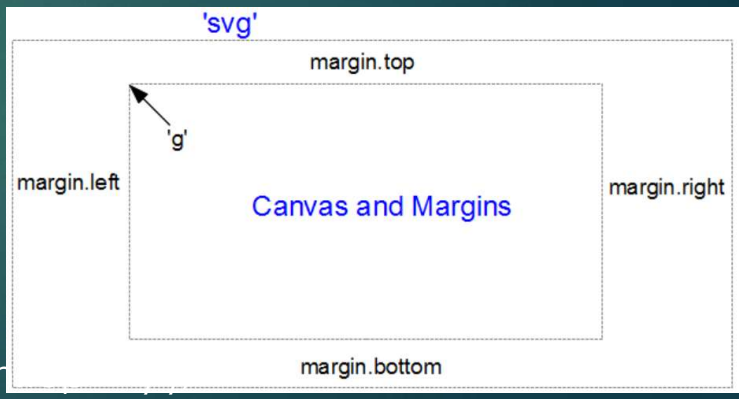
SVG g element

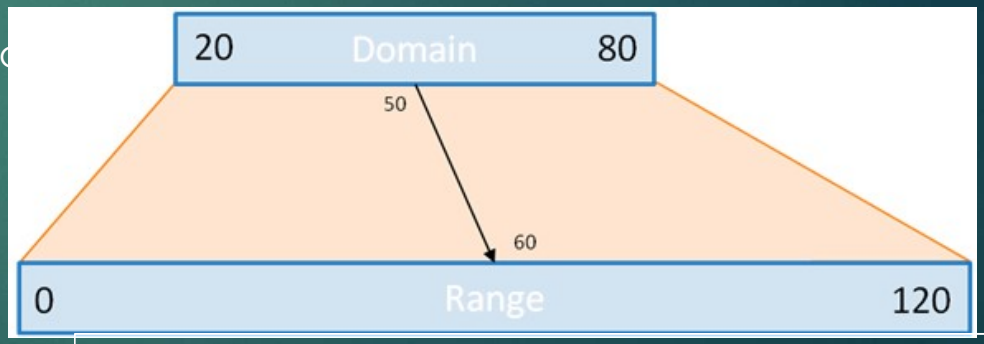
- Transformations applied to the <g> element are performed on all of its child elements, and any of its attributes are inherited by its child elements.

```
let svg = d3.select("svg"),
        margin = {
                top: 20,
                right: 20,
                bottom: 50,
                left: 50
        },
        width = +svg.attr("width") - margin.left - margin.right,
        height = +svg.attr("height") - margin.top - margin.bottom,
let chartContainer = svg.append("g")
            .attr("transform", "translate(" + margin.left + "," + margin
```



D3 Scale Functions

- Scale functions map a dimension of abstract data to a visual dimension.
  - return output in another interval: Visual property
    - position coordinate
    - a length or a radius
    - a color
  - take an input in a certain interval calle
    - a number, date or category
  - Considers 2 types of data
    - Contiuous: Quantitative
    - Discrete:
      - explicit set of values
      - Ordinal, Categorical



Source http://www.jeromecukier.net/blog/2011/08/11/d3-scales-and-color/

D3 Scale Functions

- Scales with **continuous input** and **continuous output**
  - **d3.scaleLinear():**
    - most suitable scale for transforming data values into positions and lengths
    - use a linear function (y = m * x + b) to interpolate across the domain and range
    - Domain and Range:
      - .domain([minV, maxV).range([minCoordinate,maxCoordinate]);
      - ex: d3.scaleLinear().domain([0, 100]).range([0, width])
      - ex: d3.scaleLinear().domain([0, 100]).range([height, 0])

http://d3indepth.com/scales/

D3 Scale Functions

- Scales with **continuous input** and **continuous output** (continued)
  - **d3.scaleSqrt():**
    - useful for mapping data values in to area dimensions such as radius of the circle, width of a square
    - Interpolates using equation (y = m*sqrt(x) + b) function.
  - **d3.scaleLog():**
    - can be useful when the data has an exponential nature to it, data that is not uniformly distributed.
    - Interpolate using a log function (y = m * log(x) + b).
    - domain must be strictly positive.

http://d3indepth.com/scales/

D3 Scale Functions

- Scales with **continuous input** and **continuous output (continued)**
  - **d3.scaleTime():**
    - most suitable for mapping time-series data.
    - Similar to scaleLinear. Domain is [minDate, maxDate].
      - ex: .domain([new Date(2016, 0, 1), new Date(2017, 0, 1)])

http://d3indepth.com/scales/

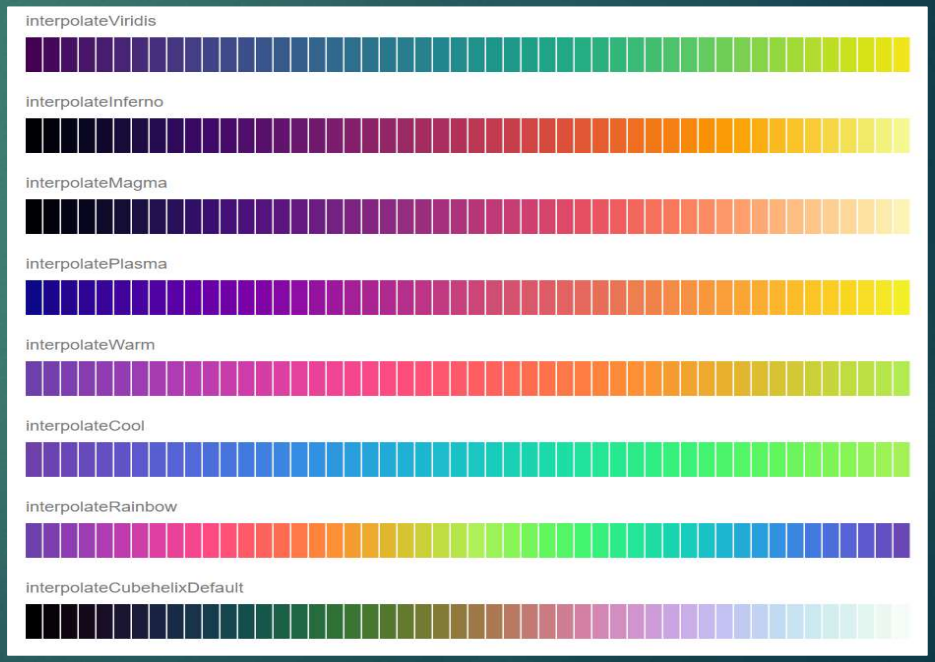
D3 Scale Functions

- Scales with **continuous input** and **continuous output** (continued)
  - **d3.scaleSequential():** similar to continuous scales in that they map a continuous, numeric input domain to a continuous output range. However, unlike continuous scales, the output range of a sequential scale is fixed by its interpolator
    - An interpolator is a function that accepts input between 0 and 1 and outputs an interpolated value between two numbers, colors, strings etc.
    - ex: d3.scaleSequential().domain([0, 100]).interpolator(d3.interpolateRainbow)

http://d3indepth.com/scales/

D3 Scale Functions

- Scales with **continuous input** and **continuous output** (continued)
  - **d3.scaleSequential():**
    - List of d3 interpolators:
      - 'interpolateViridis'
      - 'interpolateInferno'
      - 'interpolateMagma'
      - 'interpolatePlasma'
      - 'interpolateWarm'
      - 'interpolateCool'
      - 'interpolateRainbow'
      - 'interpolateCubehelixDefault'



http://d3indepth.com/scales/

D3 Scale Functions

- Scales with **continuous input** and **continuous output** (continued)
  - **d3.scaleSequential():**
    - Additional interpolators:

      Available from https://github.com/d3/d3-scale-chromatic
      - "interpolateBlues", "interpolateGreens", "interpolateGreys", "interpolateOranges", "interpolatePurples", "interpolateReds",
      - "interpolateBuGn", "interpolateBuPu", "interpolateGnBu", "interpolateOrRd", "interpolatePuBu", "interpolatePuRd", "interpolateRdPu", "interpolateYlGn",
      - "interpolatePuBuGn", "interpolateYlGnBu", "interpolateYlOrBr", "interpolateYlOrRd"

D3 Scale Functions

- Scales with **discrete input** and **discrete output**
  - **d3.scaleBand:** maps *n* bands to a specified range, where *n* is the domain size. Similar to ordinal scales, but use a continuous range instead of a discrete range.
    - Useful for creating Bar Charts with an ordinal or categorical dimension
    - The value of the last item in the domain is less than the upper bound of the interval.
    - The width of each band can be accessed using .bandwidth()
    - ex: let scale = d3.scaleBand()

      .domain(["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]

      .range([0,140]

      scale("Saturday") // 120 <= (140/7)*6
    - scale.bandwidth() // 20 <= 140/7
    - Allows specification of padding between the bands and before/after the band.

      .paddingInner(0.05) // Percent of the stepsize (also available via scae.step()

      .paddingOuter(0.05)

http://d3indepth.com/scales/

D3 Scale Functions

- Scales with **discrete input** and **discrete output**
  - **d3.scaleOrdinal:** maps discrete values (specified by an array) to discrete values (also specified by an array). The domain array specifies all possible input values and the range specifies all possible output values. The range array will repeat if it's shorter than the domain array..
  - ex: let colorScale = d3.scaleOrdinal().range(d3.schemeCategory10);

Ref:
https://github.com/d3/d3-scale-chromatic
http://d3indepth.com/scales/

D3 Scale Functions

- Scales with **discrete input** and **discrete output**
  - **d3.scalePoint:** maps from a discrete set of values to equally spaced points along the specified range.
    - useful for scatterplots an ordinal or categorical dimension
    - The distance between the points can be accessed using .step()

    ex: let scale = d3.scalePoint()

    .domain(["Sunday","Monday",'Tuesday',"Wednesday","Thursday","Friday","Saturday"]

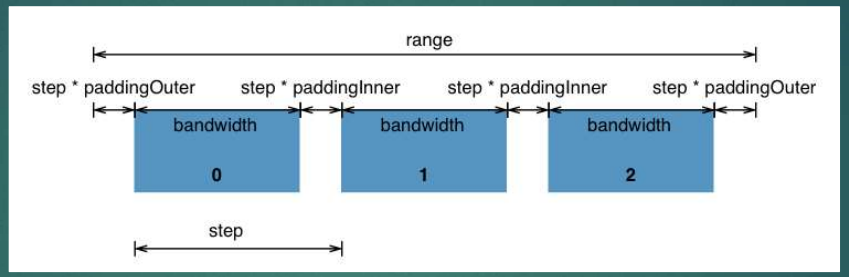
    .range([0,120]

    scale("Saturday") // 120 <= (120/6)*6
    - scale.step() // 20 <= 120/6
    - Outer padding can be specified as the ratio of the padding to point spacing. For example, for the outside padding to be a quarter of the point spacing use a value of 0.25

    scale.padding (0.25) // Percent of the point spacing

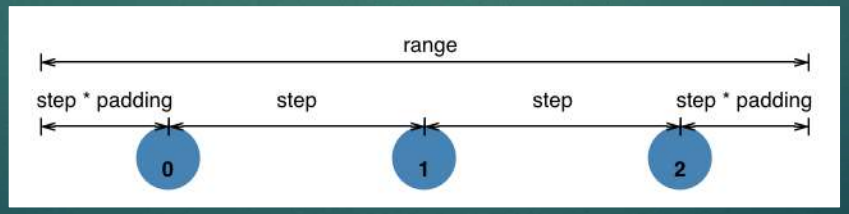
http://d3indepth.com/scales/

scaleBand() vs scalePoint()



scaleBand

scalePoint

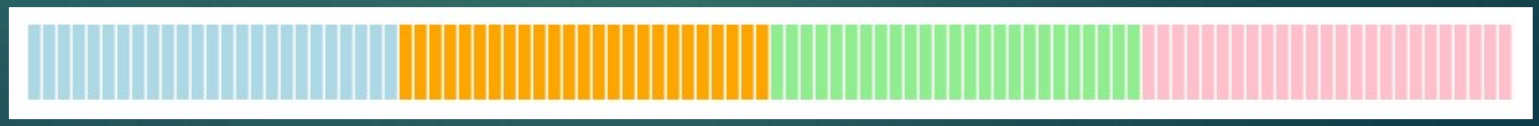


https://github.com/d3/d3-scale#ordinal-scales

D3 Scale Functions

- Scales with **continuous input** and **discrete output**
  - **d3.scaleQuantize:** Quantize scales are similar to linear scales, except they use a discrete rather than continuous range.
    - The continuous input domain is divided into uniform segments based on the number of values in (i.e., the cardinality of) the output range.
    - Each range value y can be expressed as a quantized linear function of the domain value x: y = m.round(x) + b.
  - ex: d3.scaleQuantize()

```
.domain([0, 100])
.range(['lightblue', 'orange', 'lightgreen', 'pink']);
```

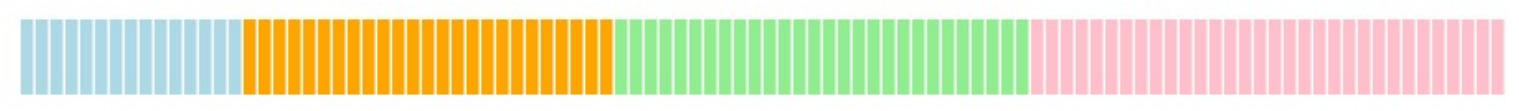


http://d3indepth.com/scales/

D3 Scale Functions

- Scales with **continuous input** and **discrete output** (continued)
  - **d3.scaleQuantile:**
    - Quantile scales map a sampled input domain to a discrete range.
    - The domain is considered continuous and thus the scale will accept any reasonable input value; however, the domain is specified as a discrete set of sample values. The number of values in (the cardinality of) the output range determines the number of quantiles that will be computed from the domain.
    - To compute the quantiles, the domain is sorted, and treated as a population of discrete values
    - ex: var myData = [0, 5, 7, 10, 20, 30, 35, 40, 60, 62, 65, 70, 80, 90, 100];

      d3.scaleQuantile().domain(myData).range(['lightblue', 'orange', 'lightgreen', 'pink']);



http://d3indepth.com/scales/

D3 Scale Functions

- Scales with **continuous input** and **discrete output** (continued)
  - **d3.scaleThreshold:**
    - maps continuous numeric input to discrete values defined by the range.
    - ex: d3.scaleThreshold()

```
.domain([10,50,90])
.range(['lightblue', 'orange', 'lightgreen', 'pink']);
```

u < 10 is mapped to '#ccc'

10 ≤ u < 50 to 'lightblue'

50 ≤ u < 9100 to 'orange'

u ≥ 90 to '#ccc'

http://d3indepth.com/scales/

D3 Shapes: “circle”

```
let data = [ 0, 2, 3, 5, 7.5, 9, 10 ];

let myScale = d3.scaleLinear()
  .domain([0, 10])
  .range([0, 600]);
```

```
svg.selectAll('circle')
        .data(dataArray)
        .enter()
        .append('circle')
        .attr('r', 3)
        .attr('cx', function(d) {
                return myScale(d);
        });
```

D3 Shapes: “path”

- selection.append(“path”)

  .attr(“d”, …)
- D3 helpers
  - Line generator: d3.line()
    - .curve(…) : curveLinear, curveCardinal, closed equivalents

D3 path generator

- d3.line()
- For example:

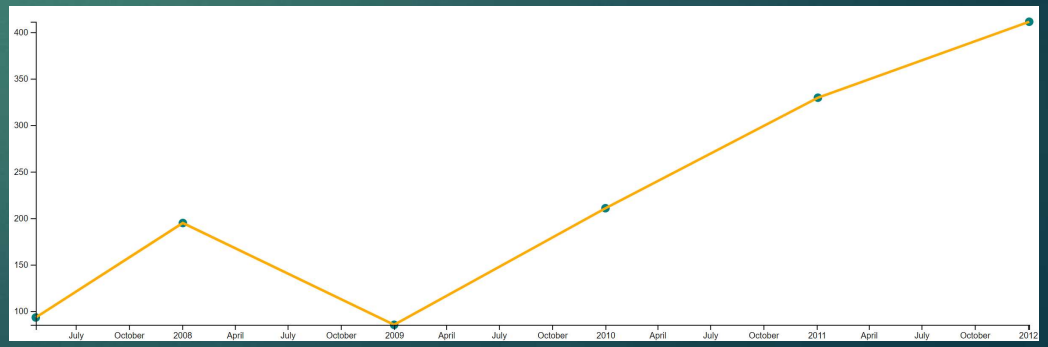
```
let lineGenerator = d3.line()
        .x(function(d, i) { return d.day; })
         // set the x values for the line generator
        .y(function(d) { return d.earning; });
         // set the y values for the line generator
```

d3.line()

- let pathGenerator = d3.line()
  .x(function(d) {
  return xScale(d.date);
  })
  .y(function(d) {
  return yScale(d.close);
  });

- chartContainer.append("path")
  .datum(D)
  .attr("d", function(d){
  return pathGenerator(d);
  })

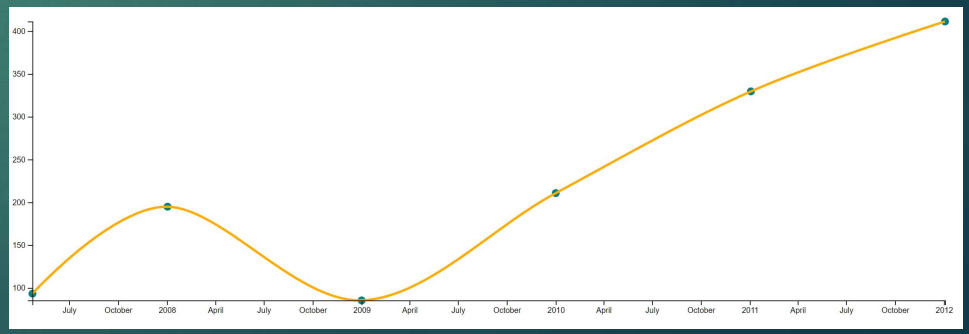
```
let D = [
        {date: "2007-04-23",close: 93.24},
        {date: "2008-01-02",close:194.84},
        {date: "2009-01-01",close:85.35},
        {date: "2010-01-01",close:210.73},
        {date: "2011-01-03",close:329.57},
        {date: "2012-01-03",close:411.23}
]
```



d3.line()

- let pathGenerator = d3.line()
  .x(function(d) {
  return xScale(d.date);
  })
  .y(function(d) {
  return yScale(d.close);
  })
  .curve(d3.curveCardinal);

- chartContainer.append("path")
  .datum(D)
  .attr("d", function(d){
  return pathGenerator(d);
  })



Curve Choices:

curveLinear, curveMonotoneX, curveCardinal, curveCatmullRom, curveMonotone, curveNatural, curveStep, …

D3 Symbols

- built-in symbol types: circle, cross, diamond, square, star, triangle, and wye all use "path"
- ex: let symbolGenerator = d3.symbol().size(100).type(d3.cross)
  - selection.append("path")

```
.attr("d",function(){
        return symbolGenerator();
})
```

https://github.com/d3/d3-shape#symbols