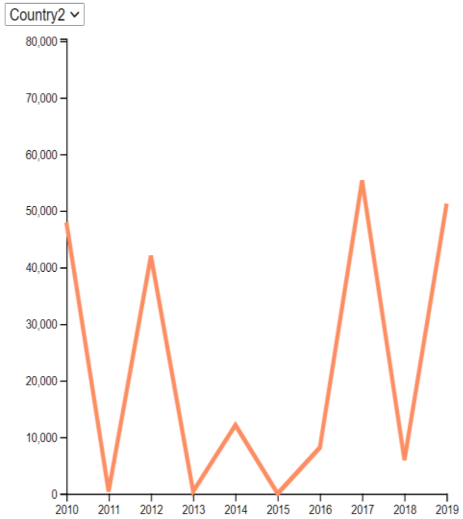
Retrieve **d.name (country name)** based on **selected option**

Reassign line data and attributes based on **new selection**

Event handling when .on “change” for selectButton

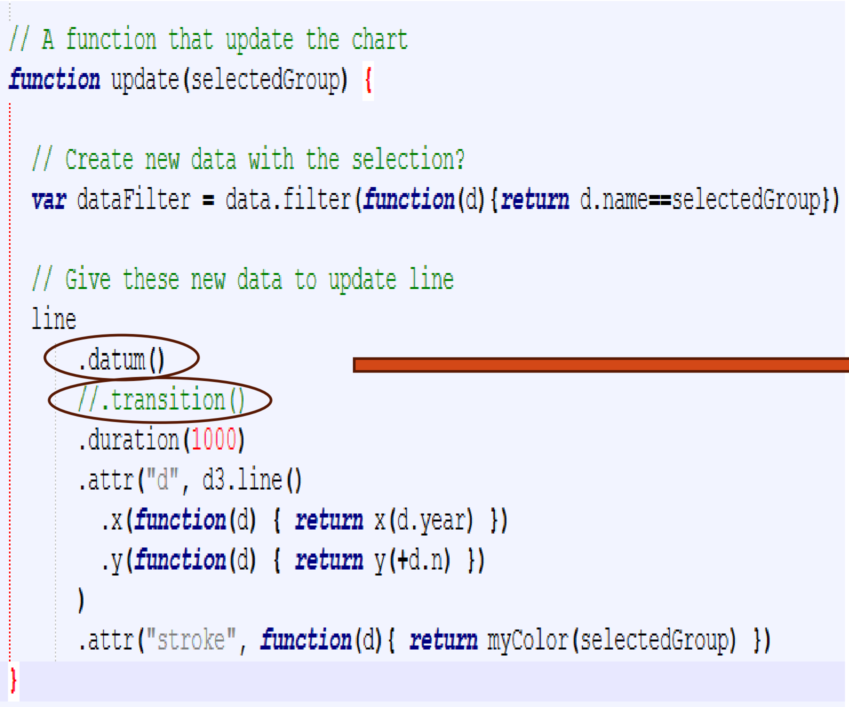
Call **update function** above

this - returns **current selected value** for #selectButton

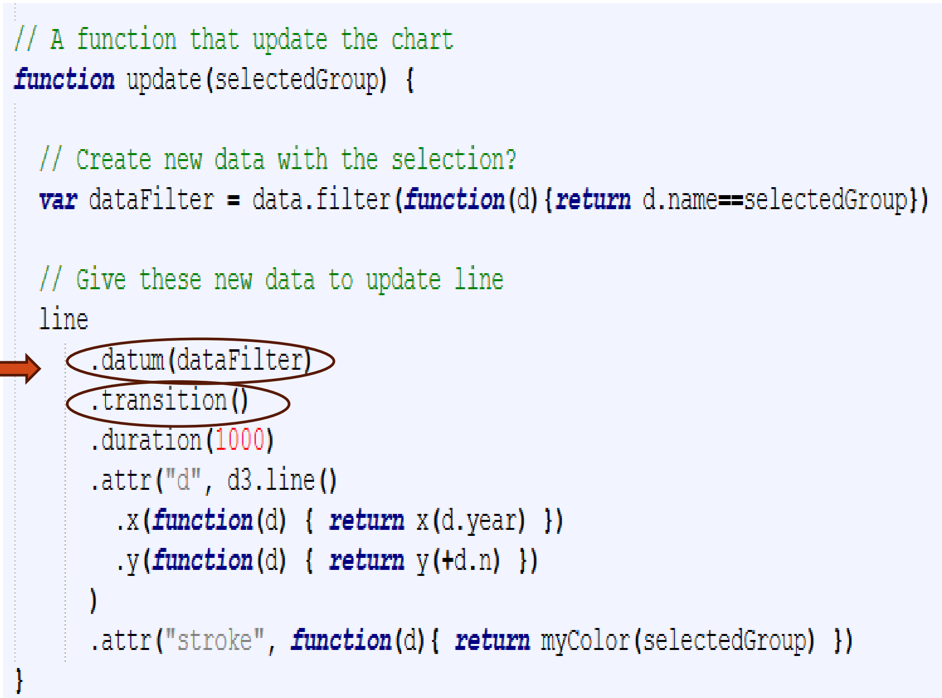


Class Exercise

- Open the **2_D3_Animations_2_LineChart.html** file and edit it to display the animation.

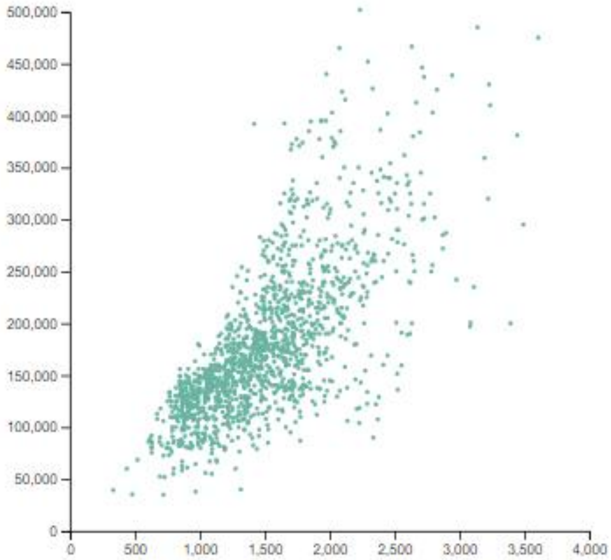


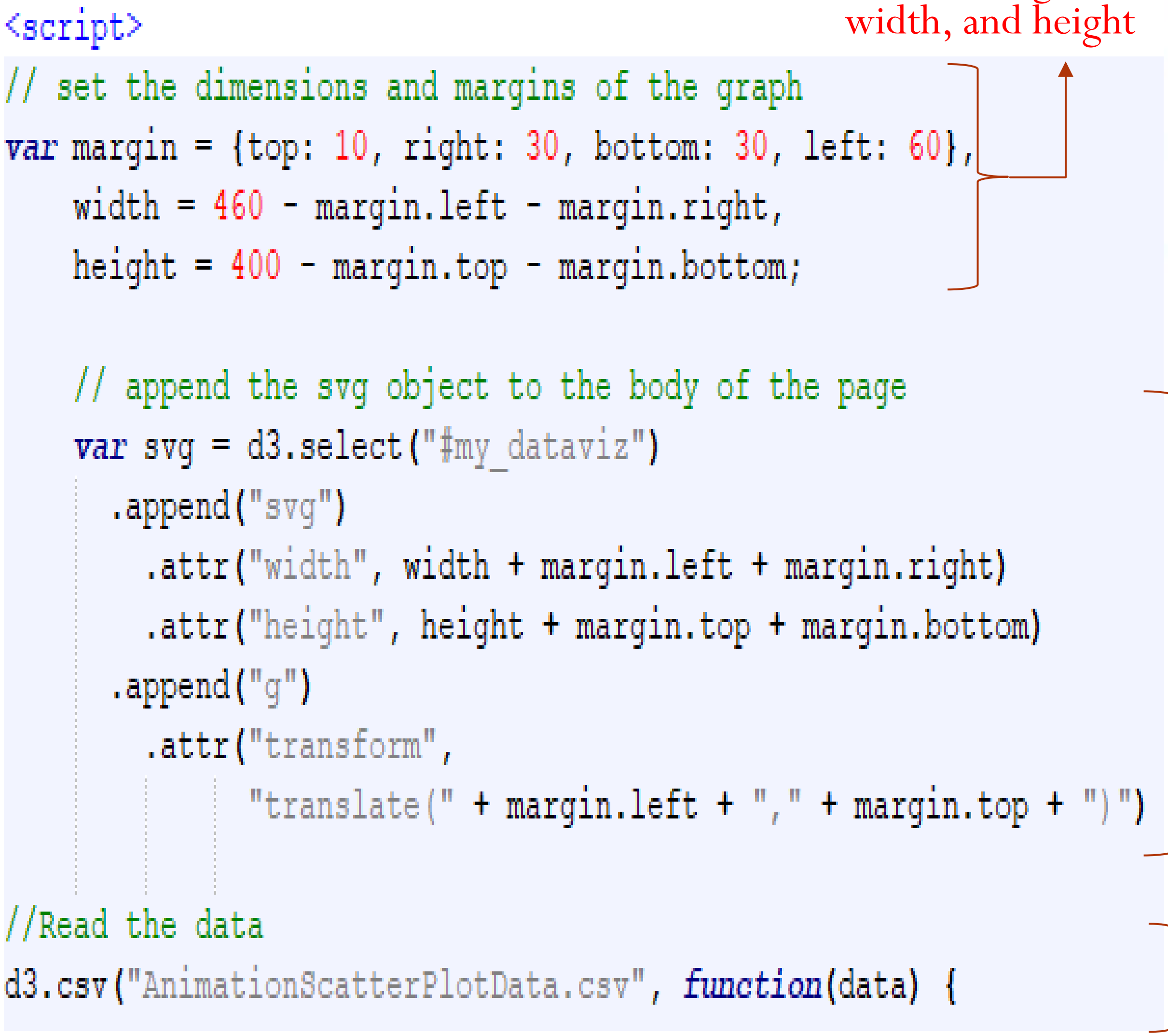
```
// A function that update the chart
function update(selectedGroup) {

  // Create new data with the selection?
  var dataFilter = data.filter(function(d){return d.name==selectedGroup})

  // Give these new data to update line
  line
    .datum()
    //.transition()
    .duration(1000)
    .attr("d", d3.line()
      .x(function(d) { return x(d.year) })
      .y(function(d) { return y(+d.n) })
    )
    .attr("stroke", function(d){ return myColor(selectedGroup) })
}
```



```
// A function that update the chart
function update(selectedGroup) {

  // Create new data with the selection?
  var dataFilter = data.filter(function(d){return d.name==selectedGroup})

  // Give these new data to update line
  line
    .datum(dataFilter)
    .transition()
    .duration(1000)
    .attr("d", d3.line()
      .x(function(d) { return x(d.year) })
      .y(function(d) { return y(+d.n) })
    )
    .attr("stroke", function(d){ return myColor(selectedGroup) })
}
```

Animation - Scatterplot





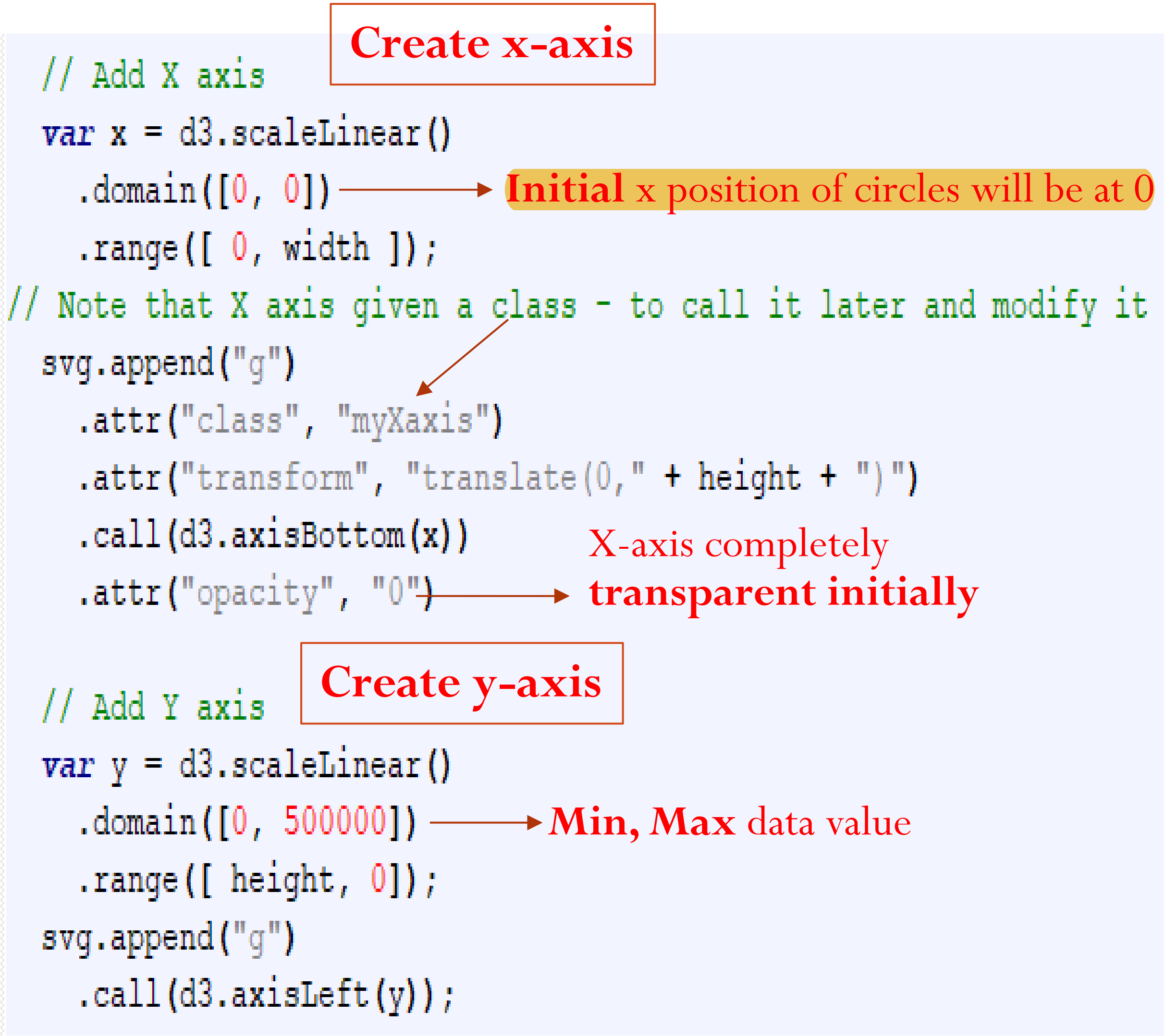
```
<script>
// set the dimensions and margins of the graph
var margin = {top: 10, right: 30, bottom: 30, left: 60},
    width = 460 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

    // append the svg object to the body of the page
    var svg = d3.select("#my_dataviz")
      .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 + ")")

//Read the data
d3.csv("AnimationScatterPlotData.csv", function(data) {
```

Define margin, width, and height

Create **svg object**, append group element and transform group with margin

Read data file

Animation - Scatterplot



```
// Add X axis
var x = d3.scaleLinear()
  .domain([0, 0])
  .range([ 0, width ]);
// Note that X axis given a class - to call it later and modify it
svg.append("g")
  .attr("class", "myXaxis")
  .attr("transform", "translate(0," + height + ")")
  .call(d3.axisBottom(x))
  .attr("opacity", "0")

// Add Y axis
var y = d3.scaleLinear()
  .domain([0, 500000])
  .range([ height, 0]);
svg.append("g")
  .call(d3.axisLeft(y));
```

Create x-axis

Initial x position of circles will be at 0

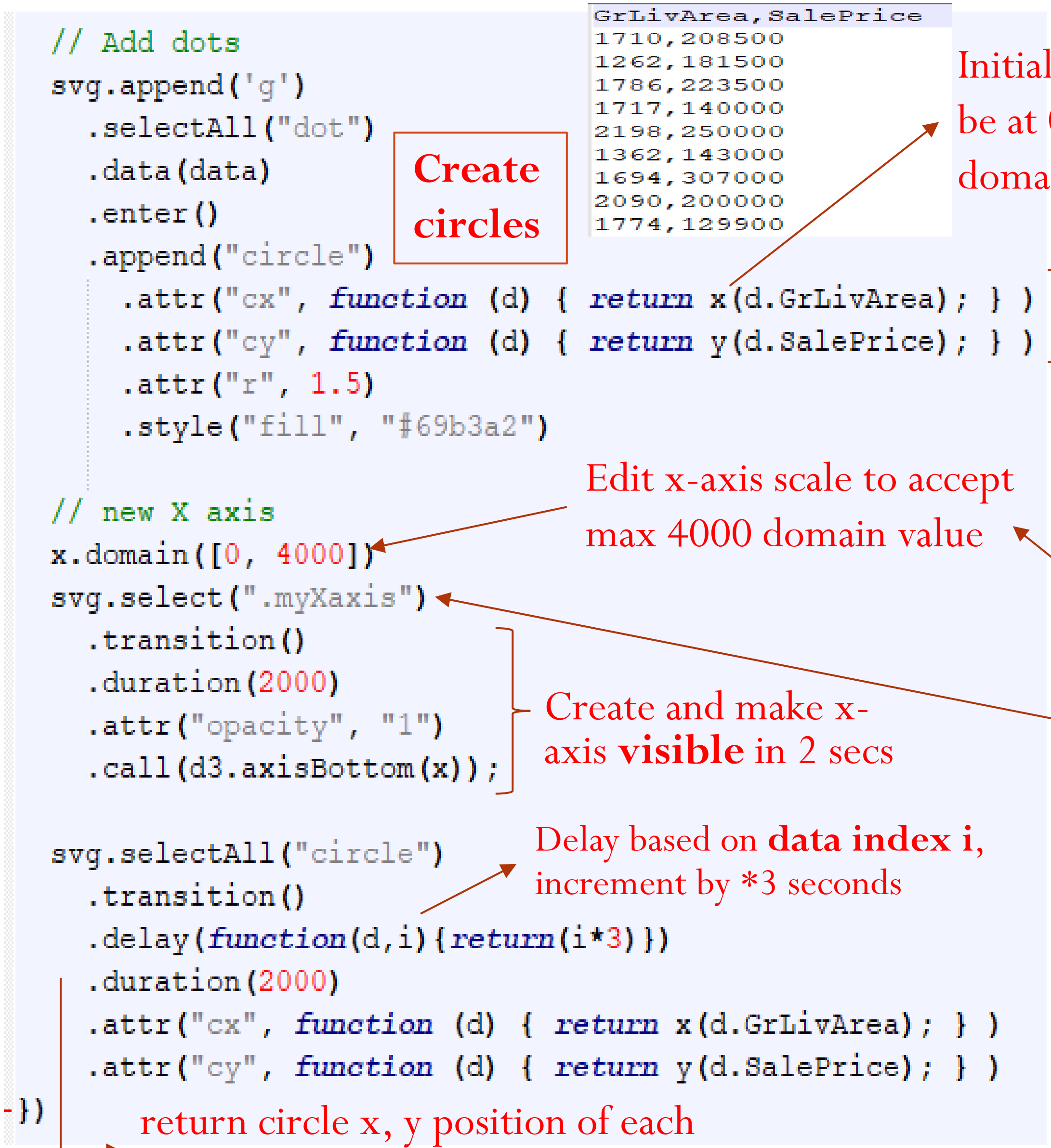
no default value

X-axis completely **transparent initially**

Create y-axis

Min, Max data value

Animation - Scatterplot



```
// Add dots
svg.append('g')
  .selectAll("dot")
  .data(data)
  .enter()
  .append("circle")
    .attr("cx", function (d) { return x(d.GrLivArea); } )
    .attr("cy", function (d) { return y(d.SalePrice); } )
    .attr("r", 1.5)
    .style("fill", "#69b3a2")

// new X axis
x.domain([0, 4000])
svg.select(".myXaxis")
  .transition()
  .duration(2000)
  .attr("opacity", "1")
  .call(d3.axisBottom(x));

svg.selectAll("circle")
  .transition()
  .delay(function(d,i){return(i*3)})
  .duration(2000)
  .attr("cx", function (d) { return x(d.GrLivArea); } )
  .attr("cy", function (d) { return y(d.SalePrice); } )
-})
```

```
GrLivArea,SalePrice
1710,208500
1262,181500
1786,223500
1717,140000
2198,250000
1362,143000
1694,307000
2090,200000
1774,129900
```

Create circles

Initial x, y position of dot based on data values

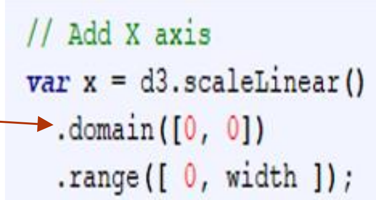
Initial x position will be at 0 due to domain specification

Edit x-axis scale to accept max 4000 domain value

Create and make x-axis **visible** in 2 secs

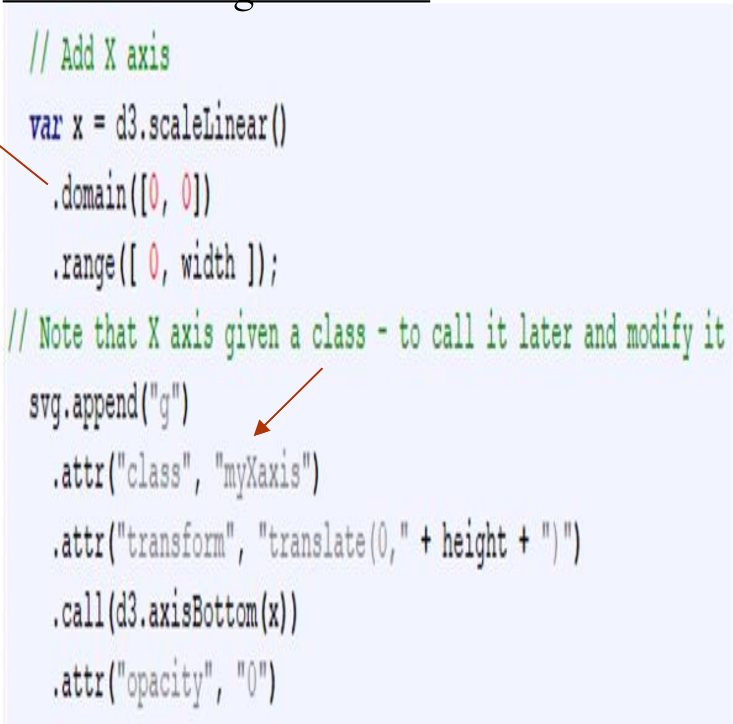
Delay based on **data index i**, increment by *3 seconds

return circle x, y position of each data value within **(delay) + 2 secs**



```
// Add X axis
var x = d3.scaleLinear()
  .domain([0, 0])
  .range([ 0, width ]);
```

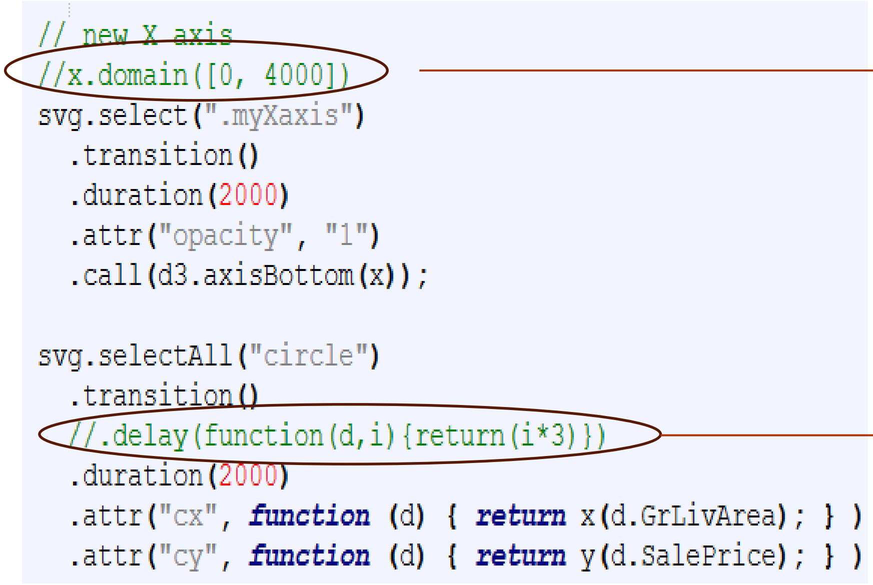
Codes for original x-axis



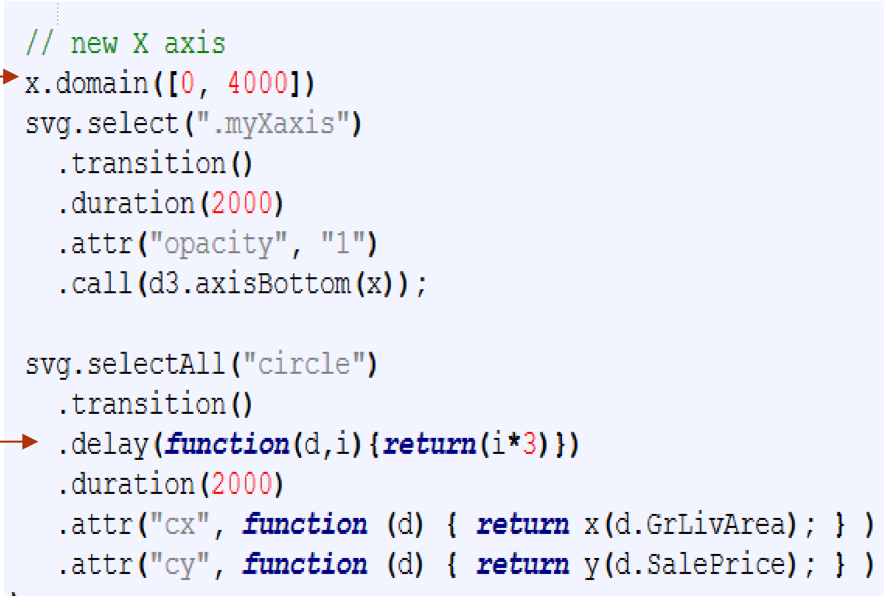
```
// Add X axis
var x = d3.scaleLinear()
  .domain([0, 0])
  .range([ 0, width ]);
// Note that X axis given a class - to call it later and modify it
svg.append("g")
  .attr("class", "myXaxis")
  .attr("transform", "translate(0," + height + ")")
  .call(d3.axisBottom(x))
  .attr("opacity", "0")
```

Class Exercise - Animation - Scatterplot

- Open the **2_D3_Animations_4_Scatterplot.html** file and edit it to display the animation.



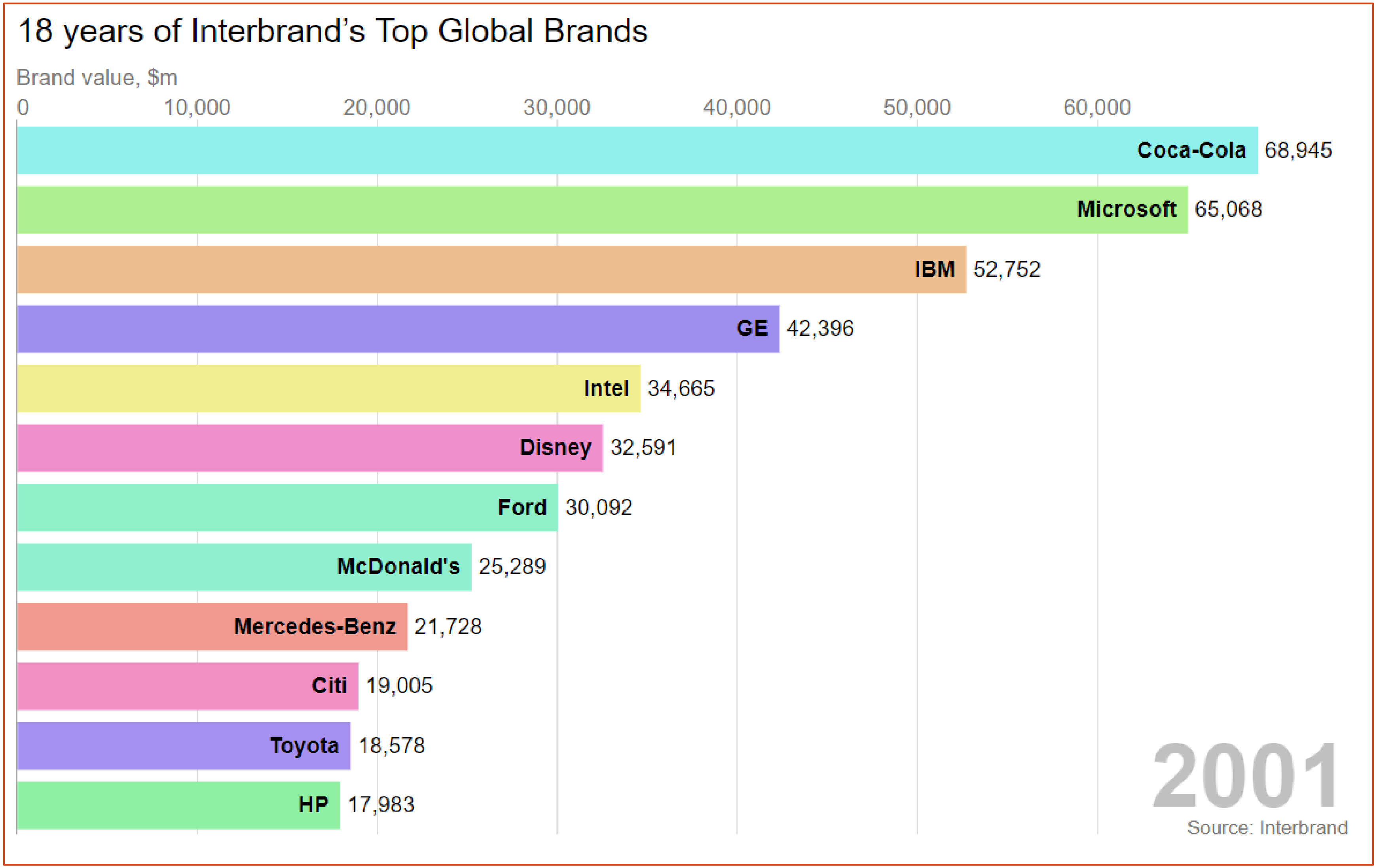
```
// new X axis
//x.domain([0, 4000])
svg.select(".myXaxis")
  .transition()
  .duration(2000)
  .attr("opacity", "1")
  .call(d3.axisBottom(x));

svg.selectAll("circle")
  .transition()
  //.delay(function(d,i){return(i*3)})
  .duration(2000)
  .attr("cx", function (d) { return x(d.GrLivArea); } )
  .attr("cy", function (d) { return y(d.SalePrice); } )
```



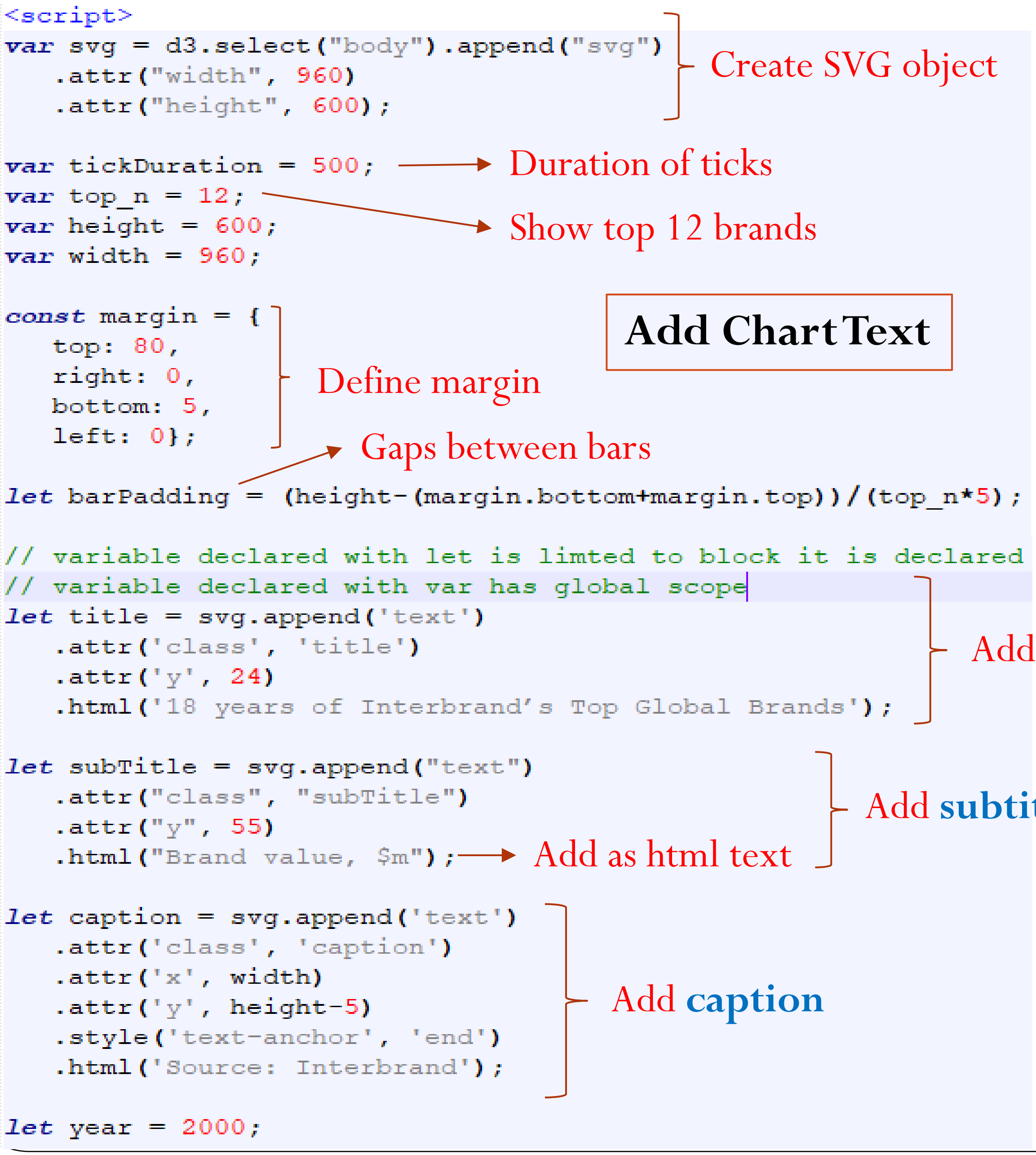
```
// new X axis
x.domain([0, 4000])
svg.select(".myXaxis")
  .transition()
  .duration(2000)
  .attr("opacity", "1")
  .call(d3.axisBottom(x));

svg.selectAll("circle")
  .transition()
  .delay(function(d,i){return(i*3)})
  .duration(2000)
  .attr("cx", function (d) { return x(d.GrLivArea); } )
  .attr("cy", function (d) { return y(d.SalePrice); } )
```

Animation - Bar Chart Race

What are the components to code?



Animation - Bar Chart Race



```
<script>
var svg = d3.select("body").append("svg")
    .attr("width", 960)
    .attr("height", 600);

var tickDuration = 500;
var top_n = 12;
var height = 600;
var width = 960;

const margin = {
    top: 80,
    right: 0,
    bottom: 5,
    left: 0};

let barPadding = (height-(margin.bottom+margin.top))/(top_n*5);

// variable declared with let is limted to block it is declared
// variable declared with var has global scope
let title = svg.append('text')
    .attr('class', 'title')
    .attr('y', 24)
    .html('18 years of Interbrand’s Top Global Brands');

let subTitle = svg.append("text")
    .attr("class", "subTitle")
    .attr("y", 55)
    .html("Brand value, $m");

let caption = svg.append('text')
    .attr('class', 'caption')
    .attr('x', width)
    .attr('y', height-5)
    .style('text-anchor', 'end')
    .html('Source: Interbrand');

let year = 2000;
```

Create SVG object

Duration of ticks

Show top 12 brands

Add Chart Text

Define margin

Gaps between bars

Add **title**

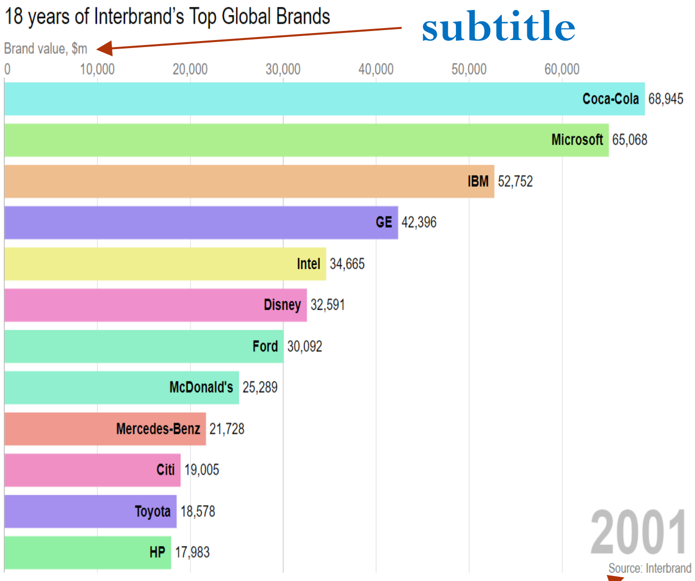
Add **subtitle**

Add as html text

Add **caption**

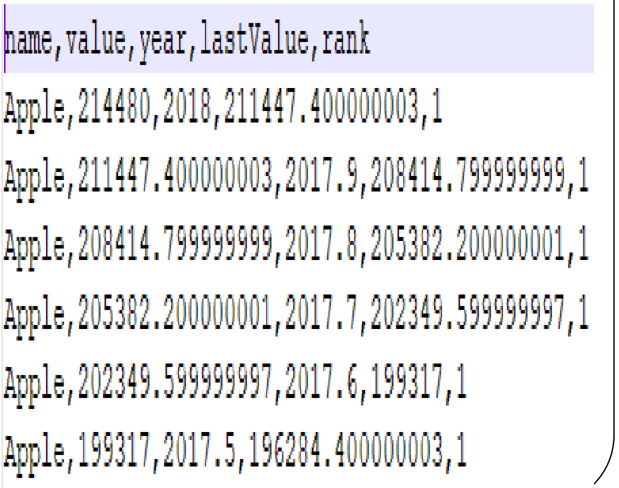
title

subtitle



caption

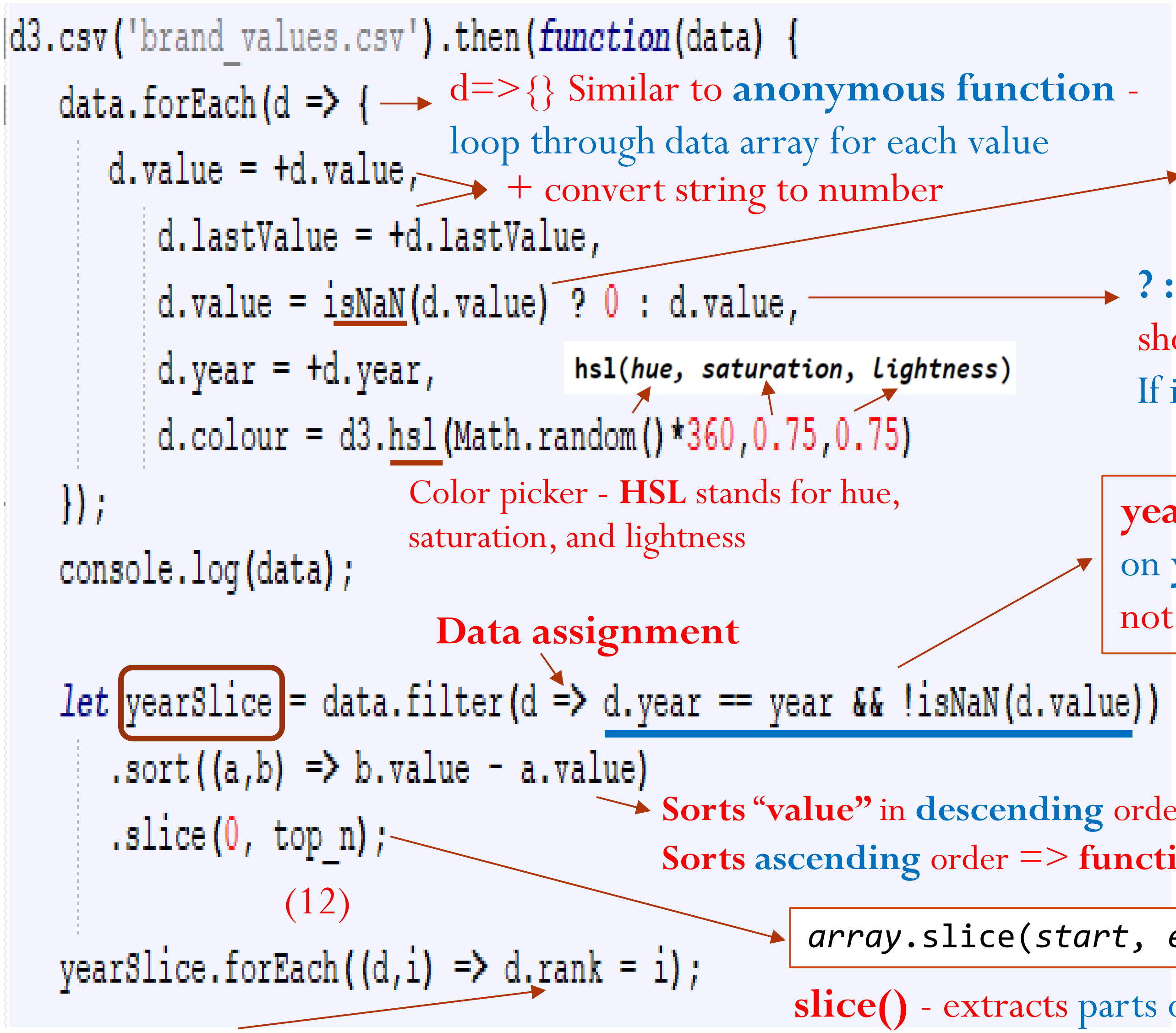
brands_values.csv



```
name,value,year,lastValue,rank
Apple,214480,2018,211447.400000003,1
Apple,211447.400000003,2017.9,208414.799999999,1
Apple,208414.799999999,2017.8,205382.200000001,1
Apple,205382.200000001,2017.7,202349.599999997,1
Apple,202349.599999997,2017.6,199317,1
Apple,199317,2017.5,196284.400000003,1
```

Animation - Bar Chart Race

Data Loading



```
d3.csv('brand_values.csv').then(function(data) {
    data.forEach(d => {
        d.value = +d.value,
          d.lastValue = +d.lastValue,
          d.value = isNaN(d.value) ? 0 : d.value,
          d.year = +d.year,
          d.colour = d3.hsl(Math.random()*360,0.75,0.75)
    });
    console.log(data);

    let yearSlice = data.filter(d => d.year == year && !isNaN(d.value))
      .sort((a,b) => b.value - a.value)
      .slice(0, top_n);

    yearSlice.forEach((d,i) => d.rank = i);
```

d=>{} Similar to **anonymous function** - loop through data array for each value

+ convert string to number

isNaN - determines whether a value is an illegal number (**Not-a-Number**)

? : => conditional operator - shorthand for **if-else** statement. If isNaN, value = 0, else d.value

```
hsl(hue, saturation, lightness)
```

Color picker - **HSL** stands for hue, saturation, and lightness

yearSlice => Filter data based on **year** (d.year = year) and value not illegal => returns an **array**

Data assignment

Sorts "value" in **descending** order => **function(a,b){return b-a}**

Sorts ascending order => **function(a, b){return a-b}**

(12)

```
array.slice(start, end)
```

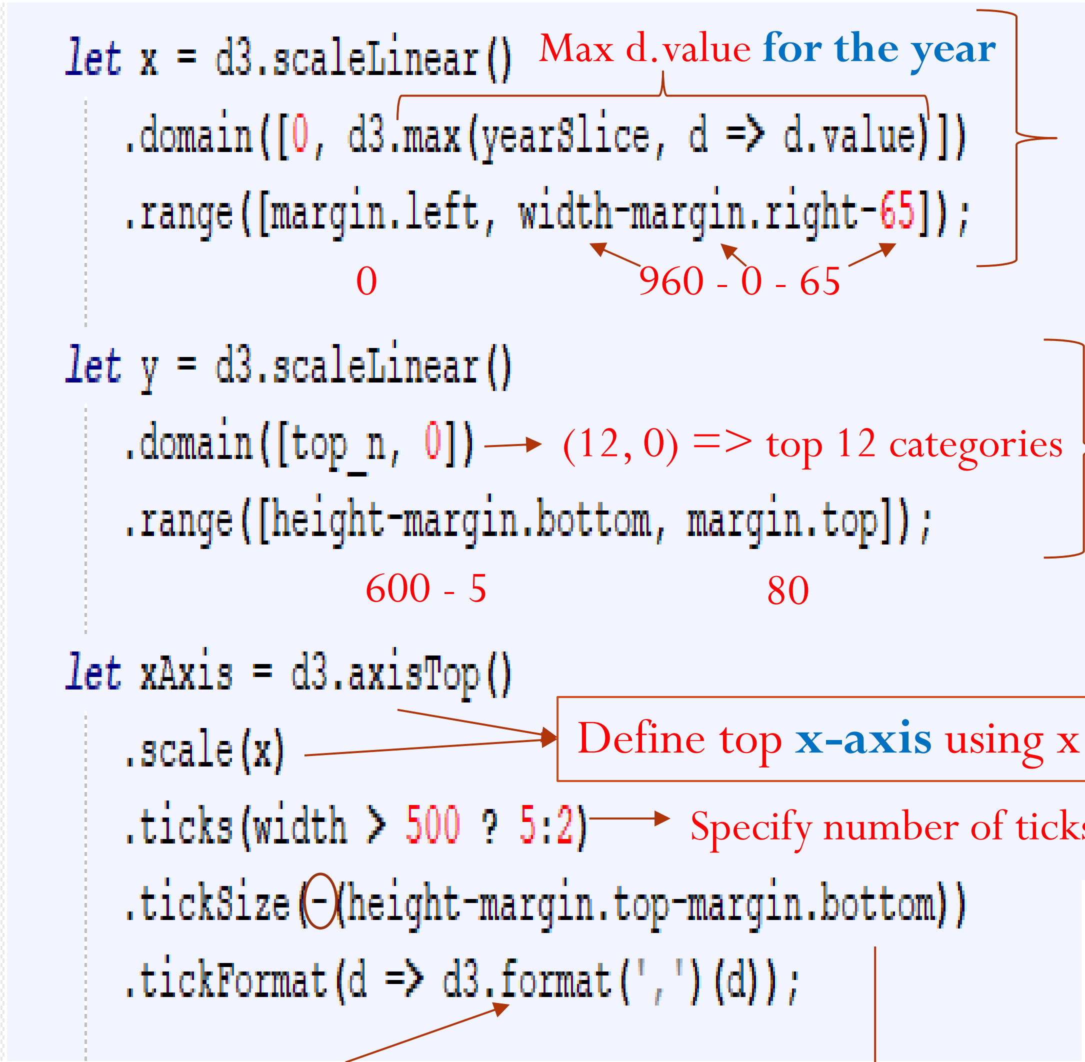
slice() - extracts parts of a string and returns the extracted parts in a new string

- **Return top 12 data records** of **selected year**

Assign rank value from dataset for each top 12 records

Animation - Bar Chart Race

Create Scales



```
let x = d3.scaleLinear()
    .domain([0, d3.max(yearSlice, d => d.value)])
    .range([margin.left, width-margin.right-65]);

let y = d3.scaleLinear()
    .domain([top_n, 0])
    .range([height-margin.bottom, margin.top]);

let xAxis = d3.axisTop()
    .scale(x)
    .ticks(width > 500 ? 5:2)
    .tickSize(-(height-margin.top-margin.bottom))
    .tickFormat(d => d3.format(',')(d));
```

Max d.value **for the year**

0

960 - 0 - 65

Create **x scale** – define **domain** (0, max data value) and **range** values based on margins

(12, 0) => top 12 categories

600 - 5

80

Create **y scale** – define **domain** (12, 0) and **range** values based on margins – lowest rank 12 starts at bottom

Define top **x-axis** using x **scale**

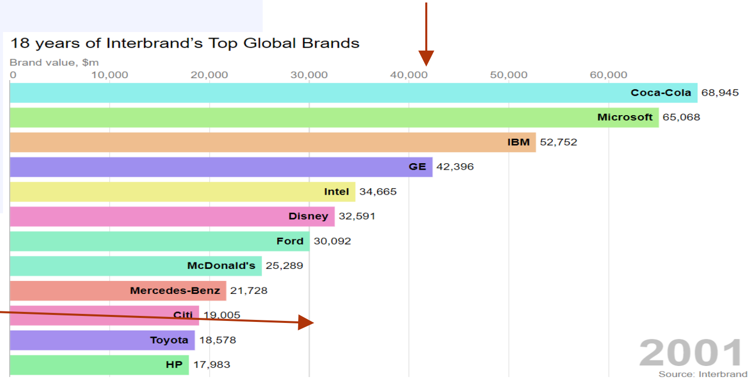
Non-strict (may not be 5) implementation by D3

Specify number of ticks - **if width > 500 => 5, else 2**

Format numbers to use **commas**

tickSize => draw grid lines.

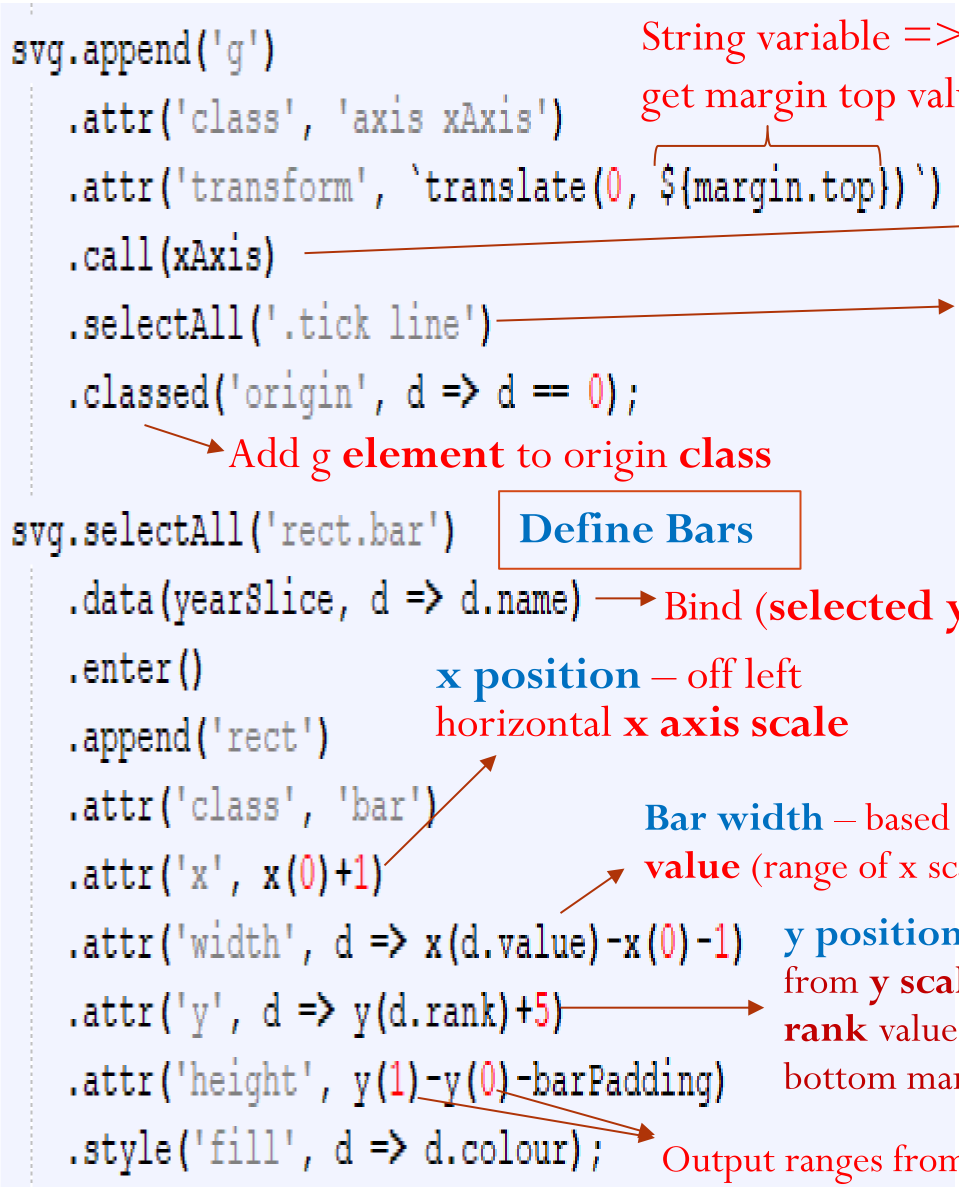
Negative tickSize – ticks drawn **below** axis line



18 years of Interbrand's Top Global Brands

Source: Interbrand

Animation - Bar Chart Race



```
svg.append('g')
  .attr('class', 'axis xAxis')
  .attr('transform', `translate(0, ${margin.top})`)
  .call(xAxis)
  .selectAll('.tick line')
  .classed('origin', d => d == 0);

svg.selectAll('rect.bar')
  .data(yearSlice, d => d.name)
  .enter()
  .append('rect')
  .attr('class', 'bar')
  .attr('x', x(0)+1)
  .attr('width', d => x(d.value)-x(0)-1)
  .attr('y', d => y(d.rank)+5)
  .attr('height', y(1)-y(0)-barPadding)
  .style('fill', d => d.colour);
```

String variable => get margin top value

Create **x axis with call function domain**

tick line Class defined in css

Add g **element** to origin **class**

Define Bars

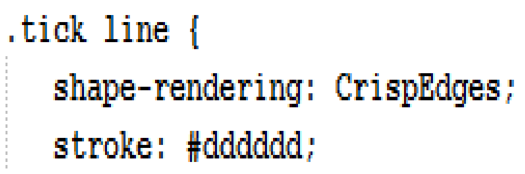
Bind (**selected year) name data** to bar

x position – off left horizontal **x axis scale**

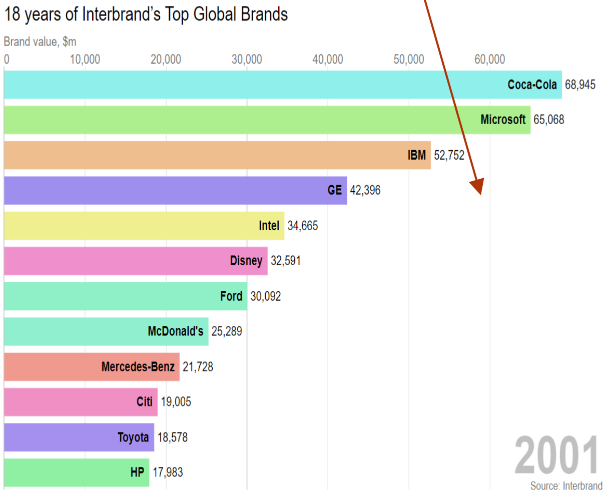
Bar width – based on **data value** (range of x scale)

y position – derived from **y scale** based on **rank** value (+5 px above bottom margin)

Output ranges from **y scale** - Get standard bar height



```
.tick line {
  shape-rendering: CrispEdges;
  stroke: #dddddd;
```

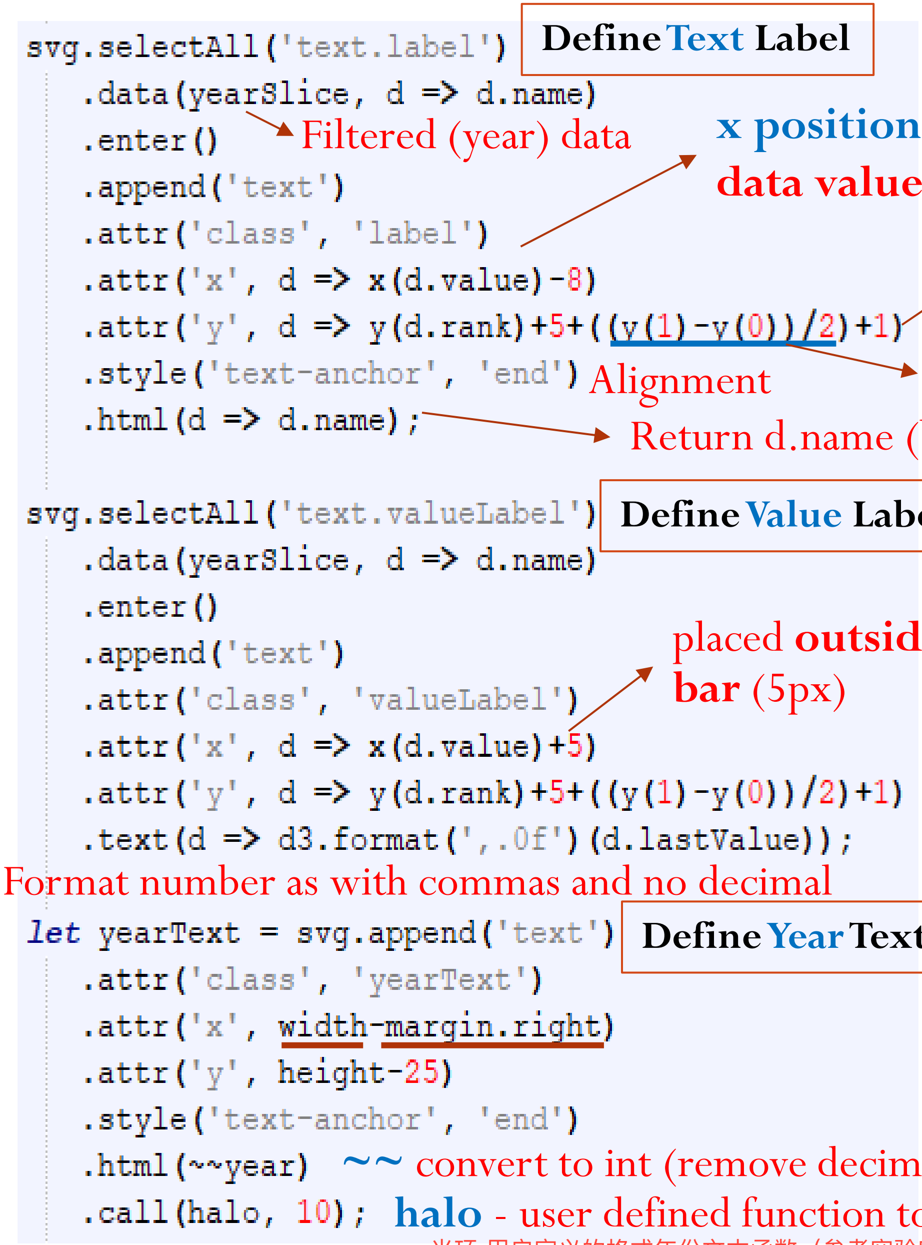


18 years of Interbrand's Top Global Brands

Source: Interbrand

Animation - Bar Chart Race

Define Labels



Define Text Label

```
svg.selectAll('text.label')
    .data(yearSlice, d => d.name)
    .enter()
    .append('text')
    .attr('class', 'label')
    .attr('x', d => x(d.value)-8)
    .attr('y', d => y(d.rank)+5+((y(1)-y(0))/2)+1)
    .style('text-anchor', 'end')
    .html(d => d.name);
```

Filtered (year) data

x position of text label – based on **data value** placed **within bar (-8px)**

y position of text label – based on rank and centered within each bar

Half of bar height

Alignment

Return d.name (brand) value to **html** as text

Define Value Label

```
svg.selectAll('text.valueLabel')
    .data(yearSlice, d => d.name)
    .enter()
    .append('text')
    .attr('class', 'valueLabel')
    .attr('x', d => x(d.value)+5)
    .attr('y', d => y(d.rank)+5+((y(1)-y(0))/2)+1)
    .text(d => d3.format(',.0f')(d.lastValue));
```

placed **outside bar** (5px)

Format number as with commas and no decimal

Define Year Text

```
let yearText = svg.append('text')
    .attr('class', 'yearText')
    .attr('x', width-margin.right)
    .attr('y', height-25)
    .style('text-anchor', 'end')
    .html(~~year)
    .call(halo, 10);
```

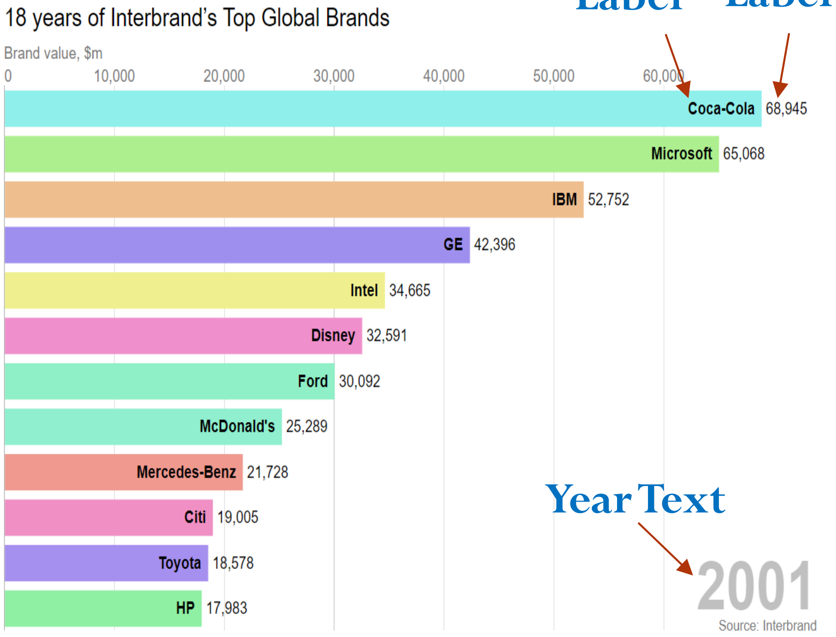
~~ convert to int (remove decimals) => shortcut for math.floor() function

halo - user defined function to **format yearText** (refer to end of lab codes)

光环-用户定义的格式年份文本函数（参考实验室代码的结尾）

Text Label

Value Label

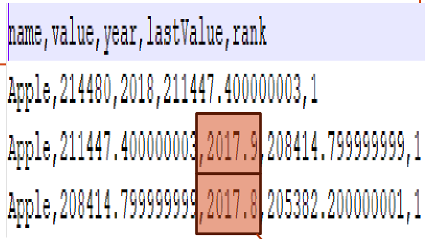


Year Text

Animation - Bar Chart Race

Create Animation

d3.interval() - **called after every given time interval or delay – looping function**.
If delay not given, delay equal to the timer. d3.interval(**callback, delay**);

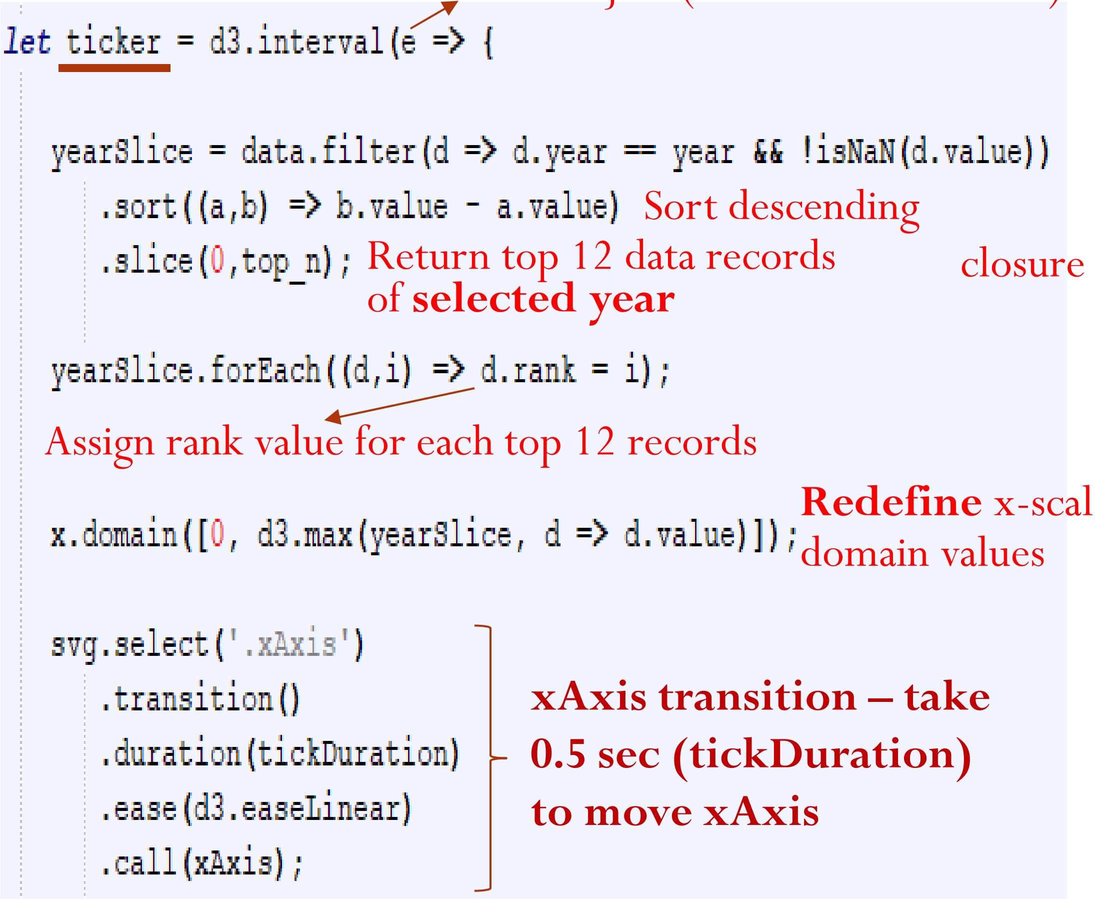


```
name,value,year,lastValue,rank
Apple,214480,2018,211447.40000003,1
Apple,211447.40000003,2017.9,208414.79999999,1
Apple,208414.79999999,2017.8,205382.20000001,1
```

callback: function executed after a particular delay.

delay: delay after which the function is executed.

e -> custom event object (return e for callback)



```
let ticker = d3.interval(e => {

  yearSlice = data.filter(d => d.year == year && !isNaN(d.value))
    .sort((a,b) => b.value - a.value)
    .slice(0,top_n);

  yearSlice.forEach((d,i) => d.rank = i);

  x.domain([0, d3.max(yearSlice, d => d.value)]);

  svg.select('.xAxis')
    .transition()
    .duration(tickDuration)
    .ease(d3.easeLinear)
    .call(xAxis);
```

Sort descending

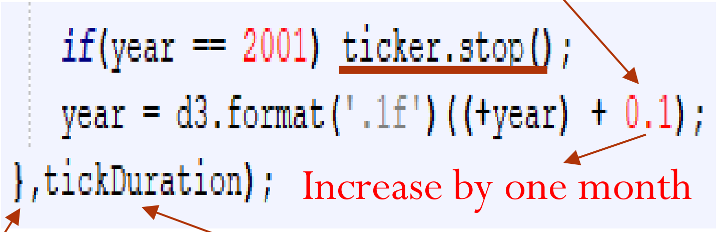
Return top 12 data records of **selected year**

Assign rank value for each top 12 records

Redefine x-scale domain values

xAxis transition – take 0.5 sec (tickDuration) to move xAxis

End of codes

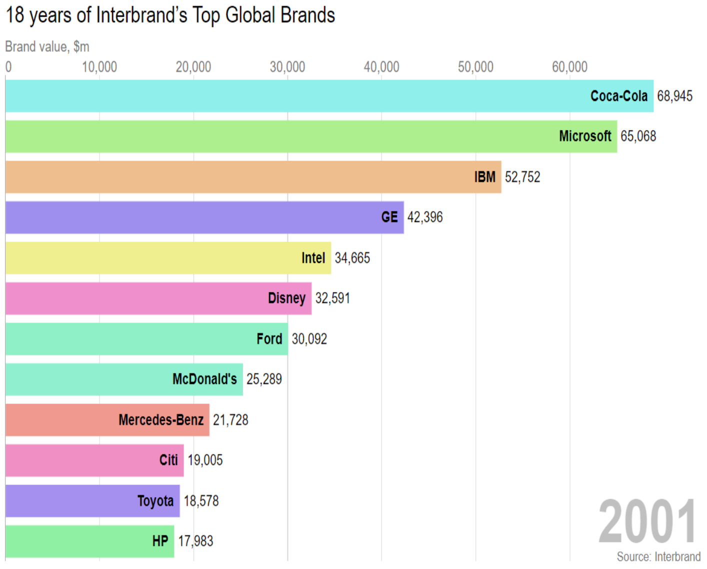


```
  if(year == 2001) ticker.stop();
  year = d3.format('.1f')((+year) + 0.1);
},tickDuration);
```

Increase by one month

closure for callback

Delay=500 (in front codes)

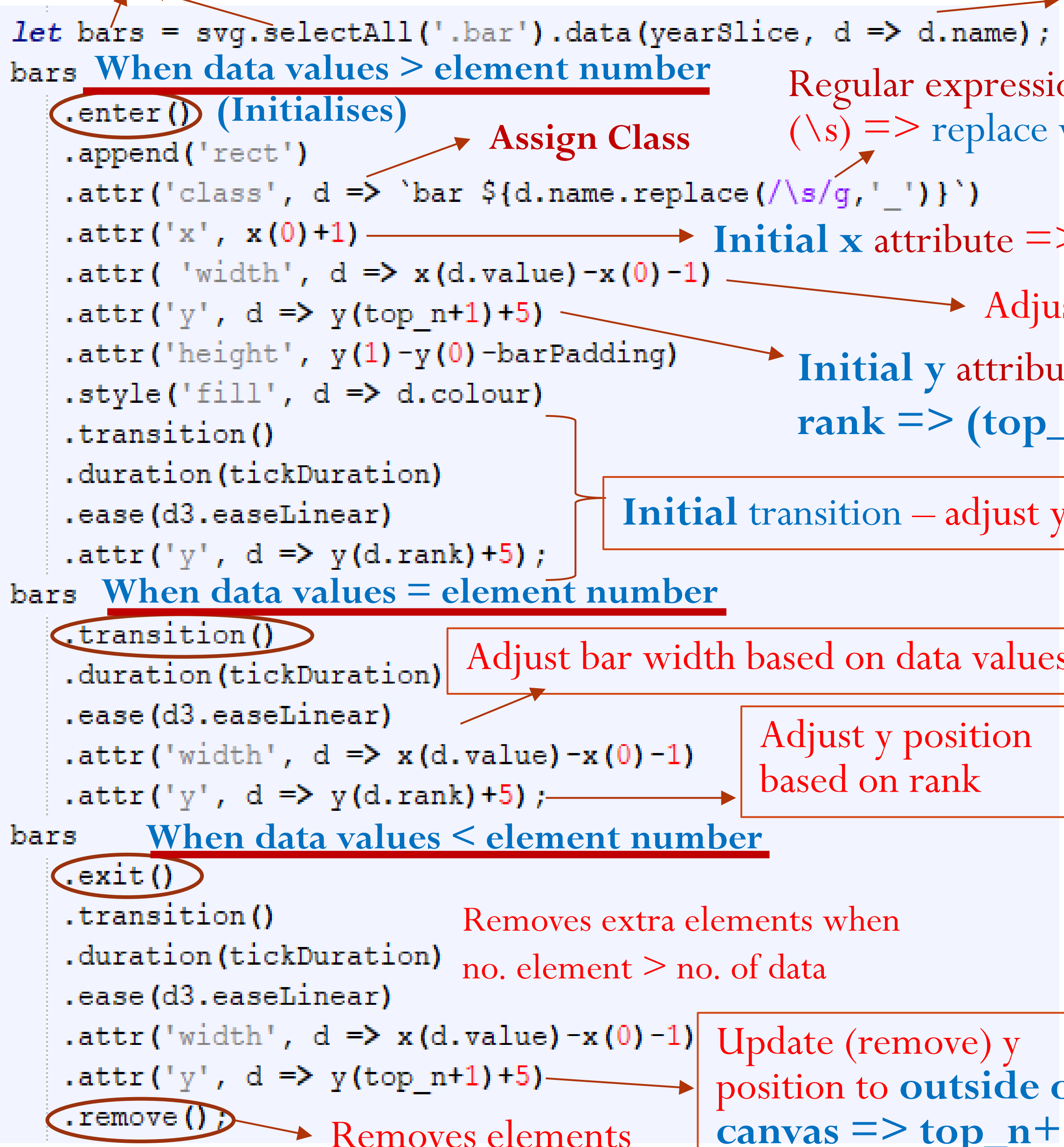


18 years of Interbrand's Top Global Brands

Source: Interbrand

Animation - Bar Chart Race

Animate the **Bars**



```
let bars = svg.selectAll('.bar').data(yearSlice, d => d.name);
bars
   .enter()
   .append('rect')
   .attr('class', d => `bar ${d.name.replace(/\s/g,'_')}`)
   .attr('x', x(0)+1)
   .attr( 'width', d => x(d.value)-x(0)-1)
   .attr('y', d => y(top_n+1)+5)
   .attr('height', y(1)-y(0)-barPadding)
   .style('fill', d => d.colour)
   .transition()
   .duration(tickDuration)
   .ease(d3.easeLinear)
   .attr('y', d => y(d.rank)+5);
bars
   .transition()
   .duration(tickDuration)
   .ease(d3.easeLinear)
   .attr('width', d => x(d.value)-x(0)-1)
   .attr('y', d => y(d.rank)+5);
bars
   .exit()
   .transition()
   .duration(tickDuration)
   .ease(d3.easeLinear)
   .attr('width', d => x(d.value)-x(0)-1)
   .attr('y', d => y(top_n+1)+5)
   .remove();
```

All bars will transition together

Return brand name data based on year

When data values > element number

(Initialises)

Assign Class

Regular expression – (/g) global search for whitespace (\s) => replace whitespace with '_'

Initial x attribute => x scale range output at 0 domain value

Adjust bar width based on data values

Initial y attribute => display **outside of canvas =>** **rank => (top_n)+1 => 12+1 =13**

Initial transition – adjust y position from outside to based on rank

When data values = element number

Adjust bar width based on data values

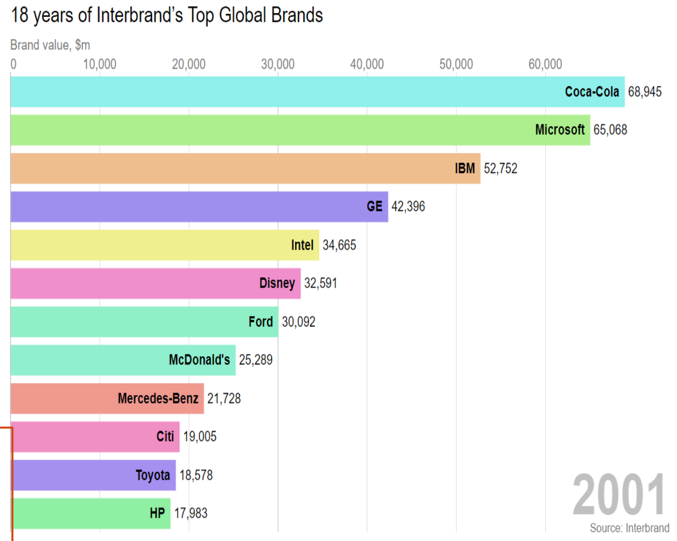
Adjust y position based on rank

When data values < element number

Removes extra elements when no. element > no. of data

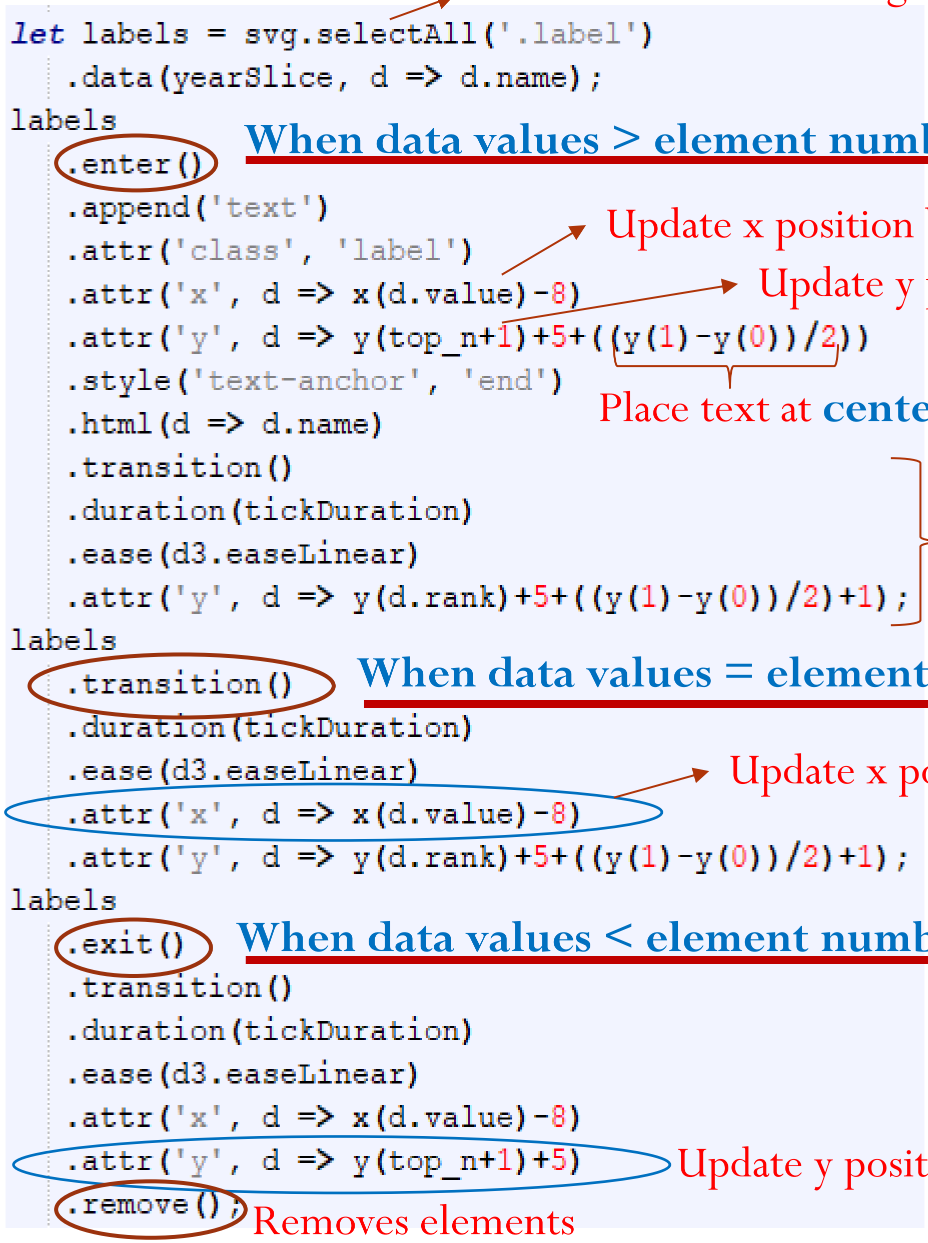
Update (remove) y position to **outside of canvas => top_n+1**

Removes elements



Animation - Bar Chart Race

Animate the Text Labels



```
let labels = svg.selectAll('.label')
    .data(yearSlice, d => d.name);
labels
    .enter()
    .append('text')
    .attr('class', 'label')
    .attr('x', d => x(d.value)-8)
    .attr('y', d => y(top_n+1)+5+((y(1)-y(0))/2))
    .style('text-anchor', 'end')
    .html(d => d.name)
    .transition()
    .duration(tickDuration)
    .ease(d3.easeLinear)
    .attr('y', d => y(d.rank)+5+((y(1)-y(0))/2)+1);
labels
    .transition()
    .duration(tickDuration)
    .ease(d3.easeLinear)
    .attr('x', d => x(d.value)-8)
    .attr('y', d => y(d.rank)+5+((y(1)-y(0))/2)+1);
labels
    .exit()
    .transition()
    .duration(tickDuration)
    .ease(d3.easeLinear)
    .attr('x', d => x(d.value)-8)
    .attr('y', d => y(top_n+1)+5)
    .remove();
```

All labels will transition together

When data values > element number (initialise)

Update x position based on **d.value inside of canvas (-8)**

Update y position to **outside of canvas (12+1)**

Place text at **center** of bar

Initial **transition** update y position based on **d.rank**

When data values = element number

Update x position within bar based on **d.value**

update y position based on **d.rank**

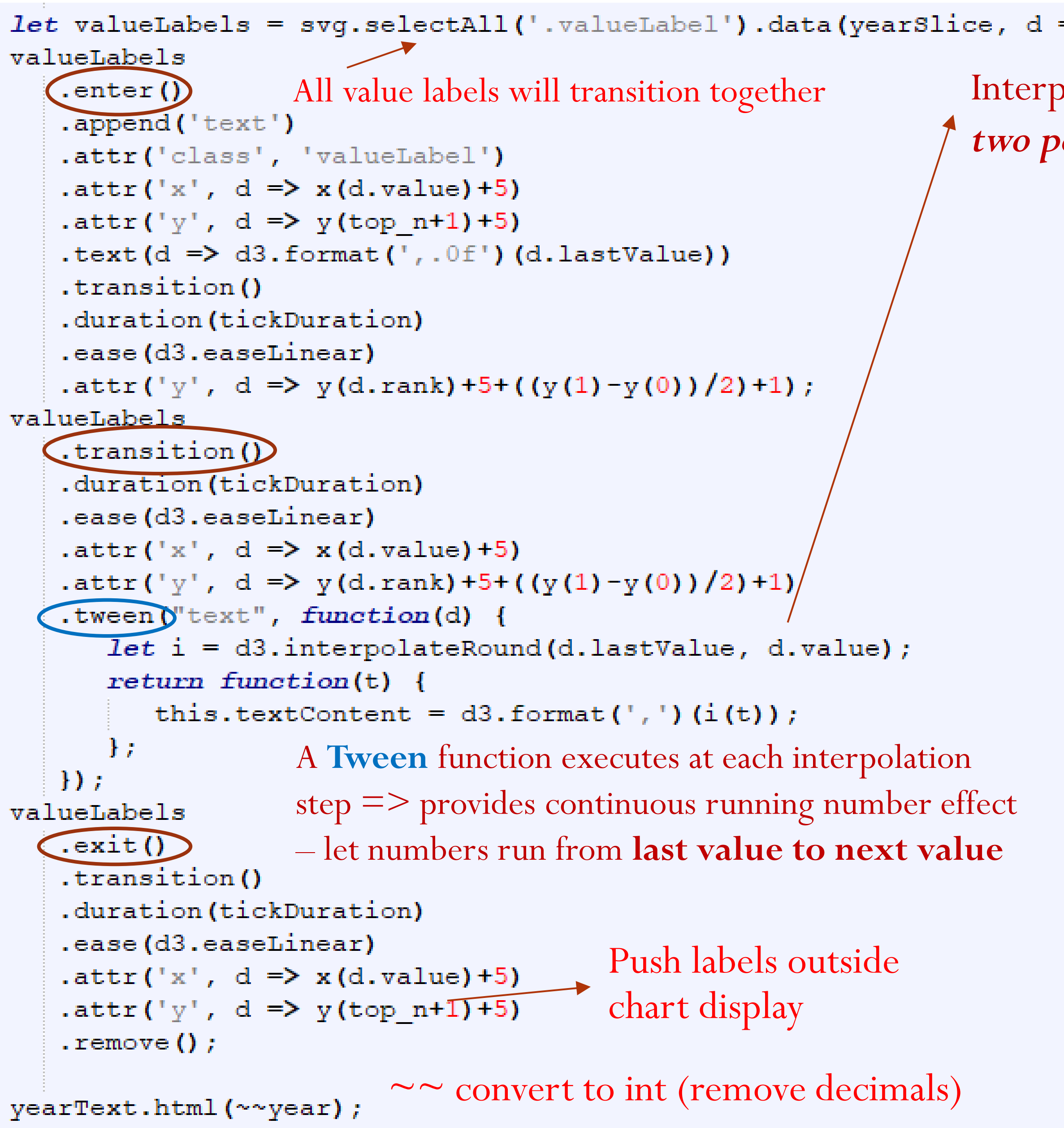
When data values < element number

Update y position to **outside of canvas**

Removes elements

Animation - Bar Chart Race

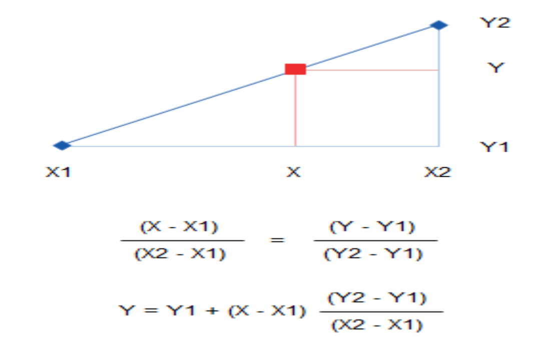
Animate the **Value Labels**



```
let valueLabels = svg.selectAll('.valueLabel').data(yearSlice, d => d.name);
valueLabels
   .enter()
   .append('text')
   .attr('class', 'valueLabel')
   .attr('x', d => x(d.value)+5)
   .attr('y', d => y(top_n+1)+5)
   .text(d => d3.format(',.0f')(d.lastValue))
   .transition()
   .duration(tickDuration)
   .ease(d3.easeLinear)
   .attr('y', d => y(d.rank)+5+((y(1)-y(0))/2)+1);
valueLabels
   .transition()
   .duration(tickDuration)
   .ease(d3.easeLinear)
   .attr('x', d => x(d.value)+5)
   .attr('y', d => y(d.rank)+5+((y(1)-y(0))/2)+1)
   .tween("text", function(d) {
      let i = d3.interpolateRound(d.lastValue, d.value);
      return function(t) {
         this.textContent = d3.format(',')(i(t));
      };
   });
valueLabels
   .exit()
   .transition()
   .duration(tickDuration)
   .ease(d3.easeLinear)
   .attr('x', d => x(d.value)+5)
   .attr('y', d => y(top_n+1)+5)
   .remove();

yearText.html(~~year);
```

All value labels will transition together

InterpolateRound => Get value between *two points (lastValue and value)*



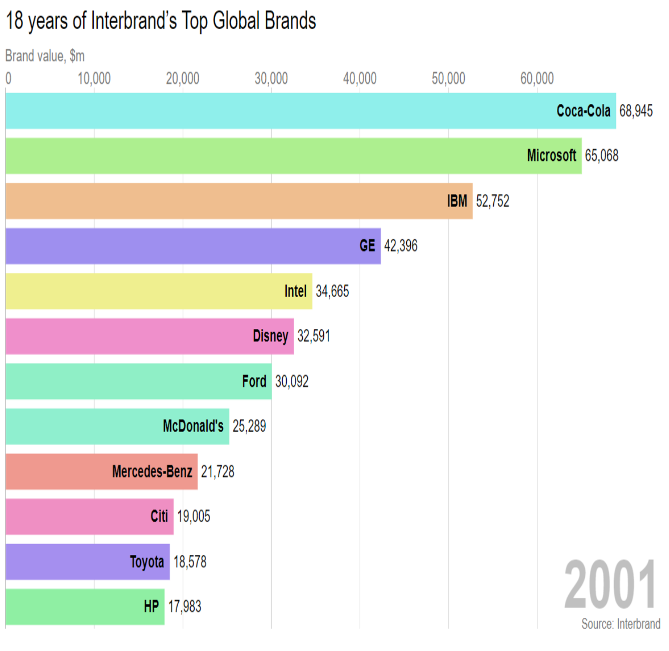
$$\frac{(X - X1)}{(X2 - X1)} = \frac{(Y - Y1)}{(Y2 - Y1)}$$

$$Y = Y1 + (X - X1)\frac{(Y2 - Y1)}{(X2 - X1)}$$

A **Tween** function executes at each interpolation step => provides continuous running number effect – let numbers run from **last value to next value**

Push labels outside chart display

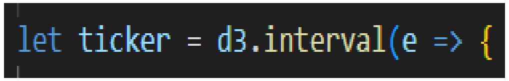
~~ convert to int (remove decimals)



Animation - Bar Chart Race

d3.**interval**(callback, delay)

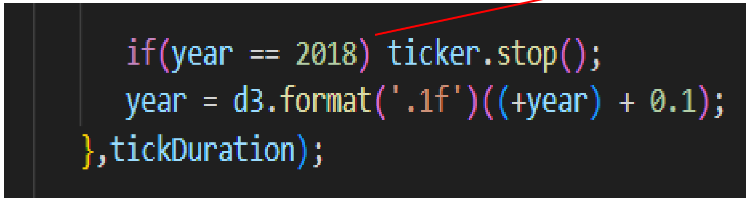
- **callback:** It is the function to be executed after a particular delay.
- **delay:** It is the delay after which the function is executed.



```
let ticker = d3.interval(e => {
```

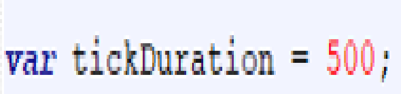
e -> custom event object (callback => return e)

Codes…



```
      if(year == 2018) ticker.stop();
      year = d3.format('.1f')((+year) + 0.1);
   },tickDuration);
```

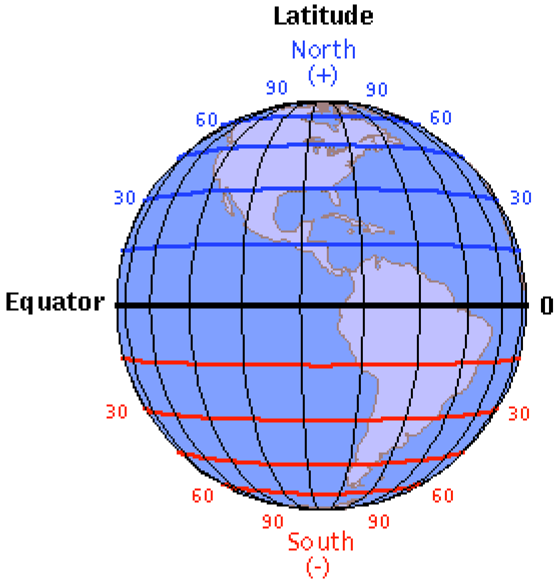
Stop condition

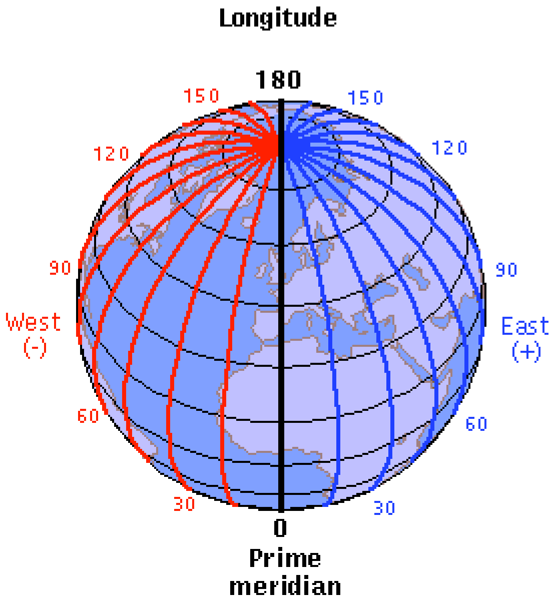


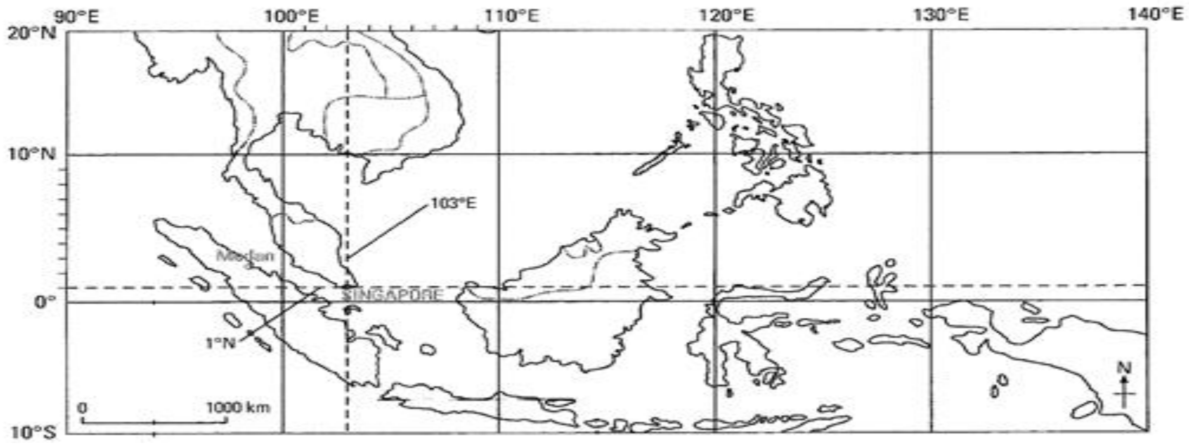
```
var tickDuration = 500;
```

Delay for the interval before starting a new loop.
Defined at the start

Geomapping

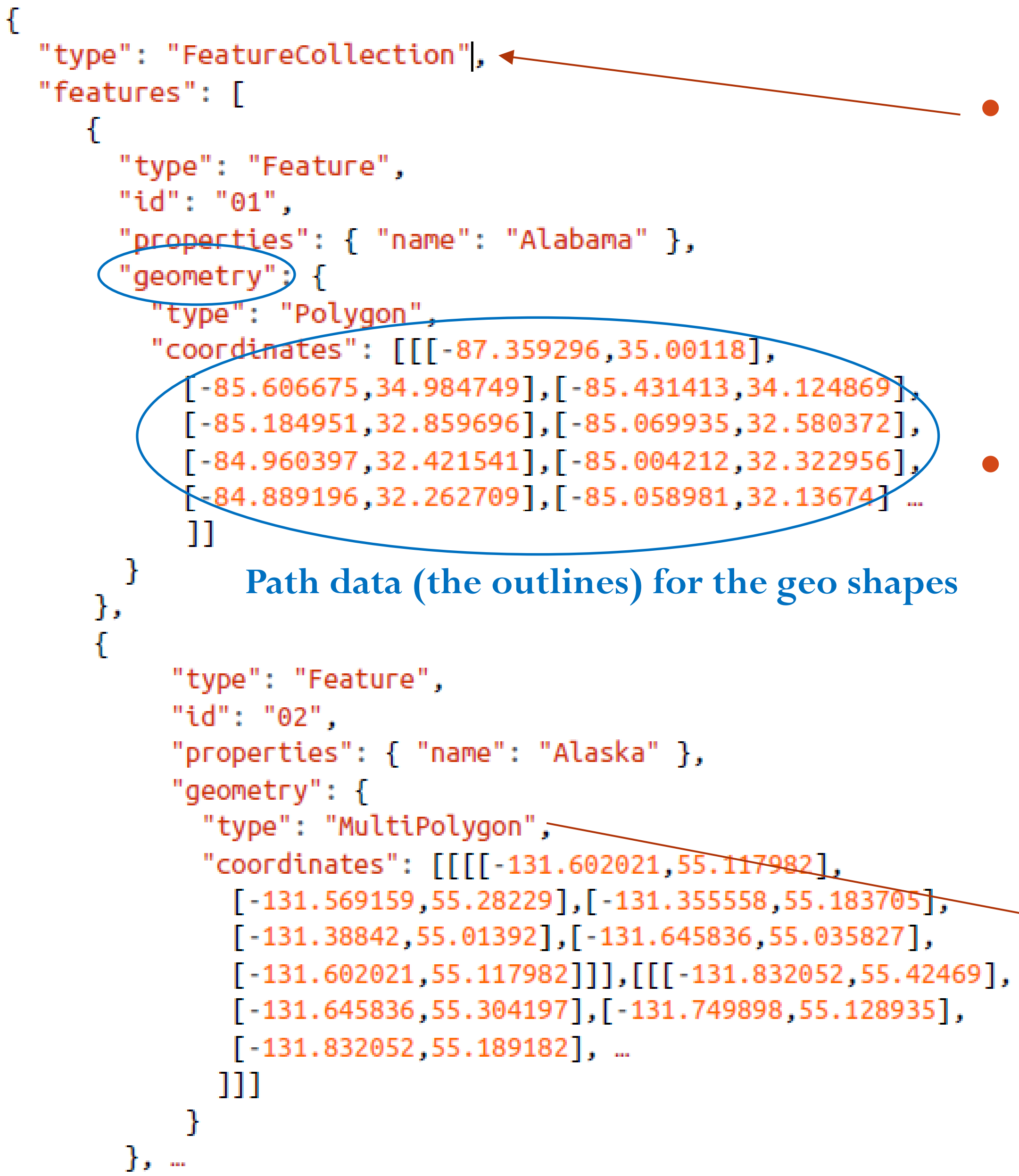

Latitude


Longitude



Singapore's Latitude and Longitude coordinates: [1N, 103E]

GeoJSON



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "id": "01",
      "properties": { "name": "Alabama" },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[-87.359296,35.00118],
          [-85.606675,34.984749],[-85.431413,34.124869],
          [-85.184951,32.859696],[-85.069935,32.580372],
          [-84.960397,32.421541],[-85.004212,32.322956],
          [-84.889196,32.262709],[-85.058981,32.13674] …
          ]]
      }
    },
    {
        "type": "Feature",
        "id": "02",
        "properties": { "name": "Alaska" },
        "geometry": {
          "type": "MultiPolygon",
          "coordinates": [[[[-131.602021,55.117982],
            [-131.569159,55.28229],[-131.355558,55.183705],
            [-131.38842,55.01392],[-131.645836,55.035827],
            [-131.602021,55.117982]]],[[[-131.832052,55.42469],
            [-131.645836,55.304197],[-131.749898,55.128935],
            [-131.832052,55.189182], …
           ]]]
        }
    }, …
```

Path data (the outlines) for the geo shapes

- **GeoJSON** is a JSON-based format for specifying geographic data.
- One giant object in curly brackets with type of **Feature Collection**, followed by **features** => array of individual **feature objects** representing a **US state**.
- The **geometry** object is where the type and coordinates that constitute the feature's boundary => sets of **longitude** and **latitude** array

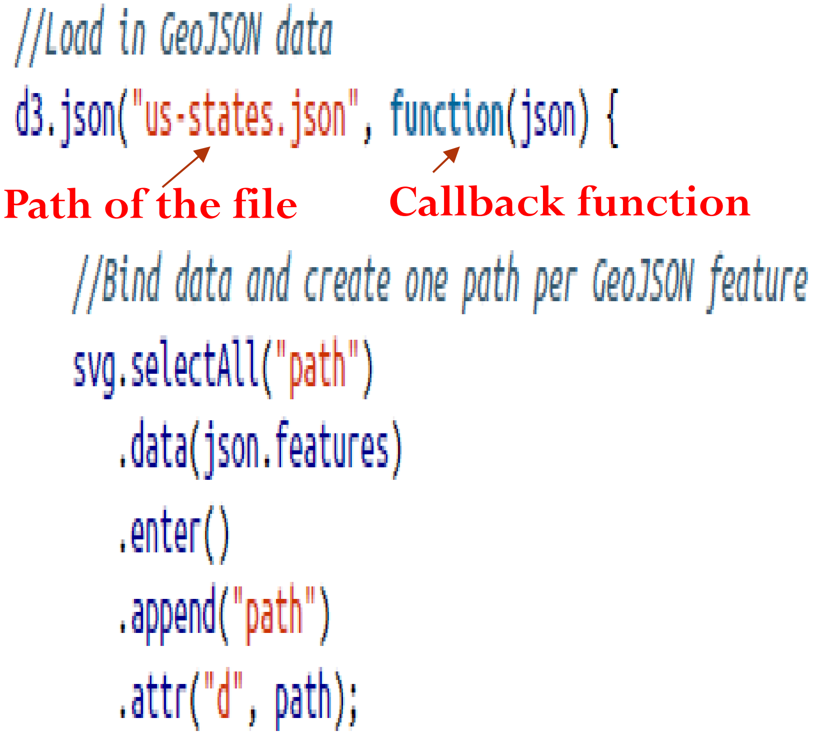
几何对象是构成特征边界的类型和坐标=>经度和纬度数组的集合

- **Note GeoJSON uses long/lat instead of lat/long**

Geomapping

d3.json() takes 2 arguments => string pointing to the **path of the file** and **callback function** that is called to load the JSON file.

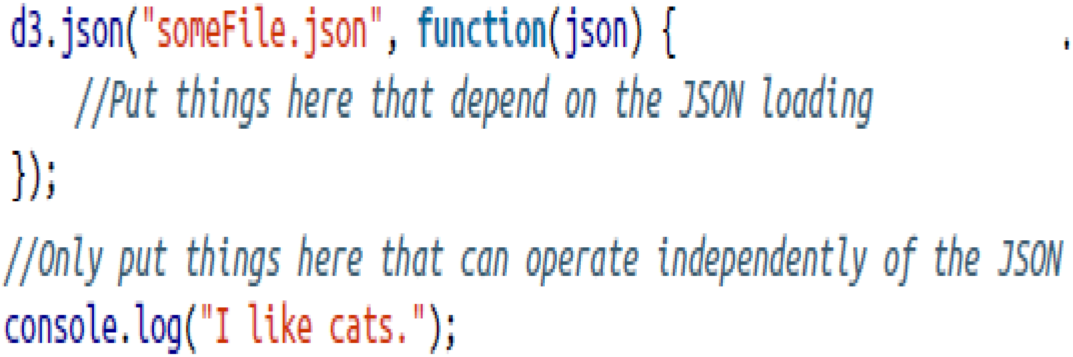
d3.json() *is* **asynchronous** => other codes will still run while the browser waits for file to load.



```
//Load in GeoJSON data
d3.json("us-states.json", function(json) {

    //Bind data and create one path per GeoJSON feature
    svg.selectAll("path")
       .data(json.features)
       .enter()
       .append("path")
       .attr("d", path);
```

Path of the file

Callback function

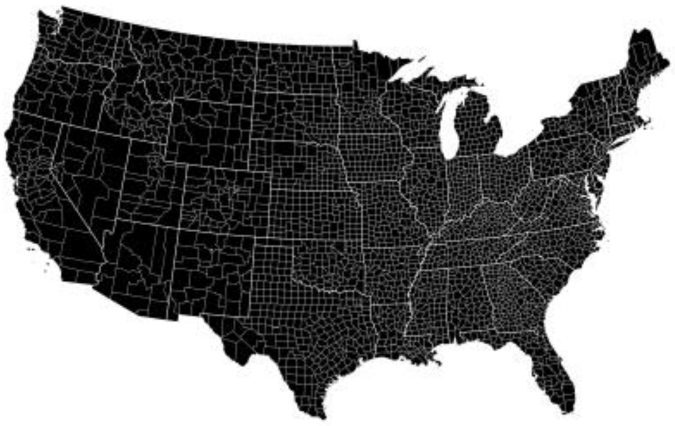


```
d3.json("someFile.json", function(json) {
    //Put things here that depend on the JSON loading
});
//Only put things here that can operate independently of the JSON
console.log("I like cats.");
```

Geomapping

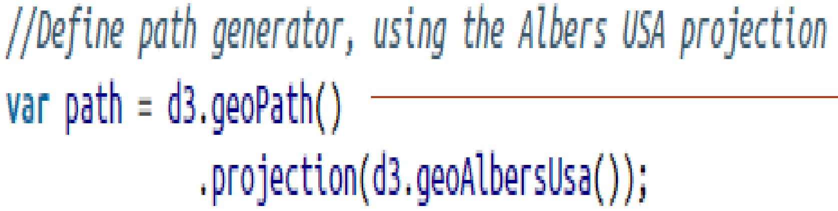
- To generate a geographic map in D3, the **path data (geometry of the outlines)** for the map shapes is required.
- D3 has three tools for geographic data:
  - **Paths** produce the final pixels
  - **Projections** turn sphere coordinates into Cartesian coordinates
  - **Streams** speed things up

⍰路径产生最终像素
⍰投影将球体坐标转换为笛卡尔坐标
⍰流加快速度



Projections - functions that convert from **longitude/latitude** co-ordinates to **x & y** co-ordinates

- **Geographic path generator:**



```
//Define path generator, using the Albers USA projection
var path = d3.geoPath()
                 .projection(d3.geoAlbersUsa());
```

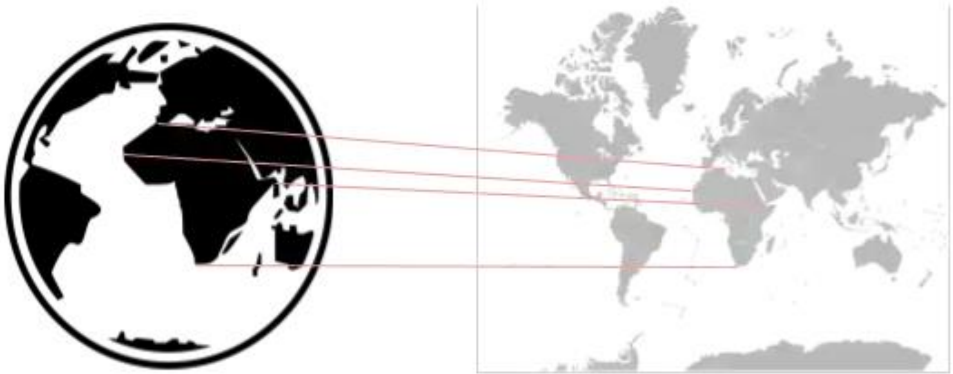
Geographic path generators (generate **SVG Path instructions** from GeoJSON data)

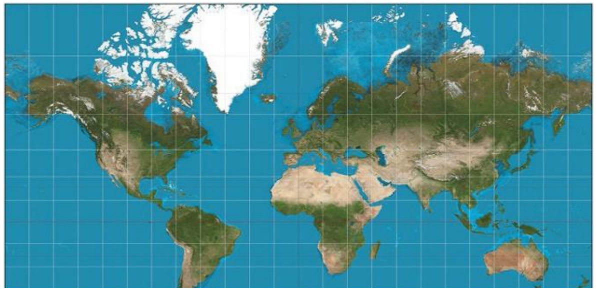
List of D3 projections - https://github.com/d3/d3-geo-projection

Map Projections

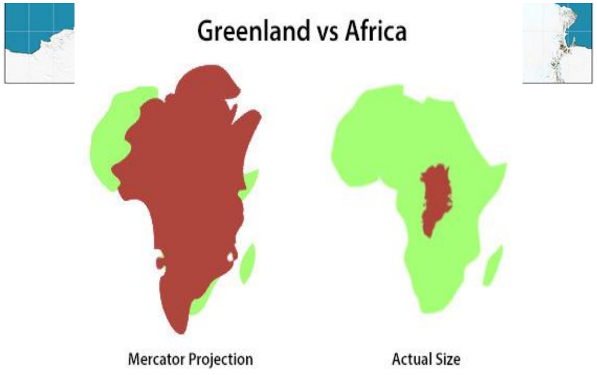
- **Map projection** flatten a **globe's** surface into a **plane** to make a map. This requires systematic transformation (math algorithm) of the **latitudes** and **longitudes** of locations from the surface of the globe into locations on a **2D plane**.
- Geradus **Mercator's** map **flattened the spherical surface** to make it **easier for navigation** => however, **distorts relative size of landmasses**, **exaggerate size of land near the poles** as compared to areas near the **equator**.

Mercator Projection





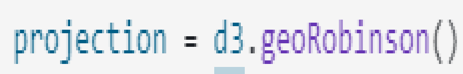
Over-exaggerate the size of landmasses near the poles



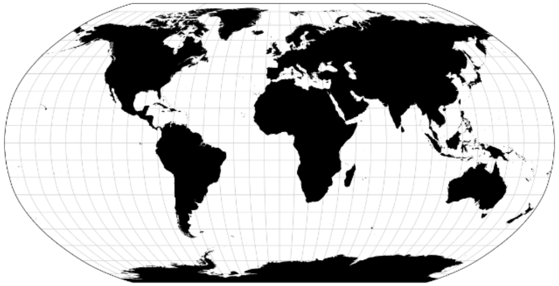
Geomapping

Robinson Projection

Area distortion grows with latitude and does not change with longitude.

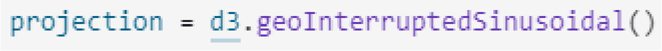


```
projection = d3.geoRobinson()
```

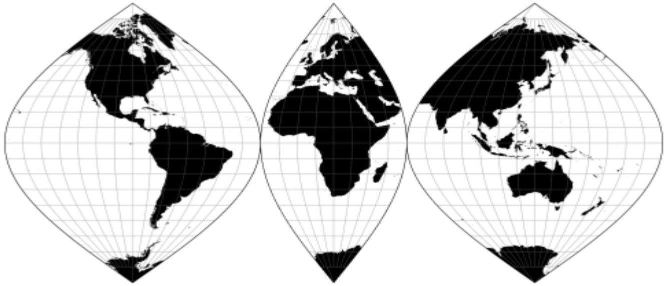


Interrupted Sinusoidal Projection

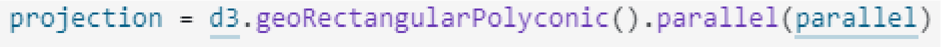
Uses asymmetrical lobe boundaries to emphasize land masses over oceans



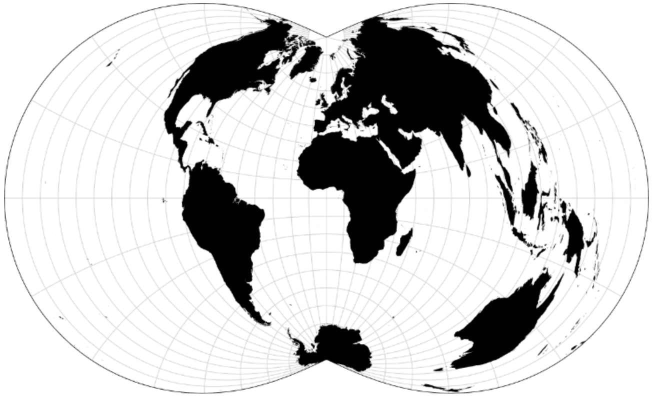
```
projection = d3.geoInterruptedSinusoidal()
```



Rectangular Polyconic Projection



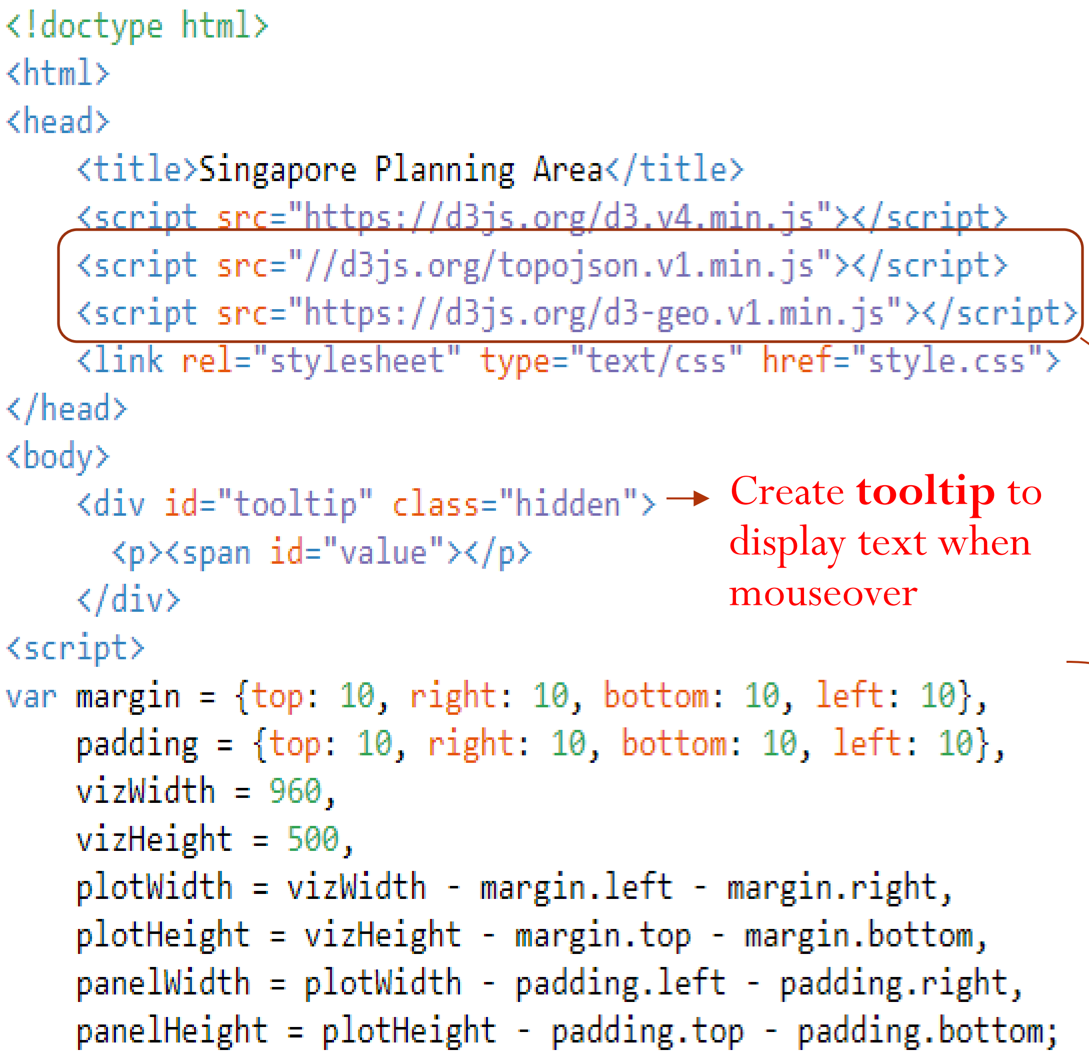
```
projection = d3.geoRectangularPolyconic().parallel(parallel)
```



Sometimes the rectangular polyconic is called the War Office projection due to its use by the **British War Office** for topographic maps.

It is not used much these days => why?

Geomapping - SGP

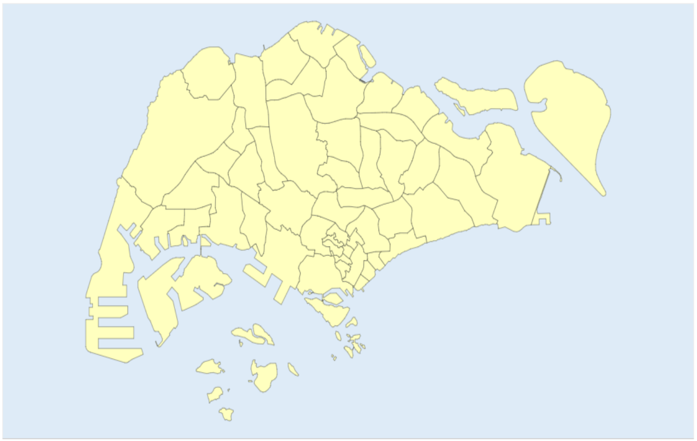


```
<!doctype html>
<html>
<head>
    <title>Singapore Planning Area</title>
    <script src="https://d3js.org/d3.v4.min.js"></script>
    <script src="//d3js.org/topojson.v1.min.js"></script>
    <script src="https://d3js.org/d3-geo.v1.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <div id="tooltip" class="hidden">
      <p><span id="value"></p>
    </div>
<script>
var margin = {top: 10, right: 10, bottom: 10, left: 10},
    padding = {top: 10, right: 10, bottom: 10, left: 10},
    vizWidth = 960,
    vizHeight = 500,
    plotWidth = vizWidth - margin.left - margin.right,
    plotHeight = vizHeight - margin.top - margin.bottom,
    panelWidth = plotWidth - padding.left - padding.right,
    panelHeight = plotHeight - padding.top - padding.bottom;
```

Create **tooltip** to display text when mouseover

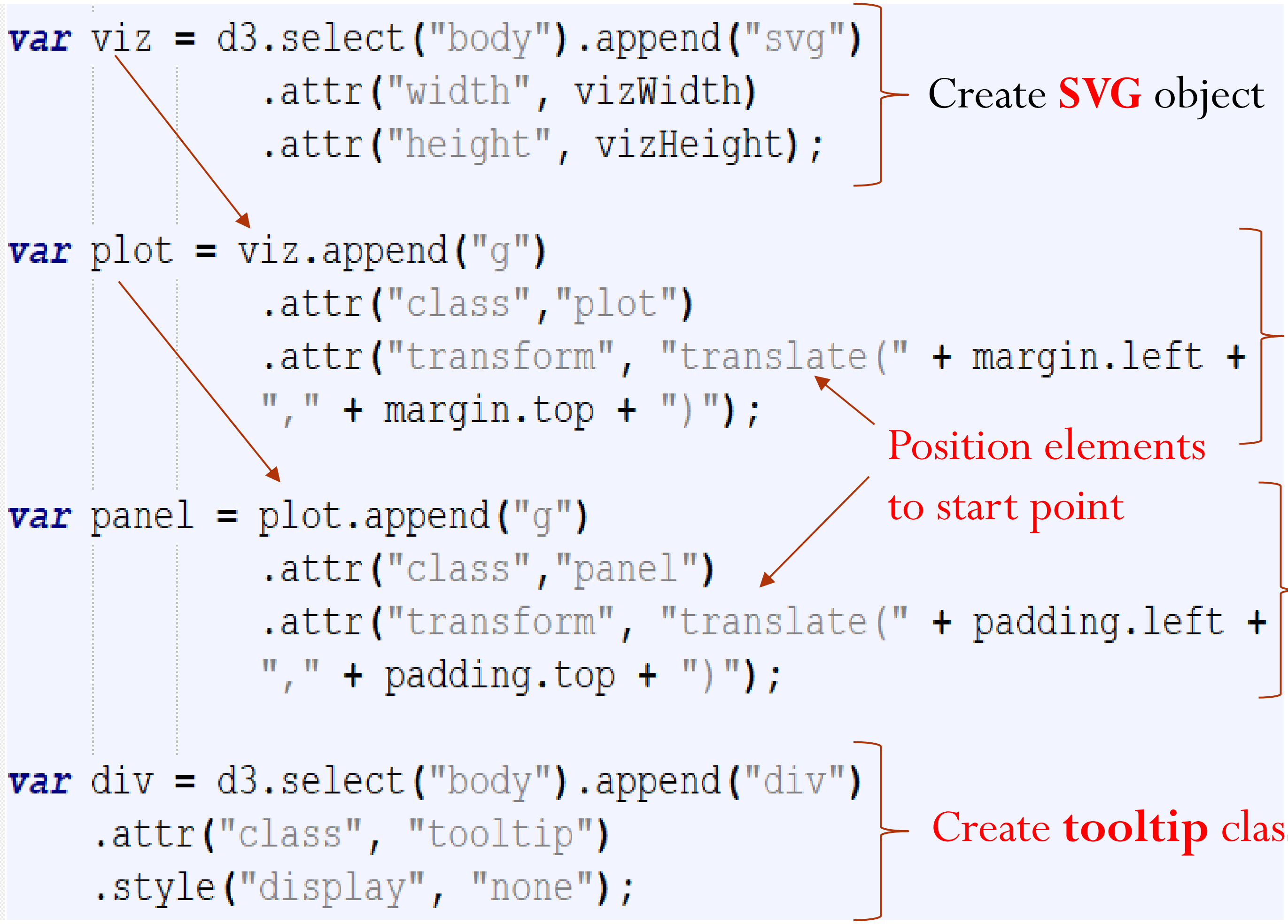
Set the **margins**, width and height

Singapore Planning Area



Additional libraries for geo functions

Geomapping - SGP



```
var viz = d3.select("body").append("svg")
            .attr("width", vizWidth)
            .attr("height", vizHeight);

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

var panel = plot.append("g")
            .attr("class","panel")
            .attr("transform", "translate(" + padding.left +
            "," + padding.top + ")");

var div = d3.select("body").append("div")
    .attr("class", "tooltip")
    .style("display", "none");
```

Create **SVG** object

Create **Plot** class

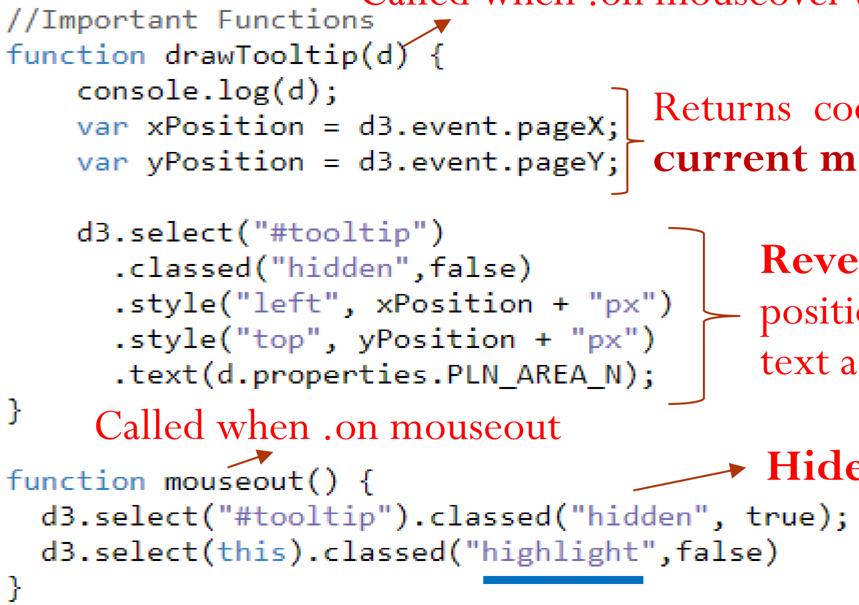
Position elements to start point

Create **Panel** class

Create **tooltip** class

Hide text

Geomapping - SGP



```
//Important Functions
function drawTooltip(d) {
    console.log(d);
    var xPosition = d3.event.pageX;
    var yPosition = d3.event.pageY;

    d3.select("#tooltip")
      .classed("hidden",false)
      .style("left", xPosition + "px")
      .style("top", yPosition + "px")
      .text(d.properties.PLN_AREA_N);
}

function mouseout() {
  d3.select("#tooltip").classed("hidden", true);
  d3.select(this).classed("highlight",false)
}
```

Called when .on mouseover (codes below)

Returns coordinates of **current mouse location**

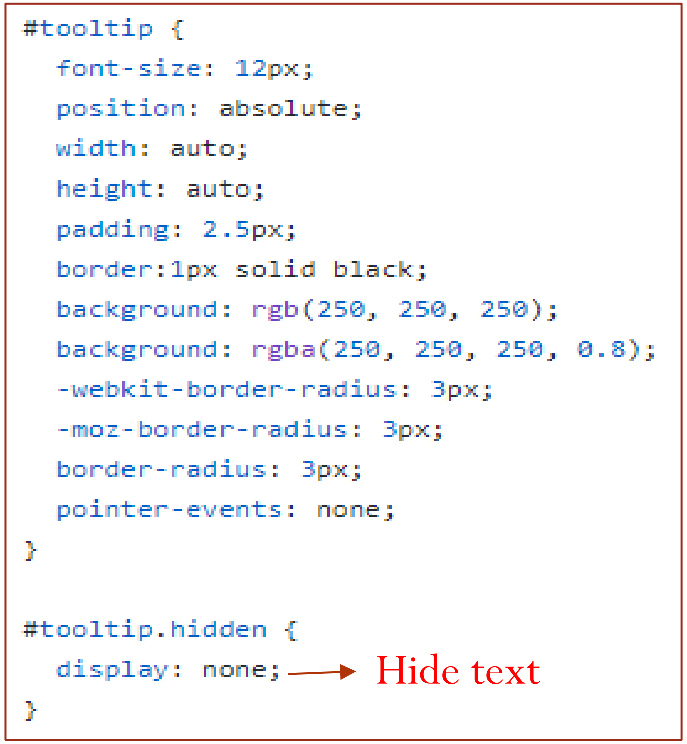
Reveal tooltip - position and display text as area name

Called when .on mouseout

Hide tooltip

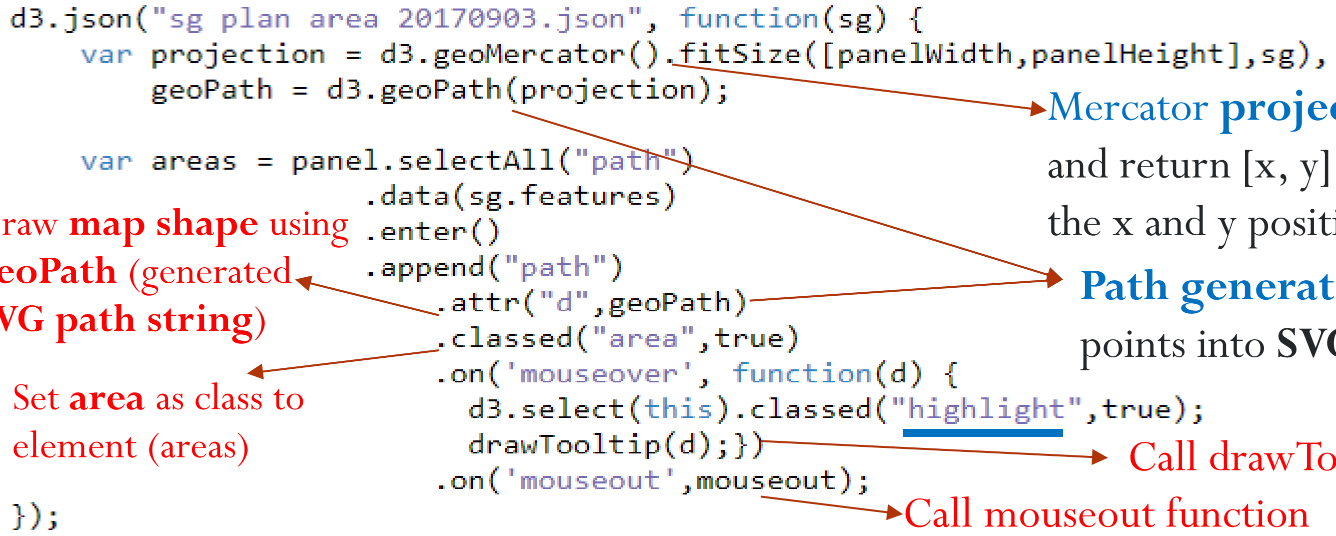
Style.css

id



```
#tooltip {
  font-size: 12px;
  position: absolute;
  width: auto;
  height: auto;
  padding: 2.5px;
  border:1px solid black;
  background: rgb(250, 250, 250);
  background: rgba(250, 250, 250, 0.8);
  -webkit-border-radius: 3px;
  -moz-border-radius: 3px;
  border-radius: 3px;
  pointer-events: none;
}

#tooltip.hidden {
  display: none;
}
```

Hide text



```
d3.json("sg plan area 20170903.json", function(sg) {
    var projection = d3.geoMercator().fitSize([panelWidth,panelHeight],sg),
        geoPath = d3.geoPath(projection);

    var areas = panel.selectAll("path")
                     .data(sg.features)
                     .enter()
                     .append("path")
                       .attr("d",geoPath)
                       .classed("area",true)
                       .on('mouseover', function(d) {
                         d3.select(this).classed("highlight",true);
                         drawTooltip(d);})
                       .on('mouseout',mouseout);
});
```

Mercator **projection** - pass [long, lat] point and return [x, y] point that corresponds to the x and y position drawn on SVG

Path generator converts x, y points into **SVG path string**

Draw **map shape** using **geoPath** (generated **SVG path string**)

Set **area** as class to element (areas)

Call drawTooltip function

Call mouseout function

Geomapping

- Open the **index.html** file in the **SingaporeGeoMap folder** to display the Singapore map and regional information.

Group Assignment

- Create a **storyboard** to visualize an open topic of choice. Style and objectives of storyboard is **open** to the group's creativity. No restrictions on number of pages/dashboards/charts/images. Students are to collect data from public domain sources. **D3.js must be used** to create the storyboard.

- Each group will have **15 mins** to present their work on **1st Nov** and the presentation should cover the problem statement, objectives, approaches/methods used to solve the problem, and **demo** of the working storyboard.

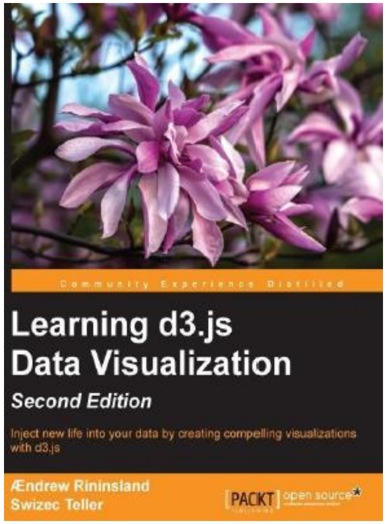
- Submit a min 1000 words report (no max limit) along with the HTML files, images, and data files => compress all files into a single zip file during submission on turnitin. Additionally, each group has to submit screenshots of their storyboard and upload to the **Group Assignment Peer Grading Discussion Forum** thread for voting and (**Team Leader**) to email me their top 3 votes. Submission Deadline – **01 Nov 2024, 23:59H**

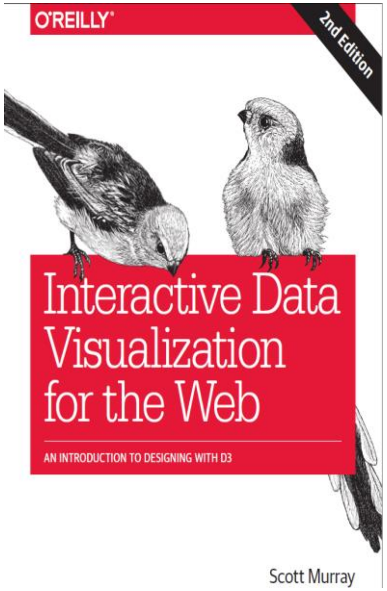
FUNG HAU YU	1
LI ZHUOMENG	1
WONG MANN JOE	1
SHANTHI RAMA	1
ANANYA BANERJEE	1
LIU JINGTAI	2
THINESH DHARAN RAHU	2
NIU YUEHAN	2
LIU MENGRU	2
LUM ENG KIT	2
ZHOU GAOQIANG	3
LIAO YUAN	3
CHIN JIA HUI	3
CHIA ANG SHEN	3
GUO KAIHUA	3
WEIXUAN	4
JU HYUNG CHA	4
FRANKY HALIM	4
WU DAN	4
EDWARD PARIWONO	4
JIANG MUROU	5
WANG ZECHAN	5
ZHANG XIWEN	5
ZHANG XINYUE	5
ZHOU QI	5
CHONG JIA ZHENG	6
OH HUAN LIN	6
ZHU JIAHUI	6
QUAH ZHENG JIE	6
TEO WEI SUEN	6
JUSTINE TAN JIA MIN	7
TAN HAN HUI	7
GUI WAN YING	7
MUHAMMAD AIMAN BIN ABDUL SAHAR	7
MOO KEE KHONG	7
TEO QI XIAN	8
TAN JUN YI	8
ZHU JIA RONG	8
QIN YU CHEN	8
MAH MUN CHOONG ALVIN	8
LIM QI WEN	9
CHUA JUNYONG SIMON	9
TAM YONG LIN	9
NICHOLAS MOK	9
MARCUS MOO WEI HAO	9
ZHOU QIANYU	10
TU XINYUE	10
LIU PEIWEN	10
LI PEIXUAN	10
HUANG YIJING	10
FU QIRUI	11
SHAO QI	11
XIE QINGLING	11
XU YIMU	11
XU ZIRAN	11

References

- Rininsland, A. (2016). Learning d3.js Data Visualization 2nd Ed. Packt Publishing.

- Murray, S.(2017) Interactive Data Visualization for the Web, 2nd Ed. O'Reilly.


Community Experience Distilled
Learning d3.js
Data Visualization
Second Edition
Inject new life into your data by creating compelling visualizations with d3.js
Ændrew Rininsland
Swizec Teller
PACKT open source


O'REILLY
2nd Edition
Interactive Data Visualization for the Web
AN INTRODUCTION TO DESIGNING WITH D3
Scott Murray