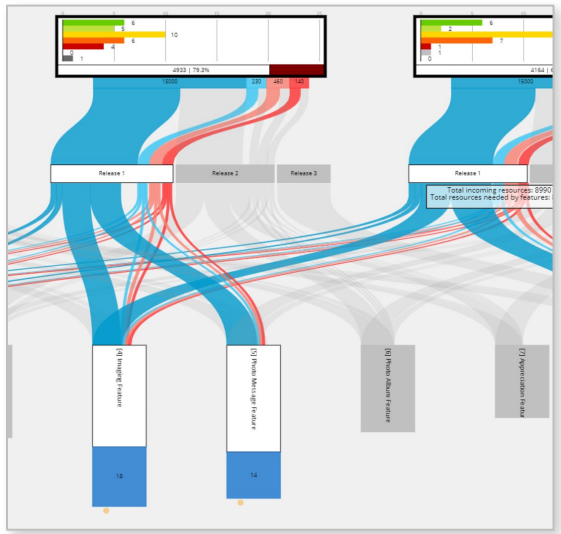
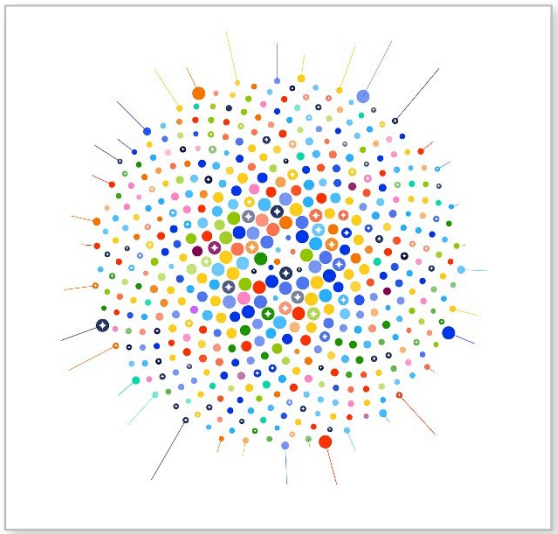
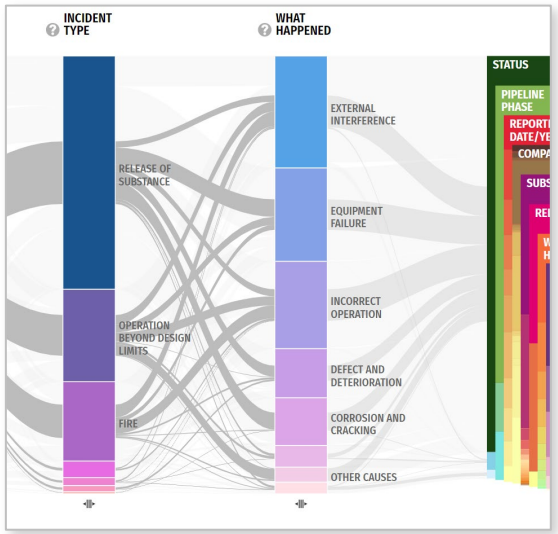
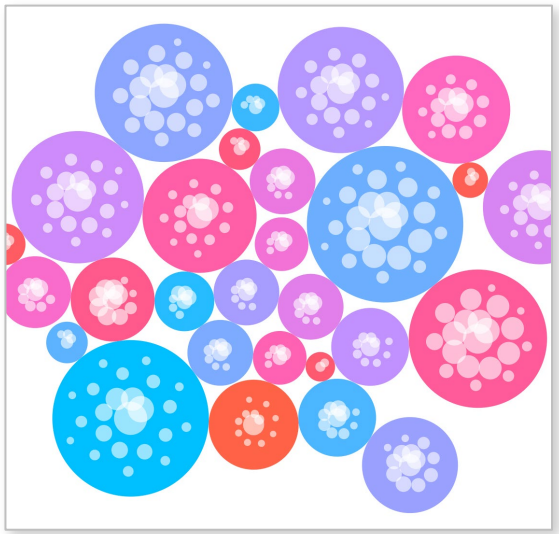
D3 Workshop

Bon Adriel Aseniero, PhD
bon.aseniero@autodesk.com

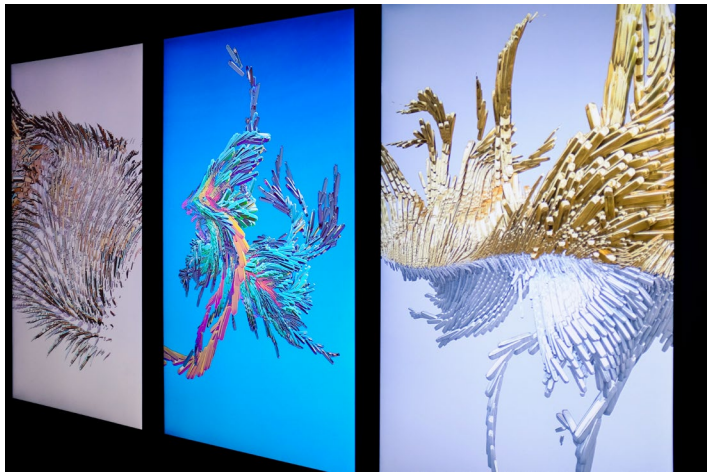




Bon Adriel Aseniero, PhD

Senior Research Scientist | Autodesk



Images taken during the 123 Data Exhibition in Paris, France, May 2018. http://123data.paris/

Webstorm IDE

https://www.jetbrains.com/webstorm/

Materials

https://www.dropbox.com/sh/nb5wdpwtx3qk58f/AACxzU4wVprytXEjmibxz5yLa?dl=0

Workshop Agenda (3 hours)

1. Introduction to D3
   - Loading and Binding Data
   - Scales
2. Drawing Data
   - Making your own representations
3. Interaction
   - Simple interaction
   - Brushing and Linking

INTRODUCTION

HTML5/JavaScript

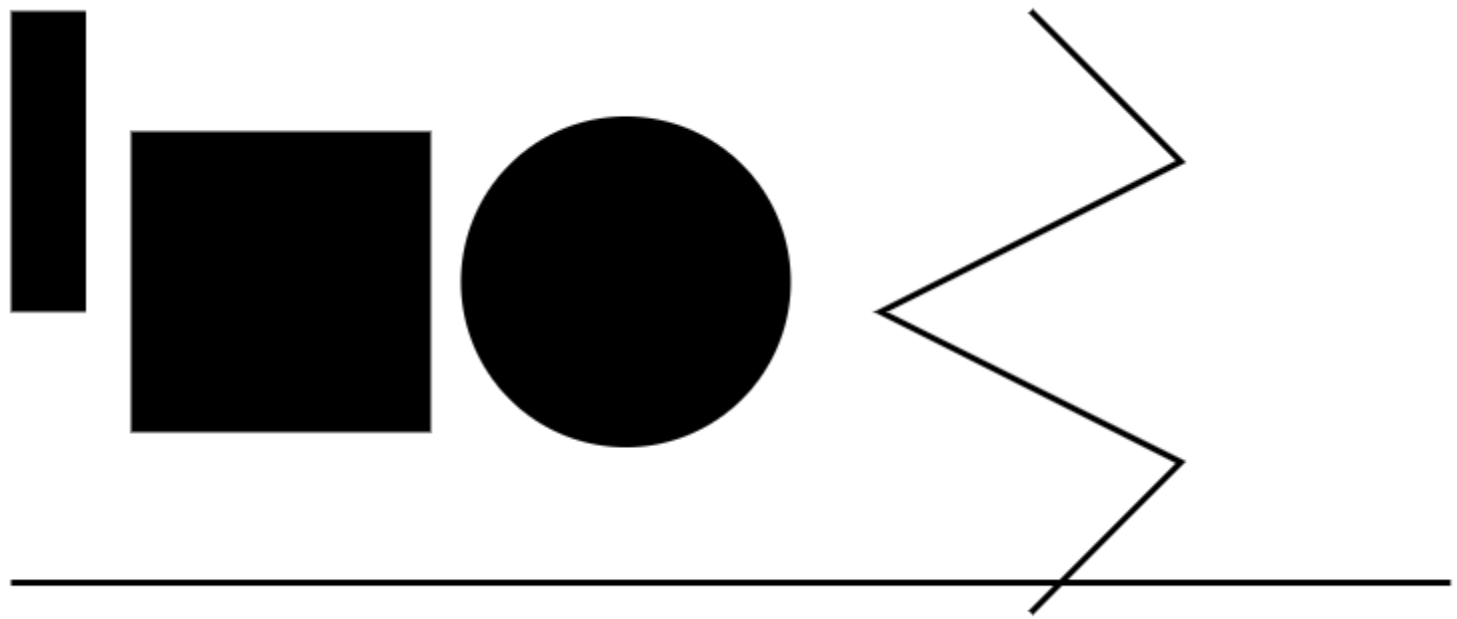
SVG: Scalable Vector Graphics

- Vector-based graphics in XML format
- Defined in the HTML DOM as SVG elements

  `<svg>...</svg>`

  1. Rectangles **<rect>**
     Attributes: x, y, width, height
  2. Circles **<circle>**
     Attributes : cx, cy, r
  3. Lines **<line>**
     Attributes : x1, y1, x2, y2
  4. Polylines **<polyline>**
     Attributes : points - string list of x, y point pairs
     e.g., “10,0 2,100, 45,38”

Hands-on:
draw SVG objects



```
<svg class="small-vis" id="vis1">
    <rect x="10" y="10" width="25" height="100"></rect>
    <rect x="50" y="50" width="100" height="100"></rect>
    <circle cx="215" cy="100" r="55"></circle>
    <line x1="10" y1="200" x2="490" y2="200"
          style="stroke:black; stroke-width:2"></line>
    <polyline points="350,10 400,60 300,110 400,160 350,210"
          style="fill:none; stroke:black; stroke-width:2"></polyline>
</svg>
```

SVG Reference:

https://www.w3schools.com/graphics/svg_intro.asp

D3

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.”

–d3js.org/

D3: Data-Driven Documents

D3 is a “declarative” programming language

- Describes **what to do**, not **how to do** something (imperative)

D3: Data-Driven Documents

D3 is a “declarative” programming language

- Describes **what to do**, not **how to do** something (imperative)
- Example: Changing the text colour of a paragraph

```
// Imperative (basic JavaScript)
var paragraphs = document.getElementsByTagName("p");
for (var i = 0; i < paragraphs.length; i++) {
    var paragraph = paragraphs.item(i);
    paragraph.style.setProperty("color", "blue", null);
}
```

D3: Data-Driven Documents

D3 is a “declarative” programming language

- Describes **what to do**, not **how to do** something (imperative)
- Example: Changing the text colour of a paragraph

```
// Declarative (D3)
D3.selectAll("p").style("color", "blue");
```

Loading Data from File

- Use D3's **.csv** function

```
/**
 * Load data from a CSV file as JSON objects
 * @param path the location of the CSV file to load
 */
function loadData(path){
    d3.csv(path).then(function(data){
        //do something with the data
        console.log(data);
    });
}
```

D3 Selection

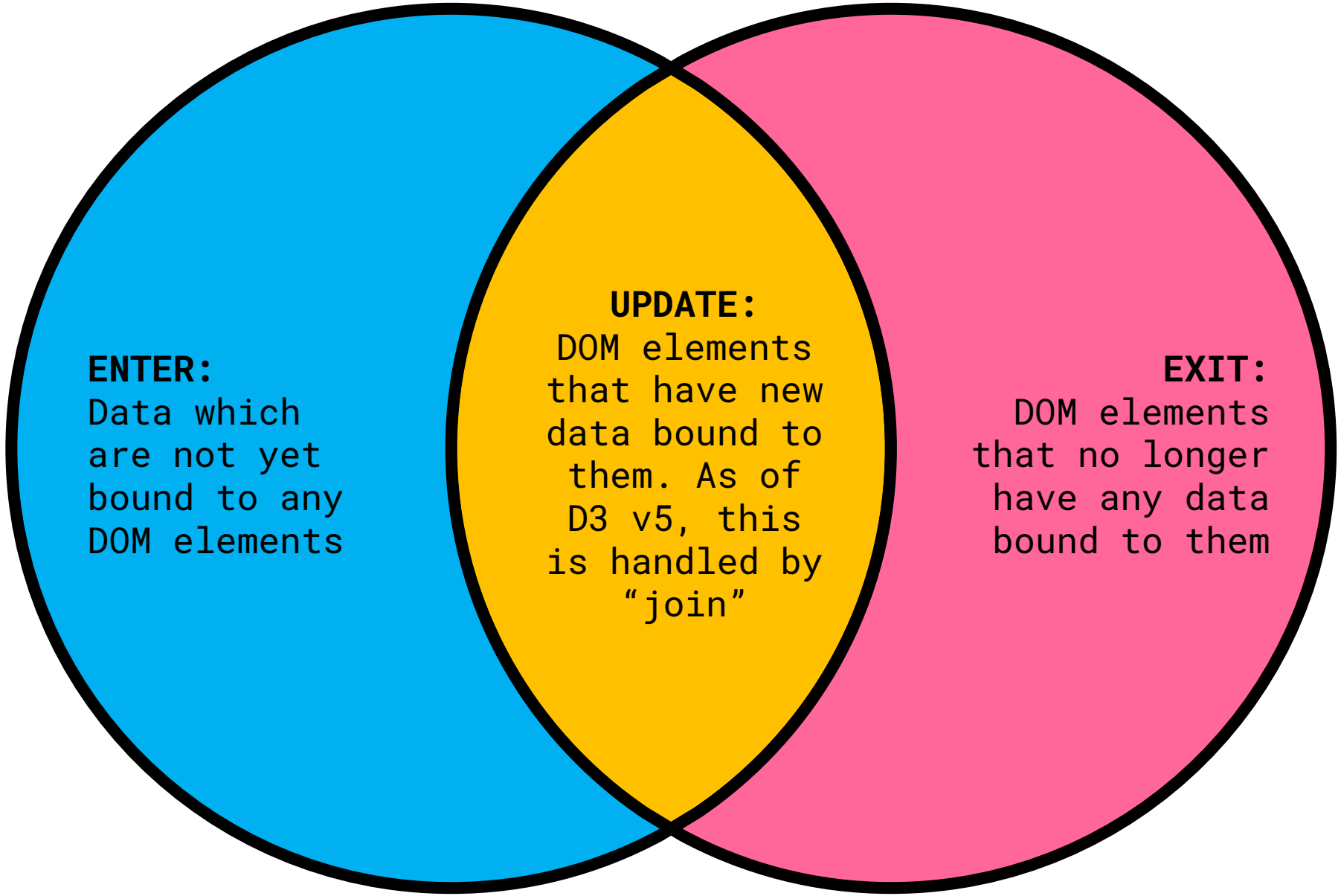
- Allows DOM elements to be selected so they can be manipulated declaratively.
- **.selectAll** – selects all matching elements

  e.g., d3.selectAll("rect") will find all rectangles in the DOM
- **.select** – selects the first matching element

  e.g., d3.select("#vis1") will select the first element in the DOM with an attribute id = "vis1“
- Resource: https://www.d3indepth.com/selections/

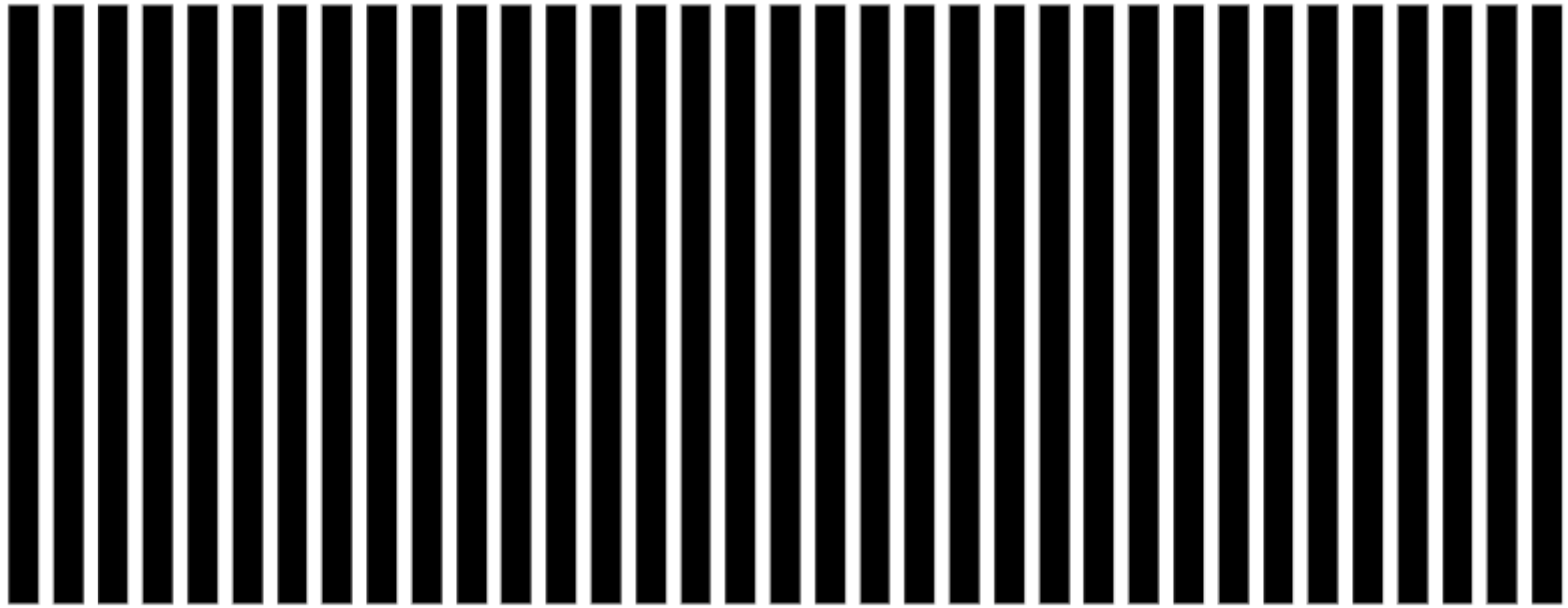
D3 Selection

- D3 selections are “virtual” as the elements do not have to exist, yet, in the DOM in order for them to be selected.

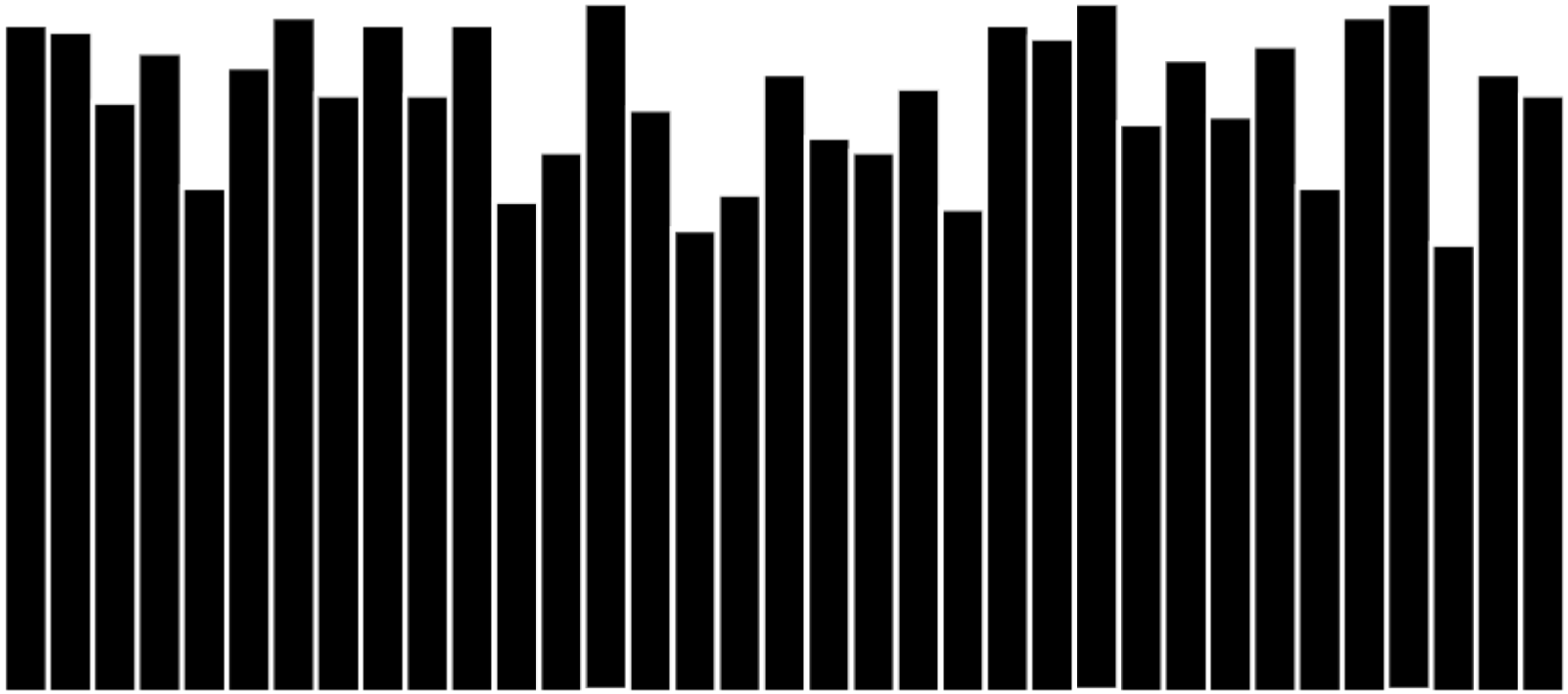

ENTER:
Data which are not yet bound to any DOM elements
UPDATE:
DOM elements that have new data bound to them. As of D3 v5, this is handled by “join”
EXIT:
DOM elements that no longer have any data bound to them

```
function drawCountries(data){
    let svg = d3.select("#vis2");
    // for dynamic sizing
    svg.attr("viewBox", `0, 0, ${$("#vis2").width()}, ${$("#vis2").height()}`);

    svg.selectAll("rect")
        .data(data)
        .enter()
        .append("rect")
        .attr("x", function(d,i){
            return (i+1) * 15;
        })
        .attr("y", 10)
        .attr("width", 10)
        .attr("height", 200);
}
```

INTRODUCTION



INTRODUCTION



```
function drawBars(data){
    let svg = d3.select("#vis3");
    // for dynamic sizing
    svg.attr("viewBox", `0, 0, ${$("#vis3").width()}, ${$("#vis3").height()}`);

    // setup margins and dimensions
    let width = $("#vis3").width(),
        height = $("#vis3").height();
    let barWidth = width / data.length;

    ...
```

D3 Scales

Scale functions that map from an input domain to an output range.

Hence, they are typically used to transform (map) data values into visual variables (e.g., position and colour).

D3 Scale Types

1. Quantitative Scales
   - Have continuous domain (e.g., set of real numbers)
   - D3 examples: Linear Scale, Power Scale
2. Ordinal Scales
   - Have a discrete domain (e.g., categories, set of names)
   - D3 example: Categorical Colours
3. Time Scales
   - A type of quantitative scale specific to measures of time (e.g., dates)

```
// setup D3 scales
// Here we create linear scales to map our x values
// to the width of the svg viewport
let xScale = d3.scaleLinear()
    .domain([0, data.length])
    .range([0, width]); // limit our range within the margins

// Here we create linear scales to map our y values
// to the height of the svg viewport
let yScale = d3.scaleLinear()
    .domain([0, d3.max(data, function(d) { return d['Water quality']; })])
    .range([height, 0]);
```

```
svg.selectAll("rect")
    .data(data)
    .enter()
    .append("rect")
    .attr("x", function(d,i){
        return xScale(i);
    })
    .attr("y", function(d){
        return yScale(d['Water quality']);
    })
    .attr("width", barWidth)
    .attr("height", height)

    // add in-line styles
    .style("stroke", "white")
    .style("stroke-width", 2);
```

INTRODUCTION

INTRODUCTION

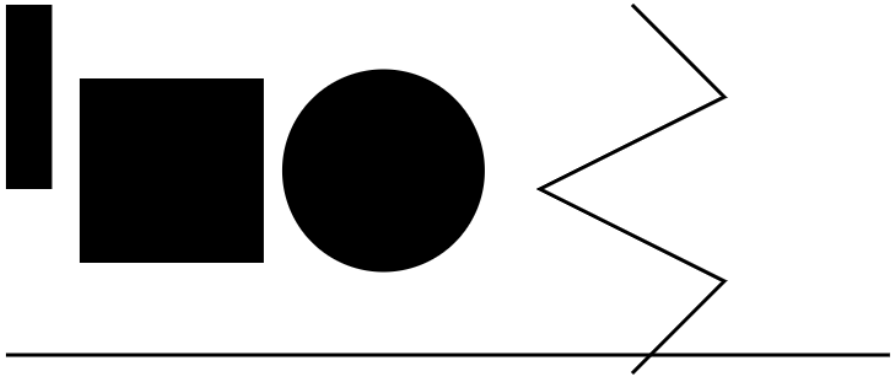
```
function drawCircles(data){
    let svg = d3.select("#vis4");
    // for dynamic sizing
    svg.attr("viewBox", `0, 0, ${$("#vis4").width()}, ${$("#vis4").height()}`);

    // setup margins and dimensions
    let width = $("#vis4").width(),
        height = $("#vis4").height();

    let maxDiameter = 40;

    // setup D3 scales
    // Here we create linear scales to map our x values
    // to the width of the svg viewport
    let xScale = d3.scaleLinear()
        .domain([30, d3.max(data, function(d) {
            return d['Feeling safe walking alone at night']; })])
        // limit our range within the margins
        .range([maxDiameter, width-maxDiameter]);

    // Here we create linear scales to map our y values
    // to the height of the svg viewport
    let yScale = d3.scaleLinear()
        .domain([d3.min(data, function(d) {
            return d['Life expectancy']; }),
            d3.max(data, function(d) { return d['Life expectancy']; })])
        .range([height-maxDiameter, maxDiameter]);
```

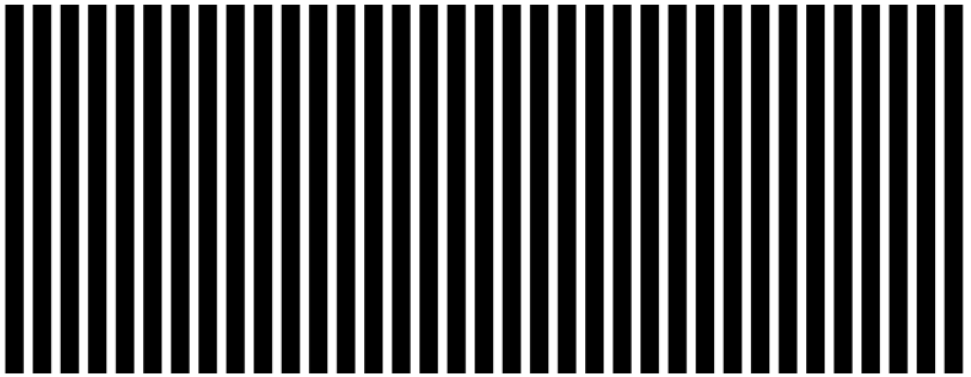
```
let rScale = d3.scaleLinear()
    .domain([0, d3.max(data, function(d) {
        return d['Homicide rate']; })])
    .range([5, 15]);

let colorScale = d3.scaleLinear()
    .interpolate(d3.interpolateHcl)
    .domain([d3.min(data, function(d) { return d['Life satisfaction']; }),
        d3.max(data, function(d) { return d['Life satisfaction']; })])
    .range(['rgb(0,0,0)', 'rgb(0,191,255)'])
    .clamp(true);
```

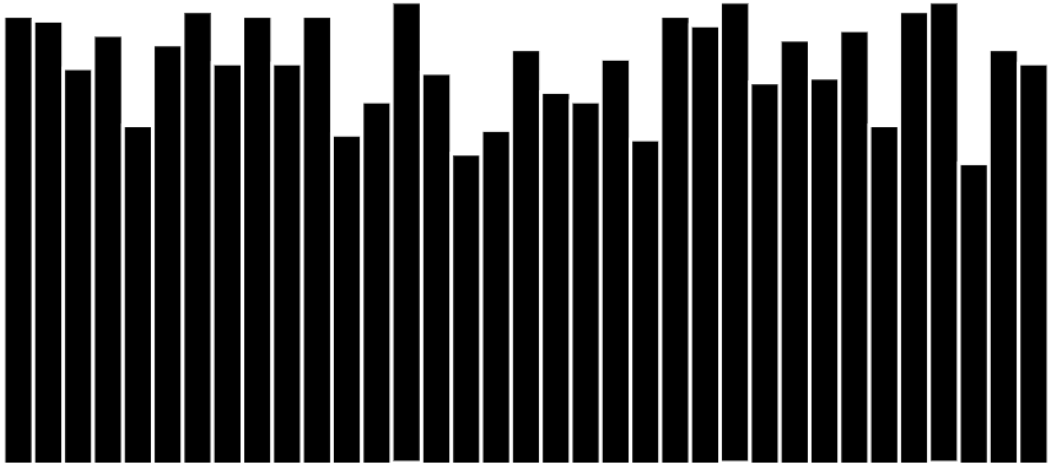
```
svg.selectAll("circle")
    .data(data)
    .enter()
    .append("circle")
    .attr("cx", function(d){
        return xScale(d['Feeling safe walking alone at night']);
    })
    .attr("cy", function(d){
        return yScale(d['Life expectancy']);
    })
    .attr("r", function(d){
        return rScale(d['Homicide rate']);
    })
    .style("fill", function(d){
        return colorScale(d['Life satisfaction']);
    })
    .style("stroke", "white")
    .style("stroke-width", 2);
```

INTRODUCTION

Part 1: Introduction





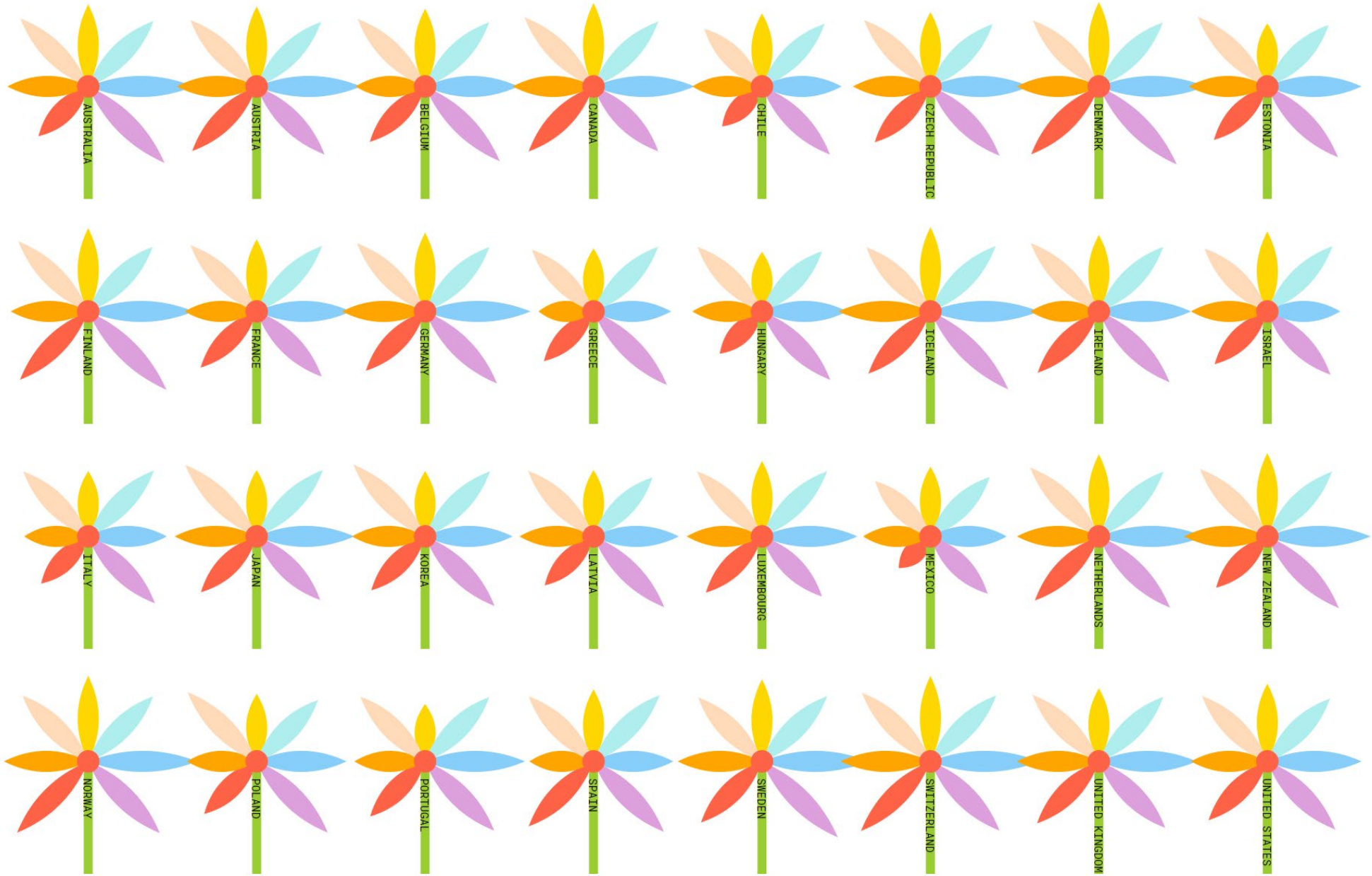


DRAWING DATA

How's life?

There is more to life than the cold numbers of GDP and economic statistics – This Index allows you to compare well-being across countries, based on 11 topics the

DRAWING DATA



Combining SVG paths to make something new

Think about the component shapes of your metaphor

Components of a flower

- Petals – **Curved Paths**
- Flower head – **a Circle**
- Stem – **a Line**

- These will all be contained in a group element “g”

SVG Groups “g”

- As we have seen before, you can use this element (denoted as **<g></g>** tags in the HTML document) to group other SVG elements together.

- You can apply transformations into the group which will apply to all the elements in it.

- Note: You **cannot** style groups.

Process

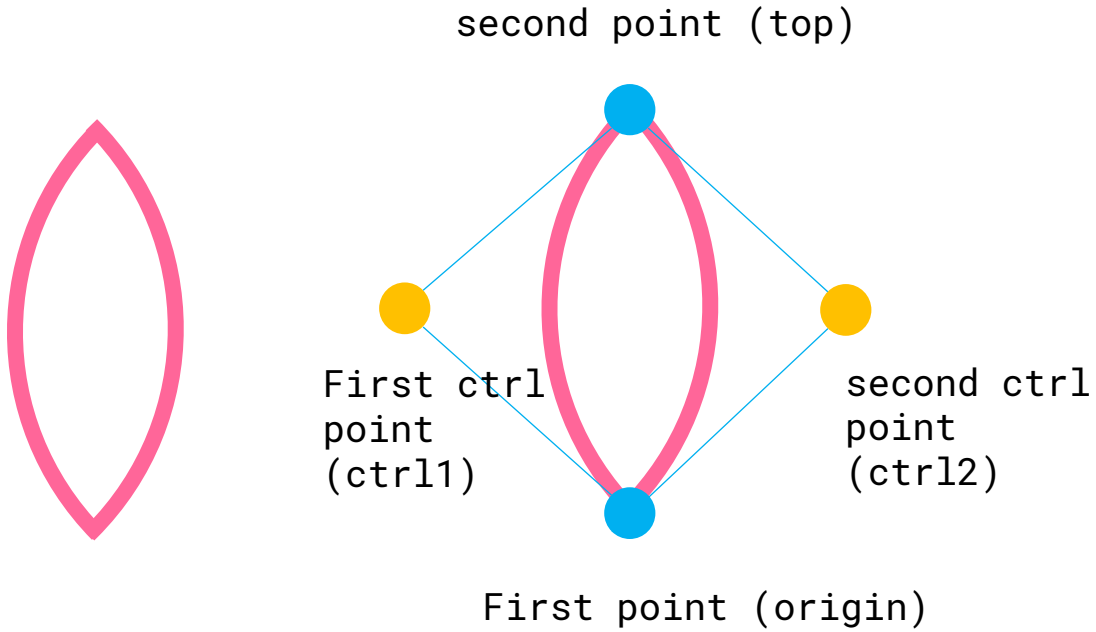
1. First, create a **group** that will contain all our svg elements

```
this.flowers = this.svgContainer.selectAll("g")
    .data(this.data)
    .enter()
    .append("g")
    .attr("class", "flower_group")
    .attr("width", this.FLOWER_WIDTH)
    .attr("height", this.FLOWER_HEIGHT)
...
```

```
.attr("transform", function(d, i){
    let new_i = i + 1;
    let x_padding = new_i % max_num_of_items === 0 ?
                    max_num_of_items : new_i % max_num_of_items,
        y_padding = i >= max_num_of_items ?
                    1 + Math.floor(i/max_num_of_items) : 1;

    let x_translate = d3.select(this).attr("width") * (x_padding - 1),
        y_translate = d3.select(this).attr("height") * (y_padding - 1);


    return `translate(${x_translate}, ${y_translate})`;
});
```

Process

2. Draw a petal (using **svg:path** definition of a quadratic Bezier curve) and add it to our **flower_group**.

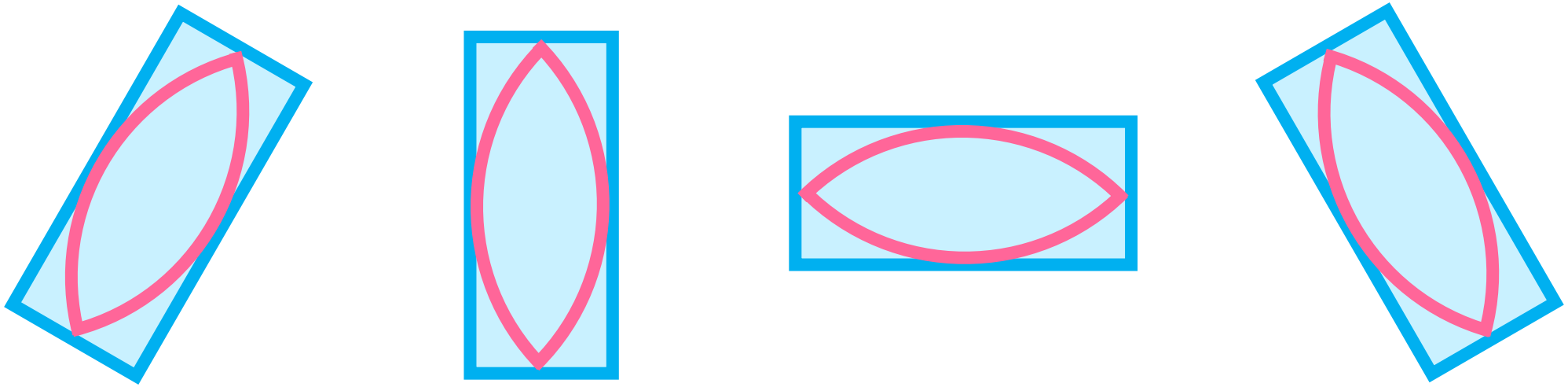



```
MX Y Q CX CY, DX DY …

M origin.x origin.y
Q ctrl1.x ctrl1.y,
top.x top.y
Q ctrl2.x ctrl2.y,
origin.x origin.y
Z
```

DRAWING DATA

```
...

// Add a path into the group using a Quadratic Bezier curve definition
// recall the quadratic curve definition: Mx y, Q cx cy, dx dy
// where x and y are the coords of the point of origin, cx and cy is the control point,
// and dx dy is the next point (this can be followed up by another control point and next point ...
let petal = petal_group.append("path")
    .attr("class", "flower_petal")
    .attr("selector", selector)
    .attr("d", function(d){
        let origin = {x : _this.FLOWER_WIDTH/2, y : _this.FLOWER_HEIGHT/2};
        let scale = d3.scaleLinear()
            .domain([0, max_val])
            .range([0, _this.FLOWER_HEIGHT/2]);
        let top = {x : origin.x, y : origin.y - scale(d[selector])};
        let ctrl1 = {x : origin.x - (origin.x/4), y: top.y + ((_this.FLOWER_HEIGHT/2-top.y)/2)};
        let ctrl2 = {x : origin.x + (origin.x/4), y: top.y + ((_this.FLOWER_HEIGHT/2-top.y)/2)};
        let path = `M${origin.x} ${origin.y} Q ${ctrl1.x} ${ctrl1.y}, ${top.x} ${top.y}
                Q ${ctrl2.x} ${ctrl2.y}, ${origin.x} ${origin.y} Z`;
        return path;
    })
    .style("opacity", 0)
    .style("fill", fill);

...
```

3. Encapsulate the path into another group which we can rotate, add width and height to, etc.



```
// create a group that will contain the SVG:path for our
petal
let petal_group = this.flowers.append("g")
    .attr("transform", `rotate(${rotation_angle},
    ${_this.FLOWER_WIDTH/2}, ${_this.FLOWER_HEIGHT/2})`);
```

4. Repeat this step to create the other petals. Use a reusable function.

```
// Add the petals to our flower
this.addPetal(0, "Life satisfaction", 10, "gold", "orange");
this.addPetal(45, "Life expectancy", 100, "paleturquoise", "mediumturquoise");
this.addPetal(90, "Water quality", 100, "lightskyblue", "deepskyblue");
this.addPetal(135, "Years in education", 20, "plum", "orchid");
this.addPetal(225, "Feeling safe walking alone at night", 100, "tomato", "firebrick");
this.addPetal(270, "Employment rate", 100, "orange", "darkorange");
this.addPetal(315, "Student skills", 600, "peachpuff", "lightsalmon");
...
```

```
/**
 * Function that adds a petal to a flower group. The petal's length is based off a data,
 * accessed through a given property selector
 * @param rotation_angle (number, degree) the angle of rotation for the petal
 * @param selector (string) the property of the data which will determine
 * @param max_val (number) the maximum value the data domain has (used for scaling values to screen pixels)
 * @param fill (colour string) the colour of the petal
 * @param stroke (colour string) the colour of the stroke outlining the petal on hover
 */
this.addPetal = function(rotation_angle, selector, max_val, fill, stroke){ ...
```

5. Add the stem and the flower head.

```
// Add our stem
this.flowers.append("line")
    .attr("x1", _this.FLOWER_WIDTH/2)
    .attr("y1", _this.FLOWER_HEIGHT/2)
    .attr("x2", _this.FLOWER_WIDTH/2)
    .attr("y2", _this.FLOWER_HEIGHT)
    .style("stroke-width", 5)
    .style("stroke", "yellowgreen")
    .style("opacity", 0)
    .transition()
    .style("opacity", 1);

// Add our flower head
let flower_head = this.flowers.append("circle")
    .style("fill", "tomato")
    .attr("cx", this.FLOWER_WIDTH/2)
    .attr("cy", this.FLOWER_HEIGHT/2)
```

INTERACTION

INTERACTION

Events

- Use the **.on** method to attach event listeners to your elements.
  - You can use predefined event names such as:
  - mouseover, mouseout, click, etc.

- Syntax:

```
.on(<event_name>, <callback_function>)
```

- Note: any listeners (**.on("eventname", callback)**) should appear separately or you could encounter errors. e.g.:

```
petal
    .transition()
    .style("opacity", 1)
    .duration(800);

petal
    .on("mouseover", function(d){
        d3.select(this)
            .style("stroke-width", 5)
            .style("stroke", stroke);

        d3.selectAll(".flower_petal")
            .filter(function(e){
                return d3.select(this).attr("selector") !== selector;
            })
            .transition()
            .style("opacity", 0.1);
    })
```

Adding Animations

- Use the **transition()** method.
- Any attribute that appears after this line of code will be animated every time it changes.

```
petal
    .transition()
    .style("opacity", 1)
    .duration(800);
```

That’s it!

Thanks :)