D3 – Data-Driven Documents

JavaScript library for data visualization. Employs HTML, SVG, and CSS.
http://d3js.org/

Basic HTML frame:

```
<!DOCTYPE HTML>
<html>
 <head>
 <meta http-equiv="content-type" content="text/html; charset=windows-1250">
 <title>D3 tutorial</title>
 </head>
 <body>

 </body>
</html>
```

Adding D3 library:

```
<head>
 <meta http-equiv="content-type" content="text/html; charset=windows-1250">
 <title>D3 tutorial</title>
 <script src="http://d3js.org/d3.v5.min.js"></script>
 </head>
```

Now we can make use of D3 functions:

```
<body>
        <p>This is a paragraph</p>
        <script>
                d3.select("p").text("Hello World");
        </script>
 </body>
```

Alternative way for creating a paragraph using d3:

```
<body>
        <script>
                d3.select("body").append("p").text("Hello");
        </script>
 </body>
```

Brows the content: using Inspect Element
Section Console:
Add to the code: console.log(d3);

Better indentation:

```
<script>
        d3.select("body")
                .append("p")
                .text("Hello");
 </script>
```

Setting colour to paragraph

If we wat to change colour of the element, we can use *style* method. It has two parameters – what we want to change and the value we want to set.

```
.style("color","red")
```

Another way to change properties of elements is to use *attr* method. Attributes are for example *src*, *href*,…

SVG – scalable vector graphics

For creating 2D graphic elements in web pages.
We will create a circle, a rectangle, and a line. But first we need to create canvas (container) where elements will be drawn:

```
d3.select("body")
      .append("svg")
      .attr("width",500)
      .attr("height",500);
```

Within the svg environment method *attr* is used instead of *style*.

We will save the canvas into a variable:

```
<script>
        var canvas = d3.select("body")
              .append("svg")
              .attr("width",500)
              .attr("height",500);
 </script>
```

Create the graphic primitives:

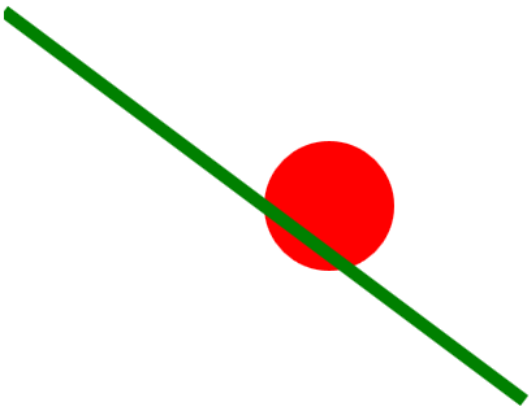
```
var circle = canvas.append("circle")
              .attr("cx", 250)
              .attr("cy", 250)
              .attr("r", 50)
              .attr("fill", "red");

    var rect = canvas.append("rect")
              .attr("width", 100)
              .attr("height", 50);
```

If we do not set the coordinates of the primitives (cx and cy for circle center, x and y for top left corner of the rectangle), the elements will be placed to top left corner of the canvas.

Add a line:

```
var line = canvas.append("line")
                .attr("x1", 0)
                .attr("y1", 100)
                .attr("x2", 400)
                .attr("y2", 400)
                .attr("stroke", "green")
                .attr("stroke-width", 10);
```



How to bind the data with visualization

We will use SVG for visualization of a basic dataset. First, we will create a dataset. One of the simplest ways is creating your own array with values (place this before the canvas initialization):

```
var dataArray = [20, 40, 50];
```

Then we will try to create a barchart showing this data array. We will store the barchart in a variable *bars*:

```
var bars = canvas.selectAll("rect")
                .data(dataArray)
                .enter()
```

Because the bars will be drawn as rectangle, we will use method *selectAll* with parameter *rect*. This method selects all elements of type *rect* within canvas – at this point an empty set.
.*data* is a function, which binds the selected set with our *dataArray*
.*enter* is a function, which then creates a placeholder for each data element in the *dataArray*.

Now we want to create a rectangle for each data element (placeholder):

```
var bars = canvas.selectAll("rect")
                .data(dataArray)
                .enter()
                     .append("rect")
```

We will set attributes for each of the rectangles, where the width of each rectangle will be derived from the corresponding data element in dataArray. Wi will achieve this using function *function,* that takes the corresponding data element (and optionally its index in *dataArray*) as an argument.

```
var bars = canvas.selectAll("rect")
                .data(dataArray)
                .enter()
                  .append("rect")
                  .attr("width", function(d) { return d; })
                  .attr("height", 50);
```

If we now try to load our page in the browser, we will see only one rectangle. This is because all the rectangles are placed over each other (you can verify this in the web browser by inspecting the elements). This can be fixed by deriving the *y* coordinate of the rectangles from their index in data array (parameter i of the *function*):

```
.attr("y", function(d, i) { return i * 100 });
```

Result:

The result is not ideal so we can adjust it a little.
Edit the line .attr("width", function(d) { return d; }) so that the value is multiplied by 10:

.attr("width", function(d) { return d * 10; })

Scales

Important part of D3. Let's try the following example - add another, bigger value to the *dataArray*:

var dataArray = [20, 40, 50, 60];

Now the width of the corresponding rectangle should be 600, so it will not fit into the canvas, which we set to 500 x 500. In such case the width of the bar will be cropped to the width of the canvas.

This is a good case for using Scales. We will try to set the size of the bars in accordance to the size of the canvas. We will set the size of the bar corresponding to biggest data value to the size of the canvas and rescale the other bars proportionally to this. For this, we will store the size of the canvas into variables and define a linear scale – a mapping function which we will use for mapping the input values to the sizes of the bars.

```
var width = 500;
var height = 500;

var widthScale = d3.scaleLinear()
        .domain([0, 60])
        .range([0, width]);

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

** use d3.scale.linear() for d3 version 3 and bellow or d3.scaleLinear() for v4 and above*

Method *.domain* defines the range of input values (in our case from 0 to the biggest value in the input *dataArray* - 60). Method *.range* defines the range of output values. These will be from 0 to the width of our canvas, since we do not want the bars to be bigger than the canvas.

Now we will edit the attribute *width* of the bars with the scale:

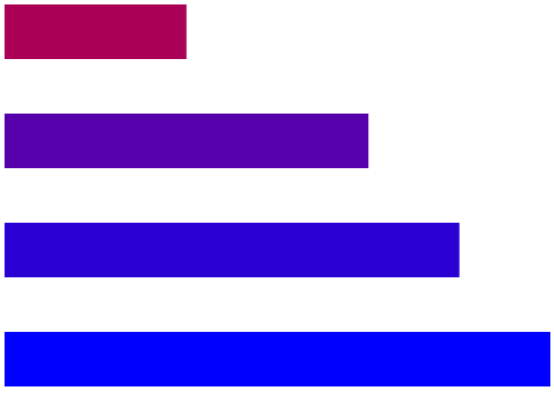
```
.attr("width", function(d) { return widthScale(d); })
```

Scale can work not only with numerical ranges but also with colours. We will try to use it to define the colour of the bars based on the data values. We will create a new scale, where range will not be given in numbers, but as a range between two colours:

```
var color = d3.scaleLinear()
            .domain([0, 60])
            .range(["red","blue"]);
```

Then we will add attribute *fill* for the bars:

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



Click Event

Now we will want to select a bar with a mouse click and highlight it. We will register a *click* operation on our bars using the *on* function:

```
bars.on("click", function(){

})
```

Within the body of the function we will specify what should happen when the bar is clicked. We will set its colour to red (we use *this* keyword to reference the clicked element).

```
bars.on("click", function(){
        d3.select(this).attr("fill", "red");
})
```

Now whenever user clicks on a bar, it will change its colour to red. However, we also want to "deselect" the previously clicked bar – we want to change it back to original colour. To access the previously clicked bar, we will store in in a variable. Each time a new bar will be clicked, we will reset the colour of the previously selected bar, then store the new one and change its colour:

```
var lastClickedBar = null;
bars.on("click", function(){
    if(lastClickedBar != null){
            d3.select(lastClickedBar).attr("fill", function(d) {return color(d);});
    }
    lastClickedBar = this;
    d3.select(this).attr("fill", "red");
})
```

Labels

We also want to add labels to our bars showing the exact values the bars represent. To do this within svg environment we will create *text* elements the same way as we created rectangles for bars. We use *.text* method to set the text within the text element.

```
canvas.selectAll("text")
        .data(dataArray)
        .enter()
          .append("text")
          .text(function(d){return d;})
          .attr("fill", "white")
          .attr("y", function(d,i) {return i*100 +30})
          .attr("x", 20);
```