**D3 - Data-Driven Documents**

* D3.js is a JavaScript library for manipulating documents based on data.
* D3 helps you bring data to life using HTML, SVG, and CSS.
* D3’s emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation
* Download the latest version from <https://d3js.org/>  
    
  OR
* Can you the CDN URL : <script src="<https://d3js.org/d3.v4.min.js>"></script>

**D3 Concepts**

* **Selections :** how d3 grabs on to html elements. (used to act on elements existing on the page or create new elements on to the page)
* **Real time updates :** way of binding data to the dom
* **Dynamic functions :**
* **Transitions :**
* **Web standards :** follows all the web standards
* **Interactive :**

**Selecting DOM elements**

* For selecting DOM elements, D3 has two main functions
* select()
* selectAll()
* d3.selectAll('p')  
   .style('background-color', 'red')  
   .style('color', 'white')  
   .text('Hello Everyone !!!');

**Adding properties dynamically (dynamic functions)**

* Styles, attributes, and other properties can be specified as functions of data in D3, not just simple constants
* The above example can be modified to apply properties dynamically
* d3.selectAll("p")  
   .style("background-color", function(d, i) {  
   if(i%2==0) {  
   return 'green';  
   } else {  
   return 'red';  
   }  
   })  
   .style("color", "white").text(function(d, i) {  
   return 'Hello '+i;  
   });
* Computed properties often refer to bound data. Data is specified as an array of values, and each value is passed as the first argument (d) to selection functions.
* d3.selectAll("p")  
   **.data([14,16,18,20,22,24])**  
   .style("background-color", function(d, i) {  
   if(i%2==0) {  
   return 'green';  
   } else {  
   return 'red';  
   }  
   })  
   .style("color", "white").text(function(d, i) {  
   return 'Hello '+i;  
   })  
   .style('font-size', function(d, i) {  
   return d+'px';  
   });

**Enter & Exit functions**

* Using D3’s enter and exit selections, you can create new nodes for incoming data and remove outgoing nodes that are no longer needed.
* When you have lesser elements in the dom compared to the dataset, enter() can be used to dynamically create new elements in the DOM.
* And when an element is no longer needed in the dom you can remove from the dataset using the exit functions
* The below example show how to use the enter() function to create new elements and exit() to remove elements
* var selection = d3.select('body').selectAll('p')  
   .data([12,14,16,18,20,22,24]);  
  selection.enter().append('p')  
   .style('background-color', function(d, i) {  
   if(i%2==0) {  
   return 'red';  
   } else {  
   return 'green';  
   }  
   })  
   .style('color', 'white')  
   .text(function(d, i) {  
   return 'Text = '+d;  
   });
* The below example show how to reuse the element already present in the dom and to create the rest of the elements
* var selection = d3.select('body').selectAll('p')  
   .data([12,14,16,18,20,22,24]);  
    
  selection.style('background-color', function(d, i) {  
   if(i%2==0) {  
   return 'red';  
   } else {  
   return 'green';  
   }  
   })  
   .style('color', 'white')  
   .text(function(d, i) {  
   return 'Text = '+d;  
   });  
     
  selection.enter().append('p')  
   .style('background-color', function(d, i) {  
   if(i%2==0) {  
   return 'red';  
   } else {  
   return 'green';  
   }  
   })  
   .style('color', 'white')  
   .text(function(d, i) {  
   return 'Text = '+d;  
   });

**Animations**

* Using transition() function you can apply animations to the D3 output

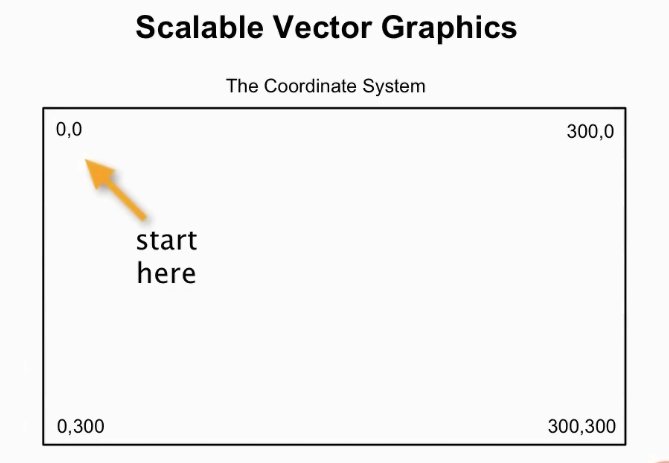
**Events**

* Using on() events can be attached.
* var selection = d3.select('body').selectAll('p')  
   .data([12,14,16,18,20,22,24], function(d, i) {  
   return d;  
   });  
  selection.enter().append('p')  
   .style('background-color', function(d, i) {  
   if(i%2==0) {  
   return 'red';  
   } else {  
   return 'green';  
   }  
   })  
   .style('color', 'white')  
   .text(function(d, i) {  
   return 'Text = '+d;  
   })  
   **.on('click', function(d, i) {  
   var event\_obj = d3.event;  
   });**

**SVG**

* Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics
* **Vector graphics :** Unlike JPEGs, GIFs, and BMP images, vector graphics are not made up of a grid of pixels.   
  + Vector graphics are comprised of paths, which are defined by a start and end point, along with other points, curves, and angles along the way
  + A path can be a line, a square, a triangle, or a curvy shape.
  + These paths can be used to create simple drawings or complex diagrams.
* JPEG, GIF, PNG are designed for specific resolution, which is not scalable. This problem can be solved using SVG
* SVG images are basically three different things, they are the shapes that define them, a circle, a line, a bar etc. we have filters which add special effects and we have gradients as well to paint those.
* But here in D3 we are gonna focus more on style SVG objects ( which are HTML objects ) using CSS properties.

**Draw shapes using SVG**

* As said SVG is an html element, we starts of by adding the <svg></svg> element
* <svg>  
   <rect width="50" height="200" style="fill:blue;"></rect>  
  </svg>
* Here the <svg> element just acts like a container and the inside the container we specify different shapes using the respective tags
* If you want to do this in D3, it is very similar  
  d3.select('body')  
   .append('svg')  
   .attr('width', 150)  
   .attr('height', 200)  
   .attr('style', 'border:1px solid red')  
   .append('rect')  
   .attr('width', 50)  
   .attr('height', 200)  
   .style('fill', 'red');
* Basically D3 allows you to do the same thing using javascript.
* SVG has a coordinate system  
    
  
* Difference in this system is that 0.0 being the bottom left, it starts from top left
* Let's draw a circle  
  + **SVG :**
  + <svg width="50" height="50">  
     <circle cx="25" cy="25" r="25" style="fill:blue;" />  
     </svg>
  + **D3 :**
  + d3.select('body')  
     .append('svg').attr('width', 50).attr('height', 50)  
     .append('circle').attr('cx', 25).attr('cy', 25).attr('r', 25)  
     .style('fill', 'red');

**Drawing with data**

var width = 200,  
 height = 100,  
 padding = 2,  
 dataset = [5,10,15,20,25];  
   
var svg = d3.select('body')  
 .append('svg')  
 .attr('width', width).attr('height', height);  
  
svg.selectAll('rect').data(dataset)  
 .enter()  
 .append('rect')  
 .attr('x', 0)  
 .attr('y', 0)  
 .attr('width', 20)  
 .attr('height', 100);

* The above example will create a 5 rectangles one above the other, based on the dataset array which is passed to D3.
* Let's modify the same so that we can make all the rectangles visible
* Let's change the **x position**
  + .attr('x', function (d, i) {  
     return i \* (width/dataset.length);  
     })
* Changing the height
  + .attr('height', function(d, i) {  
     return d;  
     })
* Changing the y position so that it show properly
  + .attr('y', function(d, i) {  
     return height - d;  
     })
* Just to make it bigger let's increment the value of x and height
  + .attr('y', function(d, i) {  
     return height - (d\*4);  
     })
  + .attr('height', function(d, i) {  
     return d \* 4;  
     })
* Making the width parameterized
  + .attr('width', width / dataset.length - padding)

**Charts**

Let's look on basic charting using D3

**Bar chart**

* Building a bar chart is exactly what we did on the last exercise.
* var width = 300,  
   height = 100,  
   padding = 2,  
   dataset = [5, 10, 15, 20 ,25];  
     
  var svg = d3.select('body').append('svg')  
   .attr('width', width)  
   .attr('height', height);  
     
  svg.selectAll('rect')  
   .data(dataset)  
   .enter()  
   .append('rect')  
   .attr('x', function(d, i) {  
   return i \* (width / dataset.length);  
   })  
   .attr('y', function(d, i) {  
   return height - d \* 4;  
   })  
   .attr('width', width/dataset.length - padding)  
   .attr('height', function(d) {  
   return d \* 4;  
   });
* **Styling bar chart**
  + var width = 200,  
     height = 100,  
     padding = 2,  
     dataset = [5,10,15,20,25];  
       
    var svg = d3.select('body')  
     .append('svg').attr('width', width).attr('height', height);  
      
    svg.selectAll('rect')  
     .data(dataset)  
     .enter()  
     .append('rect')  
     .attr('x', function(d, i) {  
     return i\*(width/dataset.length);  
     })  
     .attr('y', function(d,i) {  
     return height-(d\*4);  
     })  
     .attr('width', width / dataset.length - padding)  
     .attr('height', function(d, i) {  
     return d \* 4;  
     })  
     .attr('fill', function(d, i) {  
     return colorChart(d);  
     });  
       
    function colorChart(d) {  
     if(d>=0 && d<10) {  
     return 'red';  
     }  
     if(d>=10 && d<15) {  
     return 'green';  
     }  
     if(d>=15 && d<20) {  
     return 'yellow';  
     }  
     if(d>=20 && d<25) {  
     return 'blue';  
     }  
     if(d>=25 && d<30) {  
     return 'orange';  
     }  
    }
* Adding labels to bar chart  
  + svg.selectAll('text')  
     .data(dataset)  
     .enter()  
     .append('text')  
     .text(function(d, i) {  
     return d;  
     })  
     .attr("text-anchor", "middle")  
     .attr('x', function(d, i) {  
     return i \* (width / dataset.length) + (width / dataset.length) / 2;  
     })  
     .attr('y', function(d, i) {  
     return height-(d\*4) + 15;  
     })  
     .attr("font-family", "sans-serif")  
     .attr("font-size", "11px")  
     .attr("fill", "black");
  + SVG text-anchor attribute to center the text horizontally at the assigned x value
  + Then, let’s change the way we calculate the x position by setting it to the left edge of each bar plus half the bar width

**Line chart**

* var width = 500,  
   height = 200;  
     
  var arr = [  
   {x:10, y:20},  
   {x:20, y:12},  
   {x:30, y:70},  
   {x:40, y:30},  
   {x:50, y:10},  
   {x:60, y:40},  
   {x:70, y:50},  
   {x:80, y:70},  
   {x:90, y:60},  
   {x:100, y:40}  
  ];  
    
  **//creating a function that draw a line for us**  
  var lineFunc = d3.line()  
   .x(function(d) { return d.x\*3 })  
   .y(function(d) { return d.y\*2; })  
   .curve(d3.curveLinear);  
    
    
  var svg = d3.select('body').append('svg')  
   .attr('width', width)  
   .attr('height', height);  
    
  **// special type of svg objects “ path “ which allows to draw shapes using data, (used when you are not sure about the shape that you have to create)**svg.append('path')  
   .attr('d', lineFunc(arr))  
   .attr('stroke', 'red')  
   .attr('stroke-width', 2)  
   .attr('fill', 'none');  
     
  svg.selectAll('text')  
   .data(arr)  
   .enter()  
   .append('text')  
   .text(function(d, i) {  
   return d.x;  
   })  
   .attr('x', function(d,i) {  
   return d.x\*3;  
   })  
   .attr('y', function(d,i) {  
   return d.y\*2;  
   })  
   .attr('text-anchor', 'start')  
   .attr('font-size', '14px')  
   .attr('font-weight', function(d, i) {  
   if(i==0 || i==(arr.length-1)) {  
   return 'bold';  
   } else {  
   return 'normal';  
   }  
   });

**Building a scatter plot**

var width = 500,

height = 200;

var arr = [

{x:10, y:20},

{x:20, y:12},

{x:30, y:70},

{x:40, y:30},

{x:50, y:10},

{x:60, y:40},

{x:70, y:50},

{x:80, y:70},

{x:90, y:60},

{x:100, y:40}

];

**Creating svg element**

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

**Creating scatter plot**

var dots = svg.selectAll('circle').data(arr).enter().append('circle')

.attr('cx', function(d,i) {

return d.x\*3;

})

.attr('cy', function(d,i) {

return d.y\*2;

})

.attr('r', 5)

.attr('fill', '#000')

.attr('fill', function(d) {

return styleScatterPlot(d.x);

});

function styleScatterPlot(d) {

if(d>=10 && d<20) {

return 'red';

}

if(d>=20 && d<30) {

return 'green';

}

if(d>=30 && d<40) {

return 'yellow';

}

if(d>=40 && d<50) {

return 'blue';

}

if(d>=50 && d<60) {

return 'grey';

}

if(d>=60 && d<70) {

return 'cyan';

}

if(d>=70 && d<80) {

return 'violet';

}

if(d>=80 && d<90) {

return 'indigo';

}

if(d>=90 && d<100) {

return 'orange';

}

}

**Adding labels to scatter plot**

var labels = svg.selectAll('text').data(arr).enter().append('text')

.text(function(d) {

return d.x+', '+d.y;

})

.attr('x', function(d,i) {

return d.x\*3;

})

.attr('y', function(d,i) {

return d.y\*2;

})

.attr('text-anchor', 'start')

.attr('font-size', '12px')

.attr('font-weight', function(d, i) {

if(i==0 || i==(arr.length-1)) {

return 'bold';

} else {

return 'normal';

}

});

**Scaling data**

var width = 500,

height = 200;

var arr = [

{x:10, y:20},

{x:20, y:12},

{x:30, y:70},

{x:40, y:30},

{x:50, y:10},

{x:60, y:40},

{x:70, y:50},

{x:80, y:70},

{x:90, y:60},

{x:100, y:40}

];

**Creating scales**

var xScale = d3.scaleLinear()

.domain([

d3.min(arr, function(d) {

return d.x;

}),

d3.max(arr, function(d) {

return d.x;

})

])

.range([0, width]);

var yScale = d3.scaleLinear()

.domain([

d3.min(arr, function(d) {

return d.y;

}),

d3.max(arr, function(d) {

return d.y;

})

])

.range([height, 0]);

**Using scales to defines the points in the chart**

var lineFunc = d3.line()

.x(function(d) { return xScale(d.x); })

.y(function(d) { return yScale(d.y); })

.curve(d3.curveLinear);

**Creating svg element**

var svg = d3.select('body').append('svg')

.attr('width', width)

.attr('height', height);

**Drawing line chart using path**

svg.append('path')

.attr('d', lineFunc(arr))

.attr('stroke', 'red')

.attr('stroke-width', 2)

.attr('fill', 'none');

**Adding axis to chart**

var width = 500,

height = 200,

padding = 20;

var arr = [

{x:10, y:20},

{x:20, y:12},

{x:30, y:70},

{x:40, y:30},

{x:50, y:10},

{x:60, y:40},

{x:70, y:50},

{x:80, y:70},

{x:90, y:60},

{x:100, y:40}

];

**Creating a line chart using a scale**

var xScale = d3.scaleLinear()

.domain([

d3.min(arr, function(d) {

return d.x;

}),

d3.max(arr, function(d) {

return d.x;

})

])

.range([padding, width-padding]);

var yScale = d3.scaleLinear()

.domain([

d3.min(arr, function(d) {

return d.y;

}),

d3.max(arr, function(d) {

return d.y;

})

])

.range([height-padding, 10]);

var lineFunc = d3.line()

.x(function(d) { return xScale(d.x); })

.y(function(d) { return yScale(d.y); })

.curve(d3.curveLinear);

var svg = d3.select('body').append('svg')

.attr('width', width)

.attr('height', height);

svg.append('path')

.attr('d', lineFunc(arr))

.attr('stroke', 'red')

.attr('stroke-width', 2)

.attr('fill', 'none');

**Adding axis to chart**

var yAxis = d3.axisLeft(yScale);

var axis = svg.append('g').call(yAxis)

.attr('transform', 'translate('+padding+', 0)');

var xAxis = d3.axisBottom(xScale);

var axis = svg.append('g').call(xAxis)

.attr('transform', 'translate(0, '+(height-padding)+')');