



Basic Shape

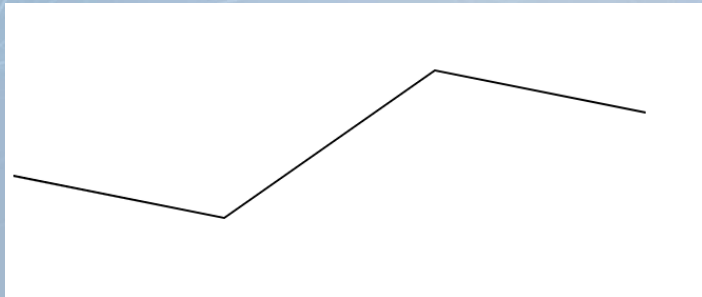
Data Visualization

D3 Shape Document

- <https://github.com/d3/d3-shape/tree/v2.1.0>

Line – Line Generator (Ex05-01)

- To have lines/curve on the svg, you need “path”
- **d3.line()** can generate path format for us



```
var points = [  
  [0, 80],  
  [100, 100],  
  [200, 30],  
  [300, 50],  
  [400, 40],  
  [500, 80],  
];
```

Unit: pixel

```
var lineGenerator = d3.line();  
//var lineGenerator = d3.line().curve(d3.curveCardinal);  
var pathData = lineGenerator(points);  
console.log(pathData);  
const svg = d3.select("#chart-area").append("svg")  
  .append('path')  
  .attr('d', pathData)  
  .attr('fill', 'none')  
  .attr('stroke', 'black');
```

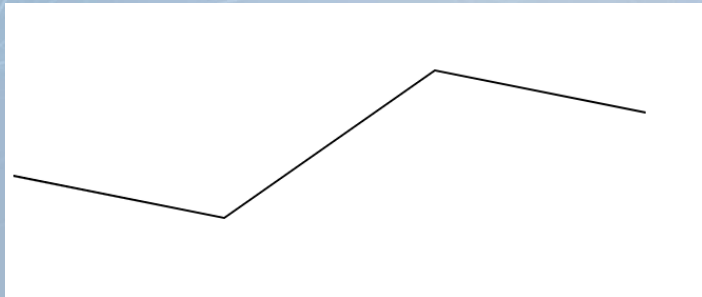
Line – Line Generator (Ex05-01)

- Create a line generator by `d3.line()`
- Path the x, y points to it. It will return 'path format' to you

```
M0,80L100,100L200,30L300,50L400,40L500,80
```

>

These two lines can be replaced by
`var pathData = d3.line()(points)`

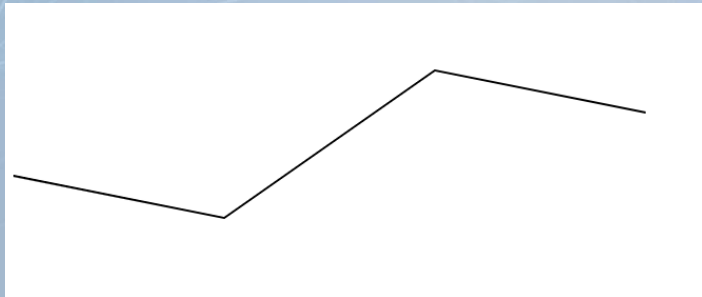


```
var points = [
  [0, 80],
  [100, 100],
  [200, 30],
  [300, 50],
  [400, 40],
  [500, 80],
];

var lineGenerator = d3.line();
//var lineGenerator = d3.line().curve(d3.curveCardinal);
var pathData = lineGenerator(points);
console.log(pathData);
const svg = d3.select("#chart-area").append("svg")
  .append('path')
  .attr('d', pathData)
  .attr('fill', 'none')
  .attr('stroke', 'black');
```

Line – Line Generator (Ex05-01)

- We draw a 'path' on svg by assigning 'pathData' to 'd' attribute of the path



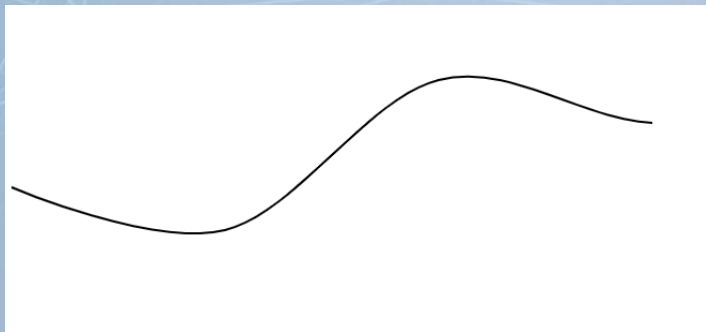
```
var points = [
  [0, 80],
  [100, 100],
  [200, 30],
  [300, 50],
  [400, 40],
  [500, 80],
];

var lineGenerator = d3.line();
//var lineGenerator = d3.line().curve(d3.curveCardinal);
var pathData = lineGenerator(points);
console.log(pathData);

const svg = d3.select("#chart-area").append("svg")
  .append('path')
  .attr('d', pathData)
  .attr('fill', 'none')
  .attr('stroke', 'black');
```

Line – Line Generator (Ex05-01)

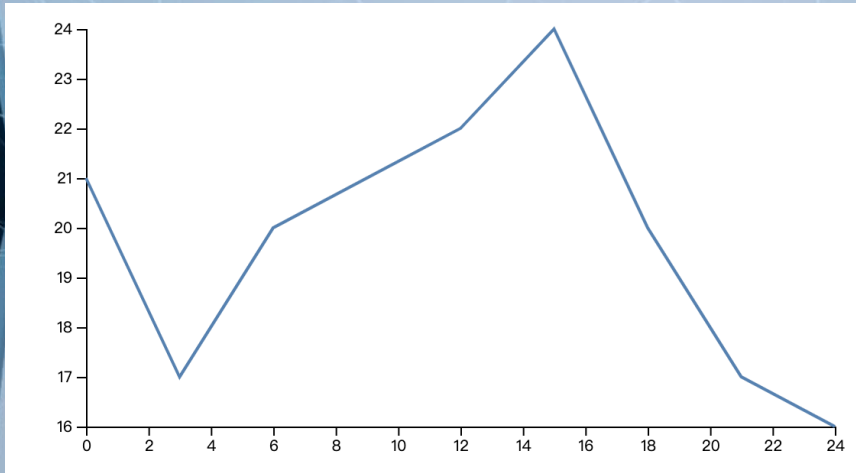
- What if you want a 'curve'
 - `d3.line().curve(curveType)`
 - Example: `curveType` -> `d3.curveCardinal`
 - More: <http://bl.ocks.org/d3indepth/raw/b6d4845973089bc1012dec1674d3aff8/>



```
var points = [  
  [0, 80],  
  [100, 100],  
  [200, 30],  
  [300, 50],  
  [400, 40],  
  [500, 80],  
];  
  
var lineGenerator = d3.line();  
//var lineGenerator = d3.line().curve(d3.curveCardinal);  
var pathData = lineGenerator(points);  
console.log(pathData);  
const svg = d3.select("#chart-area").append("svg")  
  .append('path')  
  .attr('d', pathData)  
  .attr('fill', 'none')  
  .attr('stroke', 'black');
```


Ex05-02

- Draw a line chart by data (the unit of the data is not pixels)



```
var data = [  
  {'hour': 0, 'temperature': 21},  
  {'hour': 3, 'temperature': 17},  
  {'hour': 6, 'temperature': 20},  
  {'hour': 9, 'temperature': 21},  
  {'hour': 12, 'temperature': 22},  
  {'hour': 15, 'temperature': 24},  
  {'hour': 18, 'temperature': 20},  
  {'hour': 21, 'temperature': 17},  
  {'hour': 24, 'temperature': 16},  
];
```

Ex05-02

- main.js
 - Set the layout of the plotting area
 - Append a svg
 - Create x, y scale function
 - Use the x and y scale function to append the x and y axis
- The above steps are the same as Ex04-16

```
const MARGIN = { LEFT: 100, RIGHT: 10, TOP: 10, BOTTOM: 130 }
const WIDTH = 600 - MARGIN.LEFT - MARGIN.RIGHT
const HEIGHT = 400 - MARGIN.TOP - MARGIN.BOTTOM

const svg = d3.select("#chart-area").append("svg")
    .attr("width", WIDTH + MARGIN.LEFT + MARGIN.RIGHT)
    .attr("height", HEIGHT + MARGIN.TOP + MARGIN.BOTTOM)

const g = svg.append("g")
    .attr("transform", `translate(${MARGIN.LEFT}, ${MARGIN.TOP})`)

var xScale = d3.scaleLinear()
    .domain(d3.extent(data, d=>d.hour))
    .range([0, WIDTH]);

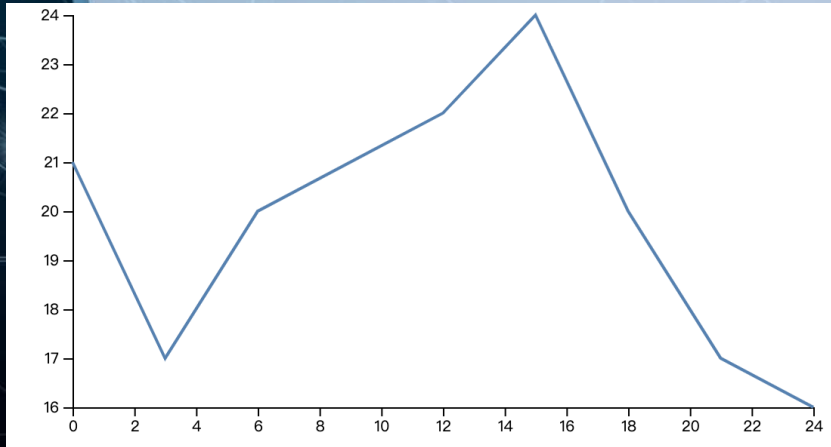
var yScale = d3.scaleLinear()
    .domain(d3.extent(data, d=>d.temperature))
    .range([HEIGHT, 0]);

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

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


Ex05-02

- main.js
- When creating the line generator, we use `.x()` and `.y()` to indicate how to get the x locations and y locations in pixel if the data is passed to the line generator **later**

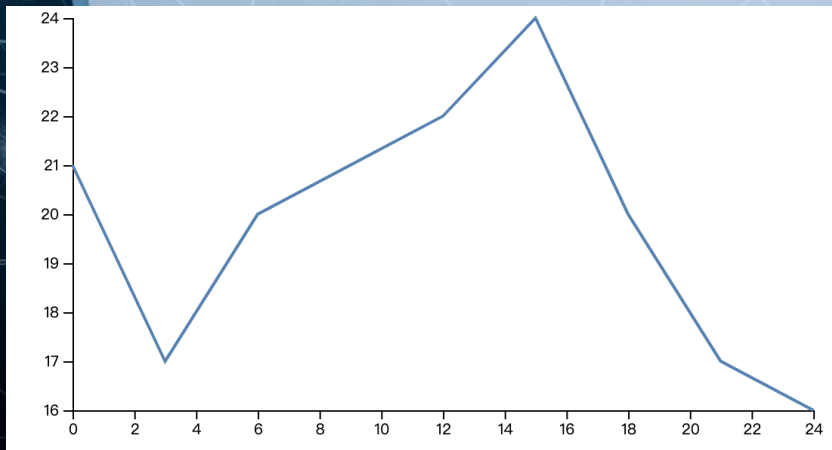


```
var lineGenerator = d3.line()  
    .x(d=>xScale(d.hour))  
    .y(d=>yScale(d.temperature));  
  
g.append("path")  
    .attr("fill", "none")  
    .attr("stroke", "steelblue")  
    .attr("stroke-width", 2)  
    .attr("d", lineGenerator(data));
```

Note: A red arrow points from the word "later" in the text above to the `lineGenerator(data)` call in the code.

Ex05-02

- main.js
- Example: when the lineGenerator receives the data, it iterates through the data and calculate all “xScale(d.hour)” as the x locations of the line chart

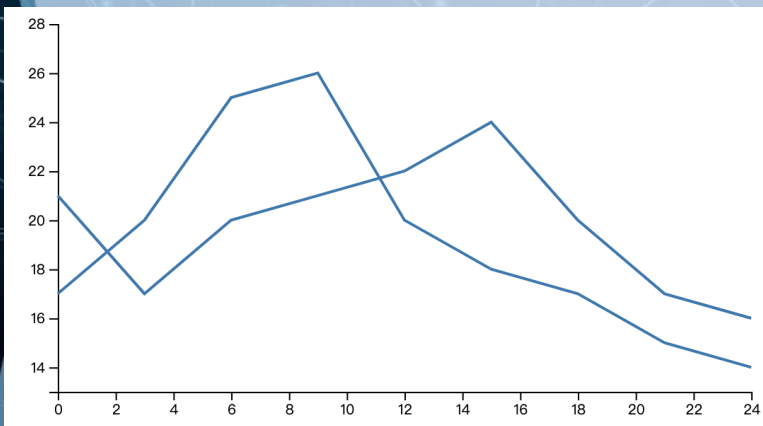


```
var data = [  
  {'hour': 0, 'temperature': 21},  
  {'hour': 3, 'temperature': 17},  
  {'hour': 6, 'temperature': 20},  
  {'hour': 9, 'temperature': 21},  
  {'hour': 12, 'temperature': 22},  
  {'hour': 15, 'temperature': 24},  
  {'hour': 18, 'temperature': 20},  
  {'hour': 21, 'temperature': 17},  
  {'hour': 24, 'temperature': 16},  
];
```

```
var lineGenerator = d3.line()  
  .x(d=>xScale(d.hour))  
  .y(d=>yScale(d.temperature));  
  
g.append("path")  
  .attr("fill", "none")  
  .attr("stroke", "steelblue")  
  .attr("stroke-width", 2)  
  .attr("d", lineGenerator(data));
```

Ex05-02Ext (Multiple Lines)

- main.js
- data: array with two elements and each one is an array of dictionary
- Visualize each array of dictionary by one line



```
var data = [
  [{ 'hour': 0, 'temperature': 21}, { 'hour': 3, 'temperature': 17},
  { 'hour': 6, 'temperature': 20}, { 'hour': 9, 'temperature': 21},
  { 'hour': 12, 'temperature': 22}, { 'hour': 15, 'temperature': 24},
  { 'hour': 18, 'temperature': 20}, { 'hour': 21, 'temperature': 17},
  { 'hour': 24, 'temperature': 16}],

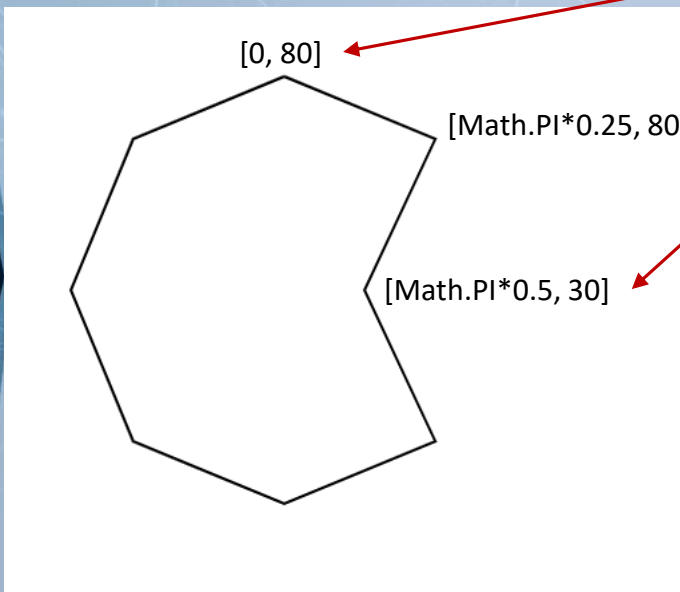
  [{ 'hour': 0, 'temperature': 17}, { 'hour': 3, 'temperature': 20},
  { 'hour': 6, 'temperature': 25}, { 'hour': 9, 'temperature': 26},
  { 'hour': 12, 'temperature': 20}, { 'hour': 15, 'temperature': 18},
  { 'hour': 18, 'temperature': 17}, { 'hour': 21, 'temperature': 15},
  { 'hour': 24, 'temperature': 14}]
];
```

```
var lineGenerator = d3.line()
  .x(d=>xScale(d.hour))
  .y(d=>yScale(d.temperature));

g.selectAll(".line")
  .data(data)
  .enter().append("path")
  .attr("fill", "none")
  .attr("stroke", "steelblue")
  .attr("stroke-width", 2)
  .attr("d", d=>lineGenerator(d))
```

Radial Line – d3.radialLine()

- Ex05-03
- The radial line generator is similar to the line generator but the points are formed by **angle in radians(clockwise)** and **radius**
 - Application: radar graphs
- <https://github.com/d3/d3-shape/blob/v2.1.0/README.md#lineRadial>



```
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]
];

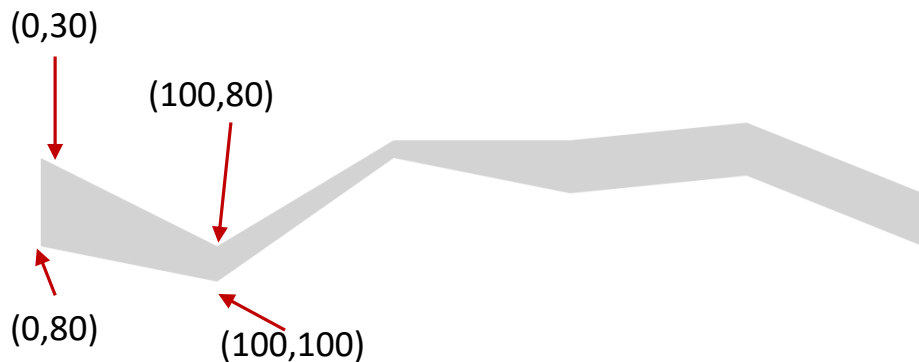
var radialLineGenerator = d3.radialLine();
var pathData = radialLineGenerator(points);
const svg = d3.select("#chart-area")
  .append("svg")
  .attr('width', 600).attr('height', 600)
  .append('g')
  .attr("transform", "translate(100,100)")
  .append('path')
  .attr('d', pathData)
  .attr('fill', 'none')
  .attr('stroke', 'black');
```

radius

Angle in radians

Area – d3.area()

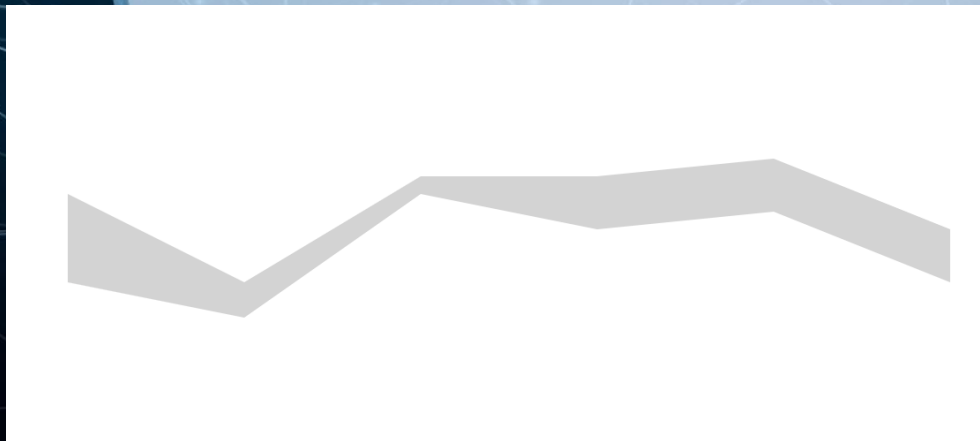
- Ex05-04
- The area generator outputs path that defines an area between two lines
 - Data can be encoded into coordinates on the two lines
 - Application: stream graph, filled line chart
 - <https://github.com/d3/d3-shape/blob/v2.1.0/README.md#area>



```
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(d=>d.x)  
  .y0(d=>d.y0)  
  .y1(d=>d.y1);  
var pathData = areaGenerator(points);  
const svg = d3.select("#chart-area")  
  .append("svg")  
  .attr('width', 600).attr('height', 600)  
  .append('g')  
  .attr("transform", "translate(100,100)")  
  .append('path')  
  .attr('d', pathData)  
  .attr('fill', 'lightgrey');
```

Area – d3.area()

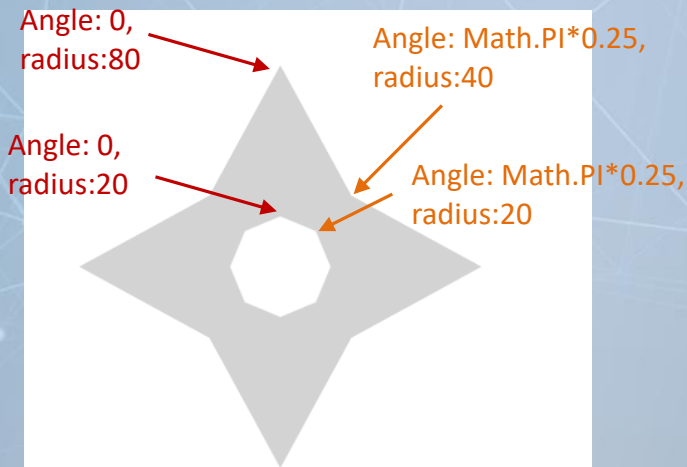
- Ex05-04
- .x(), .y0(), .y1() to indicate the format of the data



```
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(d=>d.x)  
  .y0(d=>d.y0)  
  .y1(d=>d.y1);  
var pathData = areaGenerator(points);  
const svg = d3.select("#chart-area")  
  .append("svg")  
  .attr('width', 600).attr('height', 600)  
  .append('g')  
  .attr("transform", "translate(100,100)")  
  .append('path')  
  .attr('d', pathData)  
  .attr('fill', 'lightgrey');
```


Radial Area – d3.radialArea()

- Ex05-05
- The radial area generator is similar to the area generator but the points are formed by **angle** in **radians(clockwise)** and **radius**
 - Application: filled radar graphs
 - <https://github.com/d3/d3-shape/blob/v2.1.0/README.md#areaRadial>

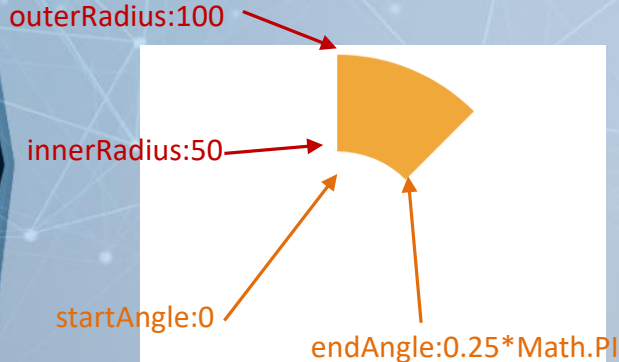


```
var points = [  
  {angle: 0, r0: 20, r1: 80},  
  {angle: Math.PI * 0.25, r0: 20, r1: 40},  
  {angle: Math.PI * 0.5, r0: 20, r1: 80},  
  {angle: Math.PI * 0.75, r0: 20, r1: 40},  
  {angle: Math.PI, r0: 20, r1: 80},  
  {angle: Math.PI * 1.25, r0: 20, r1: 40},  
  {angle: Math.PI * 1.5, r0: 20,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

```
var radialAreaGenerator = d3.radialArea()  
  .angle(d=>d.angle)  
  .innerRadius(d=>d.r0)  
  .outerRadius(d=>d.r1);  
var pathData = radialAreaGenerator(points);  
const svg = d3.select("#chart-area")  
  .append("svg")  
  .attr('width', 600).attr('height', 600)  
  .append('g')  
  .attr("transform", "translate(100,100)")  
  .append('path')  
  .attr('d', pathData)  
  .attr('fill', 'lightgrey');
```

Arc – d3.arc()

- Ex05-06
- Arc generator produce path data from **angle** and **radius** values
 - Application: pie chart
- <https://github.com/d3/d3-shape/blob/v2.1.0/README.md#arc>



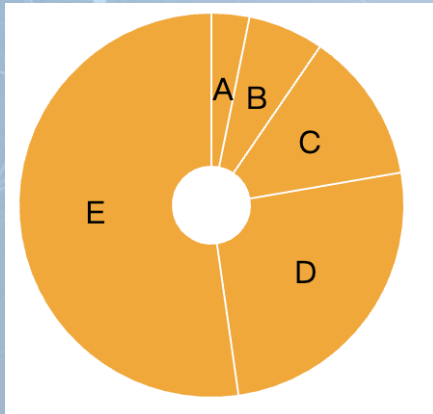
```
var data = {  
  startAngle: 0,  
  endAngle: 0.25 * Math.PI,  
  innerRadius: 50,  
  outerRadius: 100  
};
```

A dictionary for an arc

```
var arcGenerator = d3.arc();  
var pathData = arcGenerator(data);  
const svg = d3.select("#chart-area")  
  .append("svg")  
  .attr('width', 600).attr('height', 600)  
  .append('g')  
  .attr("transform", "translate(100,100)")  
  .append('path')  
  .attr('d', pathData)  
  .attr('fill', 'orange');
```

Pie Chart – Multiple Arcs

- Ex05-07



```
var data = [  
  {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}  
];
```

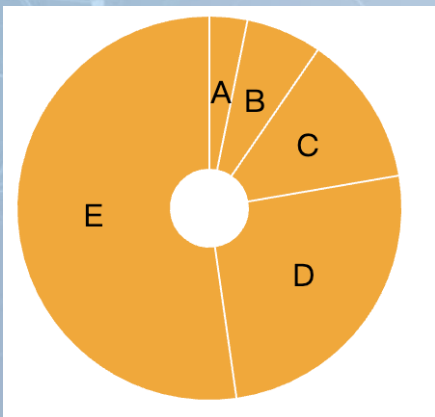
Dictionary array: an arc is a dictionary

```
var arcGenerator = d3.arc()  
  .innerRadius(20)  
  .outerRadius(100);  
  
const svg = d3.select("#chart-area")  
  .append("svg")  
  .attr('width', 600).attr('height', 600);  
const g = svg.append('g')  
  .attr("transform", "translate(100,100)");  
  
g.selectAll('path')  
  .data(data)  
  .enter()  
  .append('path')  
  .attr('d', arcGenerator)  
  .attr('fill', 'orange')  
  .attr('stroke', 'white');  
  
g.selectAll('text')  
  .data(data)  
  .enter()  
  .append('text')  
  .each(function(d) {  
    var centroid = arcGenerator.centroid(d);  
    d3.select(this)  
      .attr('x', centroid[0])  
      .attr('y', centroid[1])  
      .attr('dx', '-0.33em')  
      .attr('dy', '0.33em')  
      .text(d.label);  
  });
```

Pie Chart – Multiple Arcs

- Ex05-07

This is an alternative way to assign the inner and outer radius of arcs if they are all the same



```
var data = [
  {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);

const svg = d3.select("#chart-area")
  .append("svg")
  .attr('width', 600).attr('height', 600);
const g = svg.append('g')
  .attr("transform", "translate(100,100)");

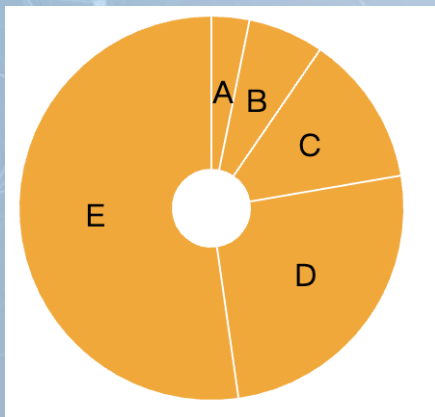
g.selectAll('path')
  .data(data)
  .enter()
  .append('path')
  .attr('d', arcGenerator)
  .attr('fill', 'orange')
  .attr('stroke', 'white');

g.selectAll('text')
  .data(data)
  .enter()
  .append('text')
  .each(function(d) {
    var centroid = arcGenerator.centroid(d);
    d3.select(this)
      .attr('x', centroid[0])
      .attr('y', centroid[1])
      .attr('dx', '-0.33em')
      .attr('dy', '0.33em')
      .text(d.label);
  });
```

Pie Chart – Multiple Arcs

- Ex05-07

Append <svg> and <g>. We will draw the pie chart in 'g'



```
var data = [
  {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);

const svg = d3.select("#chart-area")
  .append("svg")
  .attr('width', 600).attr('height', 600);
const g = svg.append('g')
  .attr("transform", "translate(100,100)");

g.selectAll('path')
  .data(data)
  .enter()
  .append('path')
  .attr('d', arcGenerator)
  .attr('fill', 'orange')
  .attr('stroke', 'white');

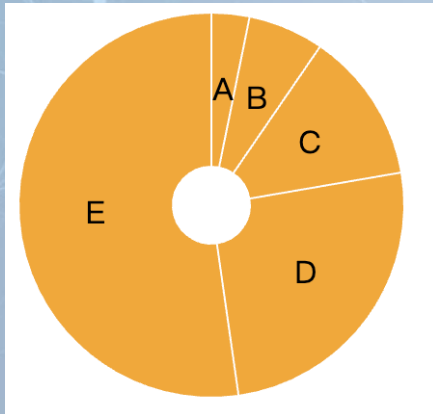
g.selectAll('text')
  .data(data)
  .enter()
  .append('text')
  .each(function(d) {
    var centroid = arcGenerator.centroid(d);
    d3.select(this)
      .attr('x', centroid[0])
      .attr('y', centroid[1])
      .attr('dx', '-0.33em')
      .attr('dy', '0.33em')
      .text(d.label);
  });
```

Pie Chart – Multiple Arcs

- Ex05-07

Draw the pie chart (without texts)

Assign out data to let d3 know we have 5 elements

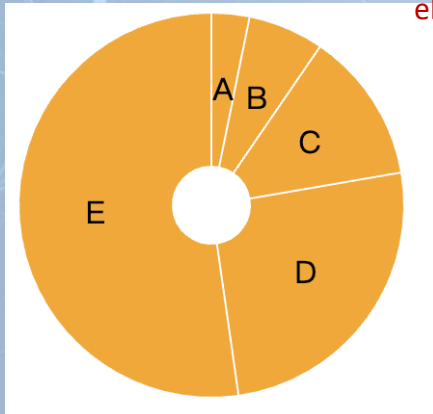


The data element is fed to 'arcGenerator' to generate path of an arc

```
var data = [  
  {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);  
  
const svg = d3.select("#chart-area")  
  .append("svg")  
  .attr('width', 600).attr('height', 600);  
const g = svg.append('g')  
  .attr("transform", "translate(100,100)");  
  
g.selectAll('path')  
  .data(data)  
  .enter()  
  .append('path')  
  .attr('d', arcGenerator)  
  .attr('fill', 'orange')  
  .attr('stroke', 'white');  
  
g.selectAll('text')  
  .data(data)  
  .enter()  
  .append('text')  
  .each(function(d) {  
    var centroid = arcGenerator.centroid(d);  
    d3.select(this)  
      .attr('x', centroid[0])  
      .attr('y', centroid[1])  
      .attr('dx', '-0.33em')  
      .attr('dy', '0.33em')  
      .text(d.label);  
  });
```


Pie Chart – Multiple Arcs

- Ex05-07



Draw texts

We will append 5 texts
(why d3 knows? We have 5
elements in the data array)

```
var data = [
  {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);

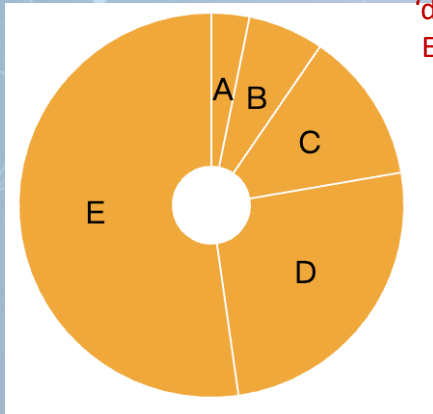
const svg = d3.select("#chart-area")
  .append("svg")
  .attr('width', 600).attr('height', 600);
const g = svg.append('g')
  .attr("transform", "translate(100,100)");

g.selectAll('path')
  .data(data)
  .enter()
  .append('path')
  .attr('d', arcGenerator)
  .attr('fill', 'orange')
  .attr('stroke', 'white');

g.selectAll('text')
  .data(data)
  .enter()
  .append('text')
  .each(function(d) {
    var centroid = arcGenerator.centroid(d);
    d3.select(this)
      .attr('x', centroid[0])
      .attr('y', centroid[1])
      .attr('dx', '-0.33em')
      .attr('dy', '0.33em')
      .text(d.label);
  });
```

Pie Chart – Multiple Arcs

- Ex05-07



Iterate through all 'd'.
What is d? an element in 'data'.
Example:

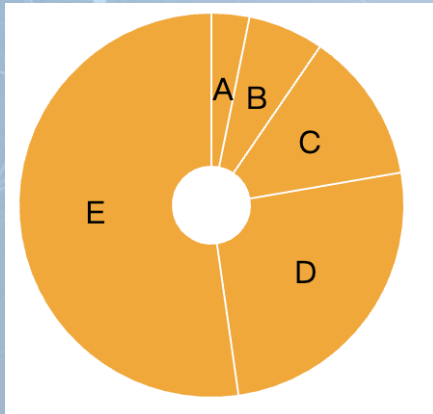
```
> {label: "A", startAngle: 0, endAngle: 0.2}
```

Draw texts

```
var data = [  
  {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);  
  
const svg = d3.select("#chart-area")  
  .append("svg")  
  .attr('width', 600).attr('height', 600);  
const g = svg.append('g')  
  .attr("transform", "translate(100,100)");  
  
g.selectAll('path')  
  .data(data)  
  .enter()  
  .append('path')  
  .attr('d', arcGenerator)  
  .attr('fill', 'orange')  
  .attr('stroke', 'white');  
  
g.selectAll('text')  
  .data(data)  
  .enter()  
  .append('text')  
  .each(function(d) {  
    var centroid = arcGenerator.centroid(d);  
    d3.select(this)  
      .attr('x', centroid[0])  
      .attr('y', centroid[1])  
      .attr('dx', '-0.33em')  
      .attr('dy', '0.33em')  
      .text(d.label);  
  });
```

Pie Chart – Multiple Arcs

- Ex05-07



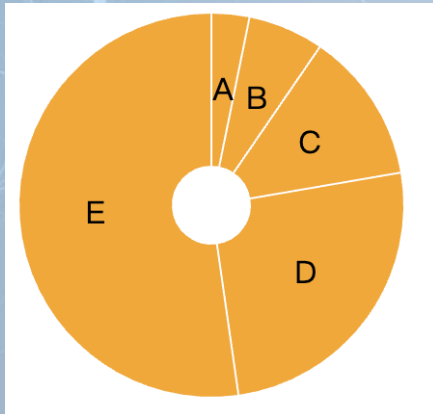
Pass the element in 'data'
to arcGenerator and get
the centroid of it

Draw texts

```
var data = [  
  {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);  
  
const svg = d3.select("#chart-area")  
  .append("svg")  
  .attr('width', 600).attr('height', 600);  
const g = svg.append('g')  
  .attr("transform", "translate(100,100)");  
  
g.selectAll('path')  
  .data(data)  
  .enter()  
  .append('path')  
  .attr('d', arcGenerator)  
  .attr('fill', 'orange')  
  .attr('stroke', 'white');  
  
g.selectAll('text')  
  .data(data)  
  .enter()  
  .append('text')  
  .each(function(d) {  
    var centroid = arcGenerator.centroid(d);  
    d3.select(this)  
      .attr('x', centroid[0])  
      .attr('y', centroid[1])  
      .attr('dx', '-0.33em')  
      .attr('dy', '0.33em')  
      .text(d.label);  
  });
```

Pie Chart – Multiple Arcs

- Ex05-07



Draw texts

'this'? Current 'text'

Set where the text should be and where it should be

```
var data = [  
  {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);  
  
const svg = d3.select("#chart-area")  
  .append("svg")  
  .attr('width', 600).attr('height', 600);  
const g = svg.append('g')  
  .attr("transform", "translate(100,100)");  
  
g.selectAll('path')  
  .data(data)  
  .enter()  
  .append('path')  
  .attr('d', arcGenerator)  
  .attr('fill', 'orange')  
  .attr('stroke', 'white');  
  
g.selectAll('text')  
  .data(data)  
  .enter()  
  .append('text')  
  .each(function(d) {  
    var centroid = arcGenerator.centroid(d);  
    d3.select(this)  
      .attr('x', centroid[0])  
      .attr('y', centroid[1])  
      .attr('dx', '-0.33em')  
      .attr('dy', '0.33em')  
      .text(d.label);  
  });
```

Symbols – d3.symbol()

- Ex05-08
- The symbol generator generates path for symbols
- <https://github.com/d3/d3-shape/tree/v2.1.0#symbols>



```
var symbolGenerator = d3.symbol()  
    .size(180)  
    .type(d3.symbolStar);  
  
// .type(d3.symbolCross);  
// .type(d3.symbolDiamond);  
// .type(d3.symbolSquare);  
// .type(d3.symbolCircle);  
// .type(d3.symbolTriangle);  
// .type(d3.symbolWye);  
  
const svg = d3.select("#chart-area")  
    .append("svg")  
    .attr('width', 600).attr('height', 600)  
    .append('path')  
    .attr("transform", "translate(100,100)")  
    .attr('d', symbolGenerator());
```



d3.symbolCircle



d3.symbolCross



d3.symbolDiamond



d3.symbolSquare



d3.symbolStar



d3.symbolTriangle



d3.symbolWye