D3 Tutorial

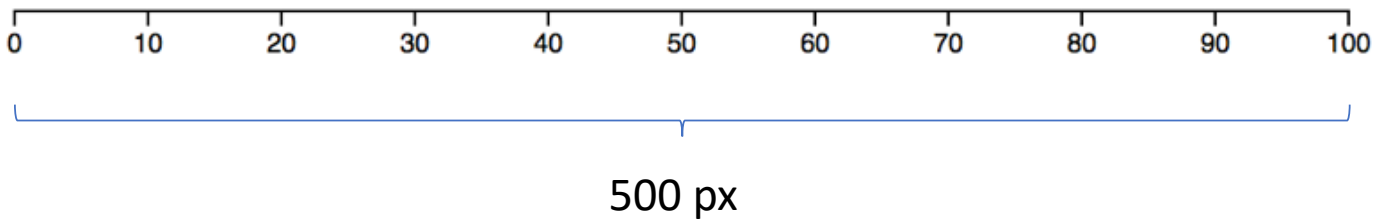
Axes and Shapes

Edit by Jiayi Xu and Han-Wei Shen, The Ohio State University

Axes - Create an axis

```
var xScale = d3.scaleLinear()
    .domain([0, 100])
    .range([0, 500]);

var xAxis = d3.axisBottom(xScale);
d3.select('svg')
    .append('g')
    .attr("transform", "translate(50, 50)")
    .call(xAxis);
```

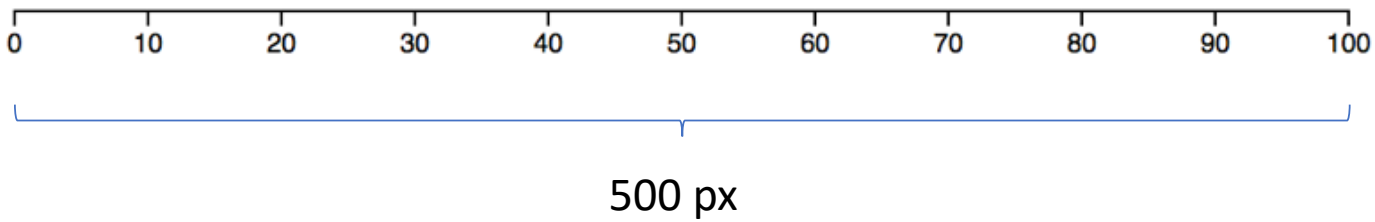
- Set the scale of the axis
  - The length of 100 units = 500 px



Axes - Create an axis

```
var xScale = d3.scaleLinear()
    .domain([0, 100])
    .range([0, 500]);

var xAxis = d3.axisBottom(xScale);
d3.select('svg')
    .append('g')
    .attr("transform", "translate(50, 50)")
    .call(xAxis);
```

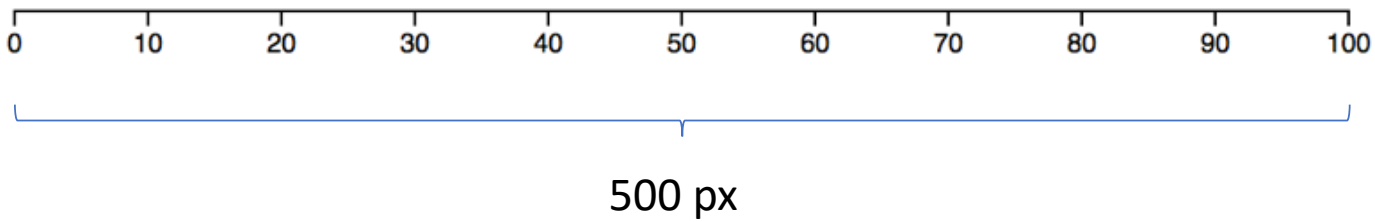
- Constructs a new **bottom-oriented** axis **generator** for the given scale



Axes - Create an axis

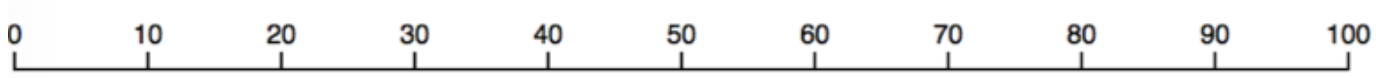
```
var xScale = d3.scaleLinear()
    .domain([0, 100])
    .range([0, 500]);

var xAxis = d3.axisBottom(xScale);
d3.select('svg')
    .append('g')
    .attr("transform", "translate(50, 50)")
    .call(xAxis);
```

- Render the axis on a *g* tag

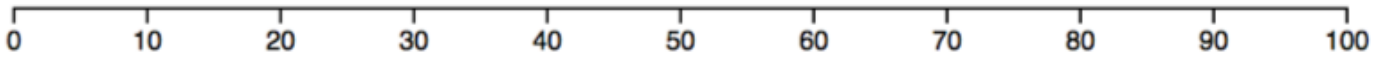


Axes - Orientation

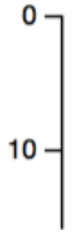
- d3.axisTop(*scale*)



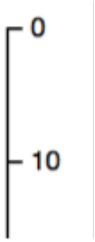
- d3.axisBottom(*scale*)



- d3.axisLeft(*scale*)



- d3.axisRight(*scale*)



Axes - Create an axis for population of cities

1. Data

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

2. Scale

```
var pop2width = d3.scaleLinear()
    .domain([0, 1.2*1e7])
    .range([0, 500]);
```

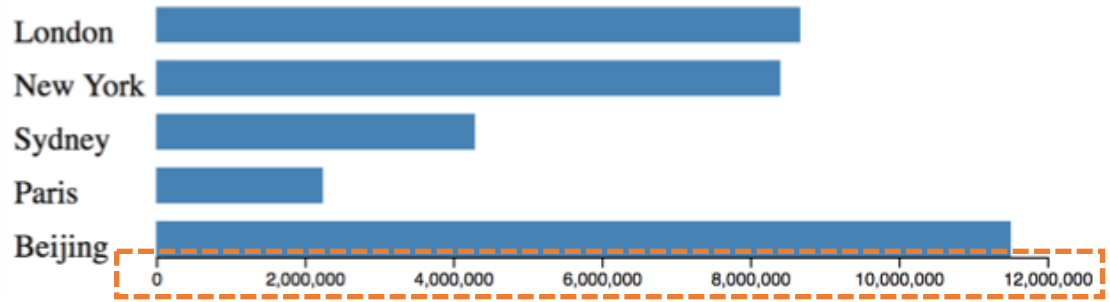
3. Axis

```
var xAxis = d3.axisBottom(pop2width)
              .ticks(6);
```

4. Render the axis

```
svg.append("g")
    .attr("transform",
        "translate(80, "
        + (barHeight + padding)*cities.length + ")")
    .call(xAxis);
```

5. Result



Give a suggested number of ticks

Lines – Line Generator

- lineGenerator is a function that accepts an array of co-ordinates and outputs a path data string

```
var lineGenerator = d3.line()
var points = [
    [0, 80],
    [100, 100],
    [200, 30],
    [300, 50],
    [400, 40],
    [500, 80]
];
var pathData = lineGenerator(points);
d3.select('path')
    .attr('d', pathData)
    .attr('fill', 'none')
    .attr('stroke', 'black');
```

- Constructs a new line generator

Lines – Line Generator

```
var lineGenerator = d3.line()
var points = [
    [0, 80],
    [100, 100],
    [200, 30],
    [300, 50],
    [400, 40],
    [500, 80]
];
var pathData = lineGenerator(points);
d3.select('path')
    .attr('d', pathData)
    .attr('fill', 'none')
    .attr('stroke', 'black');
```

- Define an array of coordinates

Lines – Line Generator

```
var lineGenerator = d3.line()
var points = [
    [0, 80],
    [100, 100],
    [200, 30],
    [300, 50],
    [400, 40],
    [500, 80]
];
var pathData = lineGenerator(points);
d3.select('path')
    .attr('d', pathData)
    .attr('fill', 'none')
    .attr('stroke', 'black');
```

- Now call lineGenerator, passing in our data points
- *pathData is "M0,80L100,100L200,30L300,50L400,40L500,80”*
  - A path string for SVG to draw a line

Lines – Line Generator

```
var lineGenerator = d3.line()
var points = [
    [0, 80],
    [100, 100],
    [200, 30],
    [300, 50],
    [400, 40],
    [500, 80]
];
var pathData = lineGenerator(points);
d3.select('path')
    .attr('d', pathData)
    .attr('fill', 'none')
    .attr('stroke', 'black');
```

- Draw the line

Lines – Curve

- Draw a curve
  - line.curve(curveType)

```
var lineGenerator = d3.line()
    .curve(d3.curveCardinal);
```

- Explore more curve types
  - http://bl.ocks.org/d3indepth/raw/b6d4845973089bc1012dec1674d3aff8/

Lines – Create a line chart

- Let’s create a line chart together!
- Data
  - Apple stock (AAPL) price from April 23rd, 2012 to May 1st, 2012

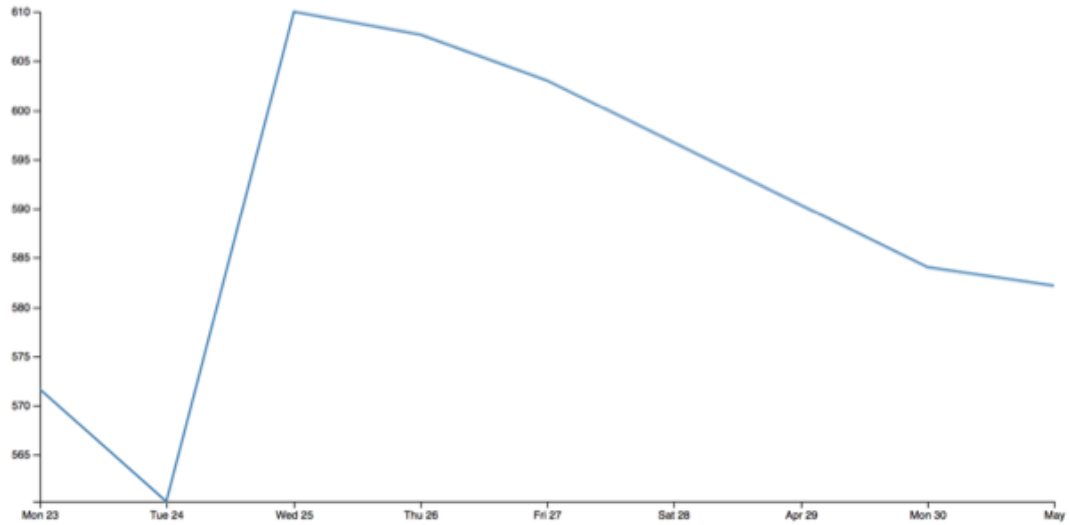
```
var parseTime = d3.timeParse("%d-%b-%y");
var data = [
    {'date': parseTime('23-Apr-12'), 'price': 571.70},
    {'date': parseTime('24-Apr-12'), 'price': 560.28},
    {'date': parseTime('25-Apr-12'), 'price': 610.00},
    {'date': parseTime('26-Apr-12'), 'price': 607.70},
    {'date': parseTime('27-Apr-12'), 'price': 603.00},
    {'date': parseTime('30-Apr-12'), 'price': 583.98},
    {'date': parseTime('1-May-12'), 'price': 582.13}
];
```

Lines – Create a line chart

- Scale
- xScale: Date to width
- yScale: Price to height

```
var xScale = d3.scaleTime()
    .domain(
        d3.extent(data, function(d) {
            return d.date;
        })
        )
    .rangeRound([0, width]);

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

- Final Result

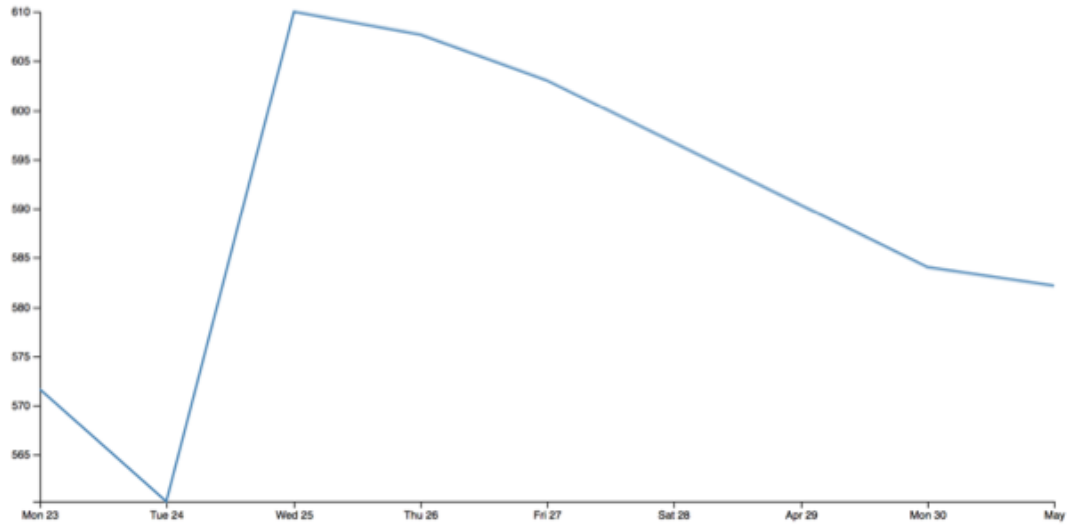


Lines – Create a line chart

- Line generator
  - Tell the generator how to map data [*date*, *price*] to coordinates [*x*, *y*]

```
var lineGenerator = d3.line()
    .x(function(d) {
        return xScale(d.date);
    })
    .y(function(d) {
        return yScale(d.price);
    });
```

- Final Result



Lines – Create a line chart

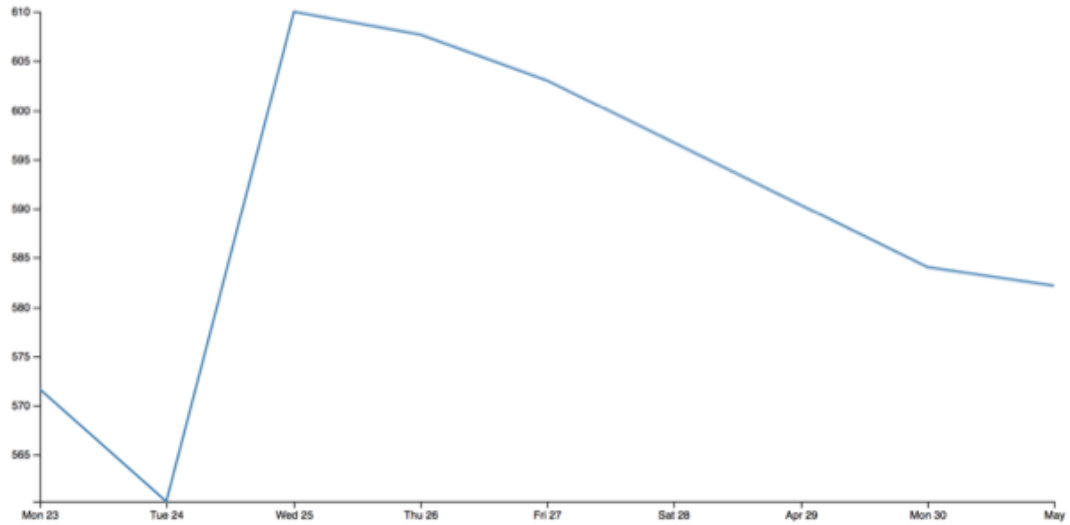
- Draw axes

```
g.append("g")
  .attr("transform", "translate(0," + height + ")")
  .call(d3.axisBottom(xScale));

g.append("g")
      .call(d3.axisLeft(yScale))
```

- Draw lines

```
g.append("path")
  .attr("fill", "none")
  .attr("stroke", "steelblue")
  .attr("stroke-width", 2)
  .attr("d", lineGenerator(data));
```

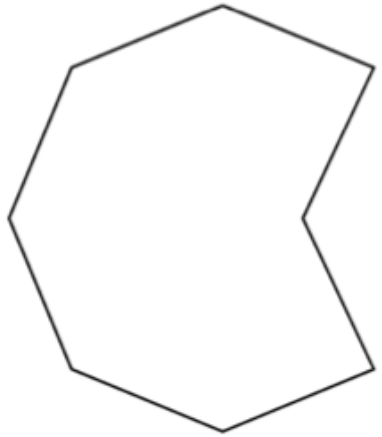
- Final Result



Lines – Radial Line d3.radialLine()

- The radial line generator is similar to the line generator but the points are formed by *angle* in radians (clockwise) and *radius*, rather than *x* and *y*
  - Data can be encoded into *angle* and *radius*
  - Application: Radar graphs

```
var radialLineGenerator = d3.radialLine();
var points = [
    [0, 80],
    [Math.PI * 0.25, 80],
    [Math.PI * 0.5, 30],
    [Math.PI * 0.75, 80],
    [Math.PI, 80],
    [Math.PI * 1.25, 80],
    [Math.PI * 1.5, 80],
    [Math.PI * 1.75, 80],
    [Math.PI * 2, 80]
];
d3.select('g')
    .append('path')
    .attr('d', radialLineGenerator(points))
    .attr('fill', 'none')
    .attr('stroke', 'black');
```



Area – d3.area()

- The area generator outputs path that defines an area between two lines.
  - Data can be encoded into *coordinates* on the two lines
  - Application: Stream graphs, filled line charts

```
var points = [
    {x: 0, y0: 30, y1: 80},
    {x: 100, y0: 80, y1: 100},
    {x: 200, y0: 20, y1: 30},
    {x: 300, y0: 20, y1: 50},
    {x: 400, y0: 10, y1: 40},
    {x: 500, y0: 50, y1: 80}
];
var areaGenerator = d3.area()
    .x(function(d) {
        return d.x;
    })
    .y0(function(d) {
        return d.y0;
    })
    .y1(function(d) {
        return d.y1;
    });
d3.select('g')
    .append('path')
    .attr('d', areaGenerator(points))
    .attr('fill', 'lightgrey');
```

Area – Radial Area d3.radialArea()

- The radial area generator is similar to the area generator but the points are formed by *angle* in radians (clockwise) and *radius*, rather than *x* and *y*
  - Data can be encoded into *angle* and *radius*
  - Application: Filled radar graphs

```
var points = [
    {angle: 0, r0: 20, r1: 80},
    {angle: Math.PI * 0.25, r0: 20, r1: 40},
    {angle: Math.PI * 0.5, r0: 20, r1: 80},
    {angle: Math.PI * 0.75, r0: 20, r1: 40},
    {angle: Math.PI, r0: 20, r1: 80},
    {angle: Math.PI * 1.25, r0: 20, r1: 40},
    {angle: Math.PI * 1.5, r0: 20, r1: 80},
    {angle: Math.PI * 1.75, r0: 20, r1: 40},
    {angle: Math.PI * 2, r0: 20, r1: 80}
];
```

```
var radialAreaGenerator = d3.radialArea()
    .angle(function(d) {
        return d.angle;
    })
    .innerRadius(function(d) {
        return d.r0;
    })
    .outerRadius(function(d) {
        return d.r1;
    });
d3.select('g')
    .append('path')
    .attr('d', radialAreaGenerator(points))
    .attr('fill', 'lightgrey');
```

Arc - d3.arc()

- Arc generators produce path data from *angle* and *radius* values
  - Data can be encoded into *angle* and *radius*
  - Application: Pie Chart

```
var data = {
  startAngle: 0,
  endAngle: 0.25 * Math.PI,
  innerRadius: 50,
  outerRadius: 100
};

var arcGenerator = d3.arc();

d3.select('g')
    .append('path')
    .attr('d', arcGenerator(data))
    .attr('fill', 'orange');
```

Arc – Multiple arcs

- Multiple arcs
  - A template of pie chart

```
var arcData = [
    {label: 'A', startAngle: 0, endAngle: 0.2},
    {label: 'B', startAngle: 0.2, endAngle: 0.6},
    {label: 'C', startAngle: 0.6, endAngle: 1.4},
    {label: 'D', startAngle: 1.4, endAngle: 3},
    {label: 'E', startAngle: 3, endAngle: 2* Math.PI}
];

var arcGenerator = d3.arc()
    .innerRadius(20)
    .outerRadius(100);

d3.select('g')
    .selectAll('path')
    .data(arcData)
    .enter()
    .append('path')
    .attr('d', arcGenerator);
```

Symbols - d3.symbol()

- The symbol generator produces path data for symbols

```
var symbolGenerator = d3.symbol()
    .size(80)
    .type(d3.symbolStar);

d3.select('g')
    .append('path')
    .attr('transform', 'translate(20, 20)')
    .attr('d', symbolGenerator());
```

  - position
    - We can use *transform* to set coordinates
  - types