Interactive Web Programming

1st semester of 2021

Murilo Camargos
(**murilo.filho@fgv.br**)

Heavily based on **Victoria Kirst** slides

Schedule

Today:

- SVG: Scalable Vector Graphics
- D3: Data-driven Documents
  - Selections
  - Data joins

HW4 is out! Due May 11.

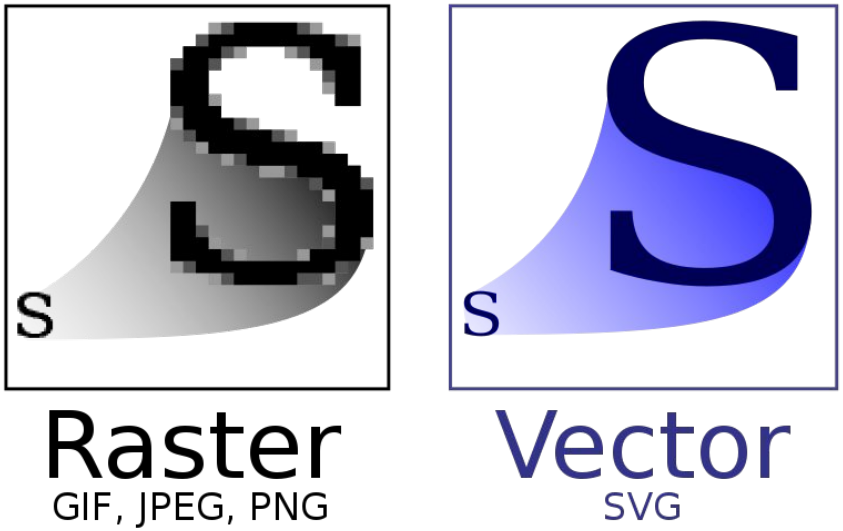
Credits:

- https://www.youtube.com/watch?v=_8V5o2UHG0E
- Intro to Data Viz at Ohio State University
  - http://web.cse.ohio-state.edu/~shen.94/5544/

SVG

SVG as an image format

SVG: Stands for **S**calable **V**ector **G**raphics



https://commons.wikimedia.org/wiki/File:Ghostscript_Tiger.svg

SVG with HTML

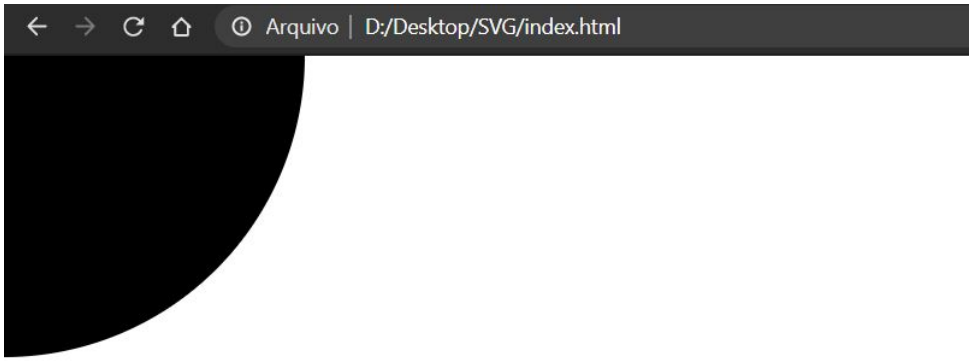
You can use the **svg** HTML tag to create an SVG canvas:

```
<body>
  <svg></svg>
</body>
```

You can use different shape SVG elements inside this tag:

- <circle>, <ellipse>, <line>, <polygon>, <polyline>, <rect>

```
<svg>
  <circle r="100"></circle>
</svg>
```



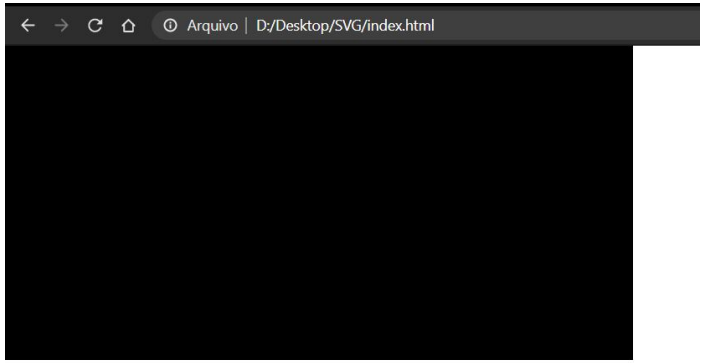
Codepen

SVG with HTML: canvas size

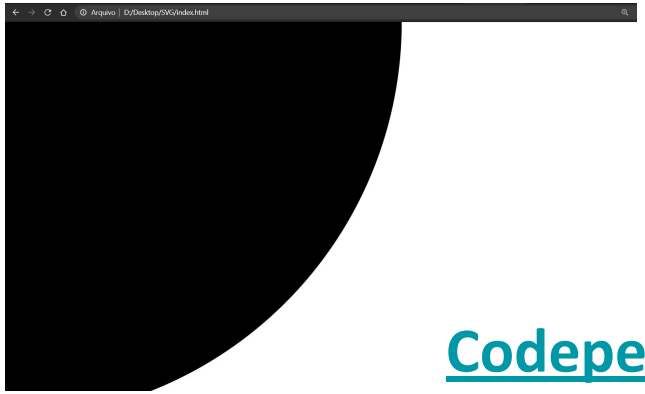
SVG's standard canvas size is **300×150**:

- Other browsers might implement 100vw and 100vh.

```
<svg>
  <circle r="400"></circle>
</svg>
```



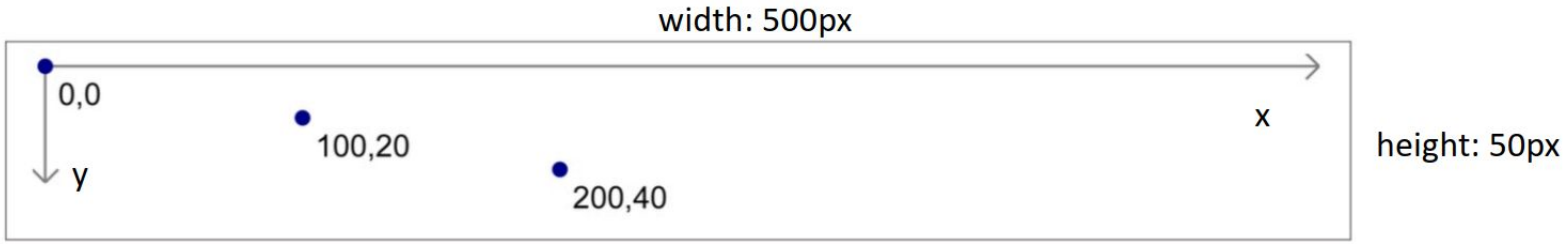

```
<svg width="400" height="400">
  <circle r="400"></circle>
</svg>
```



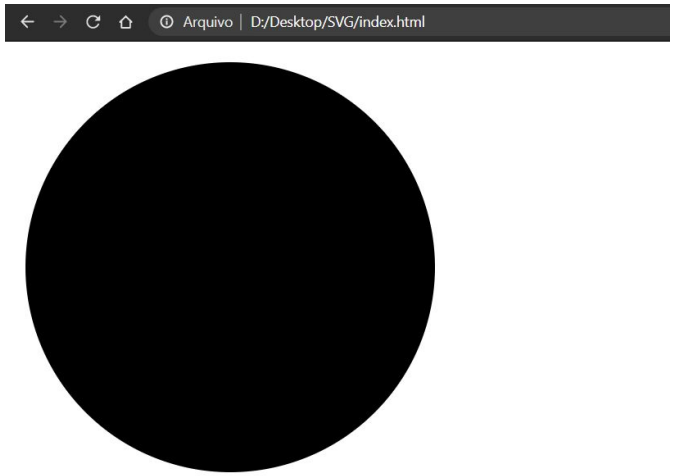
[Codepen]

SVG with HTML: coord. system

SVG's coordinate system:




```
<svg width="220" height="220">
  <circle r="100" cx="110" cy="110"></circle>
</svg>
```



Codepen

SVG with HTML and CSS

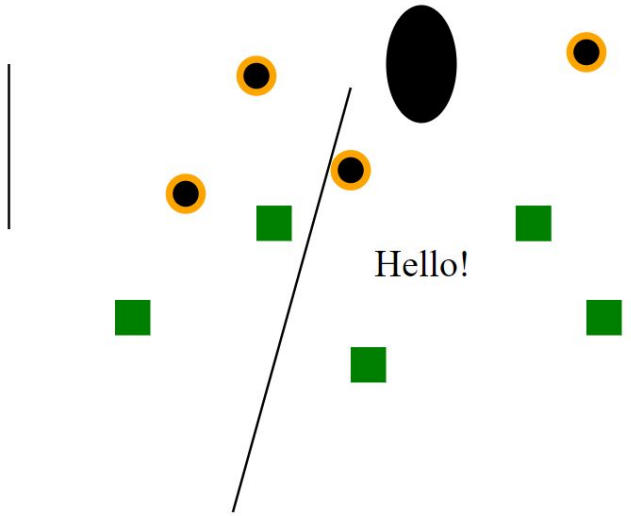
You can style the shapes with CSS code:

HTML

```
<svg width="400" height="220">
  <line x1="5" x2="5" y1="100" y2="30" stroke="black"></line>
  <line x1="100" x2="150" y1="220" y2="40" stroke="black"></line>

  <rect x="150" y="150" width="15" height="15"></rect>
  <rect x="220" y="90" width="15" height="15"></rect>
  <rect x="110" y="90" width="15" height="15"></rect>
  <rect x="220" y="90" width="15" height="15"></rect>
  <rect x="50" y="130" width="15" height="15"></rect>
  <rect x="250" y="130" width="15" height="15"></rect>

  <circle cx="250" cy="25" r="7"></circle>
  <circle cx="150" cy="75" r="7"></circle>
  <circle cx="80" cy="85" r="7"></circle>
  <circle cx="110" cy="35" r="7"></circle>
  <circle cx="150" cy="75" r="7"></circle>

  <ellipse cx="180" cy="30" rx="15" ry="25"></ellipse>

  <text x="160" y="120">Hello!</text>
</svg>
```

CSS

```
rect {
  fill: green;
}
circle {
  stroke: orange;
  stroke-width: 3px;
}
```



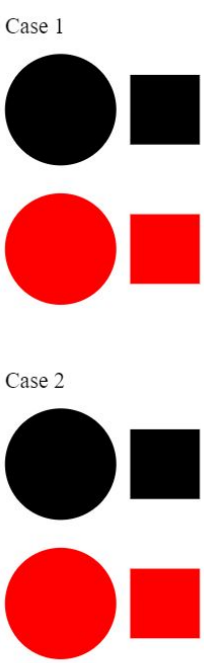
[Codepen]

SVG groups

You can group the shapes and apply rules to all of them at once:

HTML

```
<svg widht="300" height="250">
  <text x="10" y="15">Case 1</text>

  <circle cx="50" cy="70" r="40"></circle>
  <rect x="100" y="45" width="50" height="50"></rect>

  <circle cx="50" cy="170" r="40" fill="red"></circle>
  <rect x="100" y="145" width="50" height="50" fill="red"></rect>
</svg>
<br>
<svg widht="300" height="250">
  <text x="10" y="15">Case 2</text>

  <circle cx="50" cy="70" r="40"></circle>
  <rect x="100" y="45" width="50" height="50"></rect>

  <g fill="red" transform="translate(0,100)">
    <circle cx="50" cy="70" r="40"></circle>
    <rect x="100" y="45" width="50" height="50"></rect>
  </g>
</svg>
```



[Codepen]

SVG Paths

You can also create a **path**:

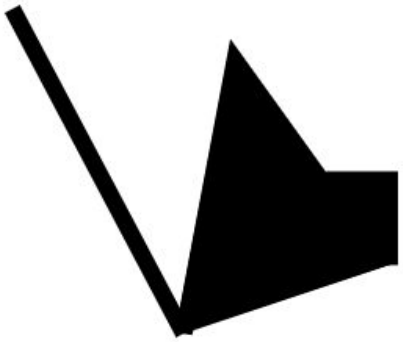
- M x y – Move to (x,y)
  - m dx dy – Move by (dx,dy)
- L x y – Line to (x,y)
  - l dx dy
- H x, V y – draw horizontal and vertical lines
  - h dx, v dy
- Z, z close path
- Curve commands (Bezier Curves and Arcs)

SVG Paths

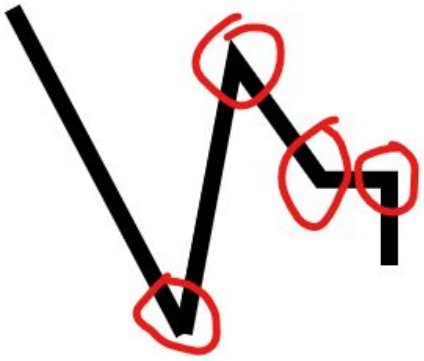
HTML

```
1  <svg widht="200" height="200">
2    <text x="10" y="15">Case 1</text>
3    <g stroke="black" stroke-width="5">
4      <line x1="10" y1="25" x2="60" y2="120"></line>
5      <path d="M60 120 L75 40 L100 75 H120 V100"></path>
6    </g>
7  </svg>
8  <br>
9  <svg widht="200" height="200">
10   <text x="10" y="15">Case 2</text>
11   <g stroke="black" stroke-width="5">
12     <line x1="10" y1="25" x2="60" y2="120"></line>
13     <path d="M60 120 L75 40 L100 75 H120 V100" fill="none"></path>
14   </g>
15 </svg>
```

Case 1



Case 2



[Codepen]

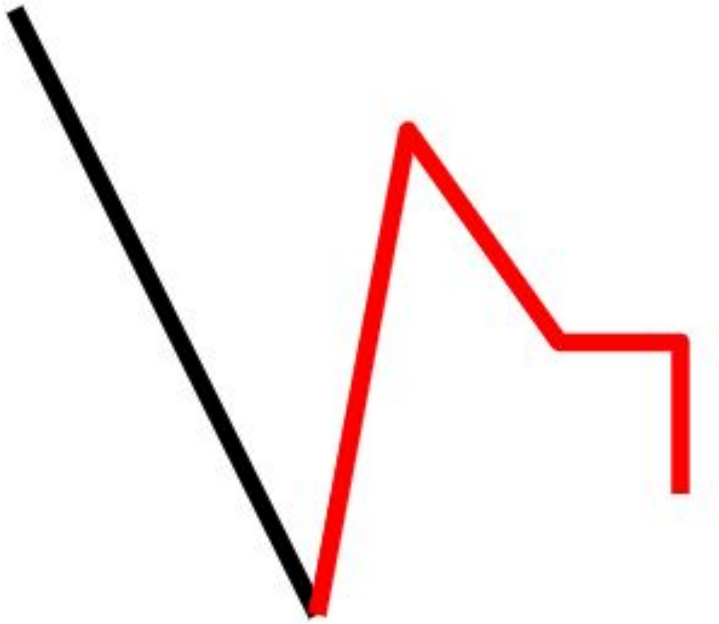
SVG Paths and CSS

HTML

```html
<svg widht="200" height="200">
  <text x="10" y="15">Case 3</text>
  <g class="lines">
    <line x1="10" y1="25" x2="60" y2="120"></line>
    <path d="M60 120 L75 40 L100 75 H120 V100"></path>
  </g>
</svg>
```

CSS

```css
.lines {
  stroke: black;
  stroke-width: 5;
}
.lines path {
  fill: none;
  stroke: red;
  stroke-linejoin: round;
}
```



[Codepen]

Introduction to D3

What is D3?

D3: Data-Driven Documents

- A **JS** library by Mike Bostock
- It allows you to **bind** arbitrary **data** to the **DOM**, and then apply data-driven **transformations** to the document.
- You can use it to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction.

How to load D3?

- The whole lib can be loaded from a single JS file:
  - https://d3js.org/d3.v6.min.js
  - https://cdnjs.cloudflare.com/ajax/libs/d3/6.7.0/d3.min.js
  - https://unpkg.com/d3@6.7.0/dist/d3.min.js
  - …
- When loaded, it'll introduce a global object **d3**

HTML

```
1 <html>
2   <head>
3     <script src="https://d3js.org/d3.v6.min.js" defer></script>
4   </head>
5   <body></body>
6 </html>
```

D3 Selections

- You can select DOM elements in an easy way with D3
- Say you want to change the text color of paragraph elements; we would do something like this:

```
const paragraphs = document.querySelectorAll("p");
for (const p of paragraphs) {
  p.style.color = 'blue';
}
```

- With D3, we could achieve the same result with:

```
d3.selectAll('p').style('color', 'blue');
```

Codepen

D3 Let’s make a face!

- First of all, we need to define our canvas:

HTML

```
1  <svg></svg>
```

JS

```
1  const svgWidth = 960;
2  const svgHeight = 500;
3
4  const svg = d3.select('svg');
5  svg.attr('width', svgWidth);
6  svg.attr('height', svgHeight);
7  svg.style('background-color', 'red');
```

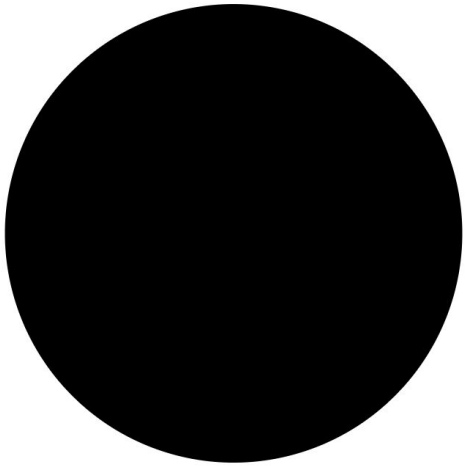
Codepen

D3 Let's make a face!

- Now we can add shapes to it using D3. A face starts with a circle!

JS

```
1  const svgWidth = 960;
2  const svgHeight = 500;
3
4  const svg = d3.select('svg')
5    .attr('width', svgWidth)
6    .attr('height', svgHeight);
7
8  const circle = svg.append('circle')
9    .attr('r', svgHeight/2)
10   .attr('cx', svgWidth/2)
11   .attr('cy', svgHeight/2);
12
```



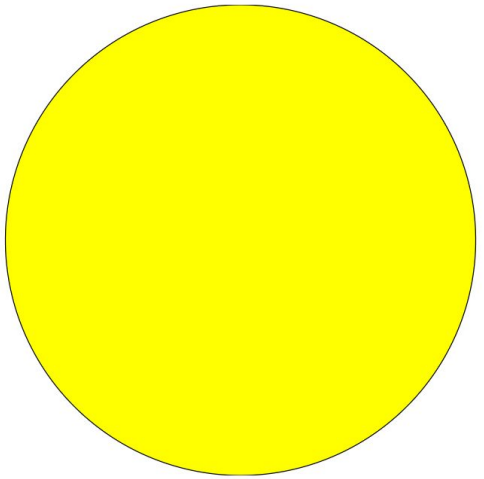
[Codepen]

D3 Let’s make a face!

- Now we can add shapes to it using D3. A face starts with a circle!

JS

```
const circle = svg.append('circle')
  .attr('r', svgHeight/2)
  .attr('cx', svgWidth/2)
  .attr('cy', svgHeight/2)
  .attr('fill', 'yellow')
  .attr('stroke', 'black');
```



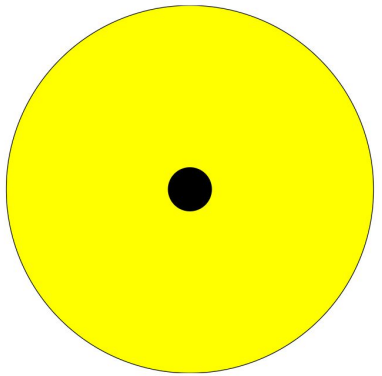
Codepen

D3 Let's make a face!

- Let's add the left eye:

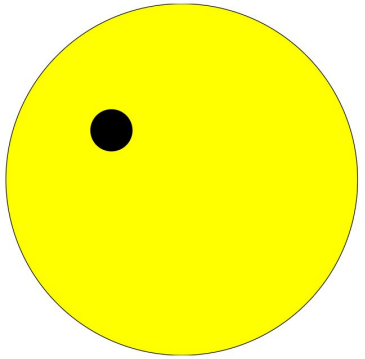
JS

```
14
15  const leftEye = svg.append('circle')
16    .attr('r', 30)
17    .attr('cx', svgWidth/2)
18    .attr('cy', svgHeight/2)
19    .attr('fill', 'black');
```



JS

```
14
15  const leftEye = svg.append('circle')
16    .attr('r', 30)
17    .attr('cx', svgWidth/2 - 100)
18    .attr('cy', svgHeight/2 - 70)
19    .attr('fill', 'black');
```



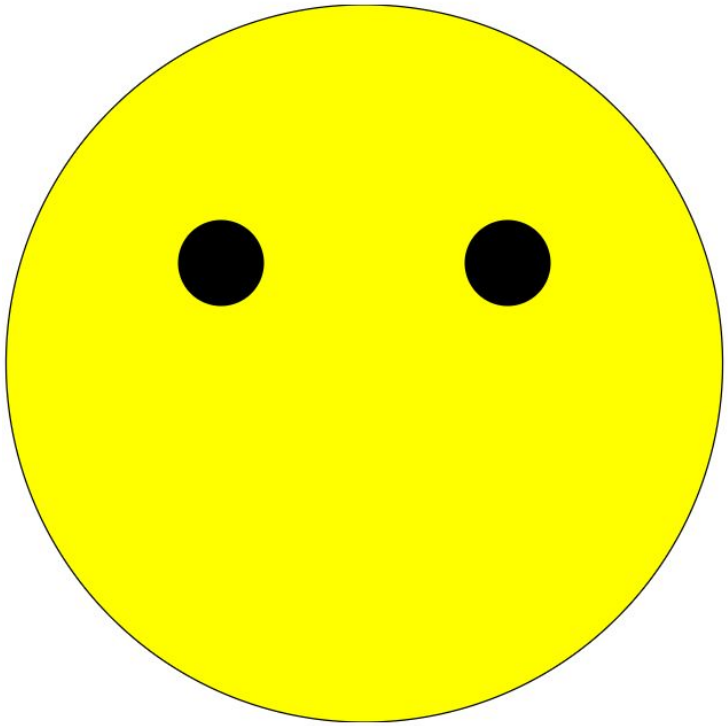
Codepen

D3 Let's make a face!

- And the right eye:

JS

```
15  const eyeSpacing = 100;
16  const eyeYOffset = -70;
17  const eyeRadius = 30;
18
19  const leftEye = svg.append('circle')
20    .attr('r', eyeRadius)
21    .attr('cx', svgWidth/2 - eyeSpacing)
22    .attr('cy', svgHeight/2 + eyeYOffset)
23    .attr('fill', 'black');
24
25  const rightEye = svg.append('circle')
26    .attr('r', eyeRadius)
27    .attr('cx', svgWidth/2 + eyeSpacing)
28    .attr('cy', svgHeight/2 + eyeYOffset)
29    .attr('fill', 'black');
30
```



Codepen

D3 Let's make a face!

- The face's mouth is not a known shape. It should be a path!
- Paths can be a bit challenging to specify using pure SVG.
- D3 has a friendly API that generates paths for us.
- You can generate arc path strings using D3:

```
31  const mouthArc = d3.arc()
32    .innerRadius(0)
33    .outerRadius(100)
34    .startAngle(0)
35    .endAngle(Math.PI);
```



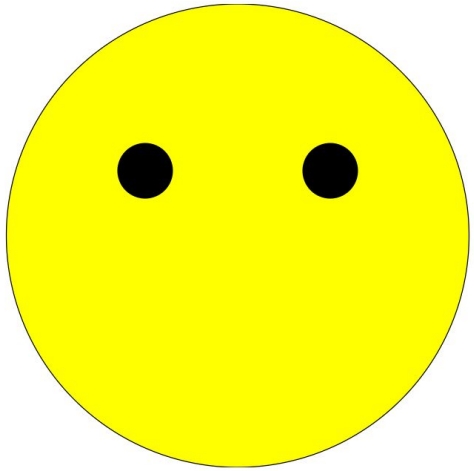
```
"M6.123233995736766e
-15,-100A100,100,0,1
,1,6.123233995736766
e-15,100L0,0Z"
```

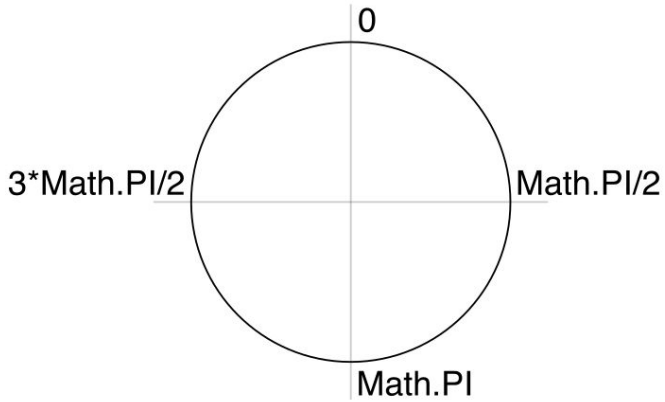
- It only generates the string. You still need to add it to the SVG.

D3 Let’s make a face!

- Paths generated by D3 API will start in (0,0)

```
31  const mouthArc = d3.arc()
32    .innerRadius(0)
33    .outerRadius(170)
34    .startAngle(Math.PI/2)
35    .endAngle(3*Math.PI/2);
36
37  const mouth = svg.append('path')
38    .attr('d', mouthArc);
```





D3 Let's make a face!

- You need to add the mouth to a group and translate it.

```
const mouthArc = d3.arc()
  .innerRadius(160)
  .outerRadius(170)
  .startAngle(Math.PI/2)
  .endAngle(3*Math.PI/2);

const g = svg.append('g')
  .attr('transform', `translate(${svgWidth/2},${svgHeight/2})`);

const mouth = g.append('path')
  .attr('d', mouthArc);
```

Codepen

- Template literals allows you to insert variables in a string.

D3 Let's make a face!

- We can use the group for all components to remove some repeating code.

Codepen

JS

```
const svgWidth = 960;
const svgHeight = 500;

const svg = d3.select('svg')
  .attr('width', svgWidth)
  .attr('height', svgHeight);

const g = svg.append('g')
  .attr('transform', `translate(${svgWidth/2},${svgHeight/2})`);

const circle = g.append('circle')
  .attr('r', svgHeight/2)
  .attr('fill', 'yellow')
  .attr('stroke', 'black');

const eyeSpacing = 100;
const eyeYOffset = -70;
const eyeRadius = 30;
```

```
const leftEye = g.append('circle')
  .attr('r', eyeRadius)
  .attr('cx', -eyeSpacing)
  .attr('cy', eyeYOffset)
  .attr('fill', 'black');

const rightEye = g.append('circle')
  .attr('r', eyeRadius)
  .attr('cx', eyeSpacing)
  .attr('cy', eyeYOffset)
  .attr('fill', 'black');

const mouthArc = d3.arc()
  .innerRadius(160)
  .outerRadius(170)
  .startAngle(Math.PI/2)
  .endAngle(3*Math.PI/2);

const mouth = g.append('path')
  .attr('d', mouthArc);
```

D3 Let's make a face!

- We can also nest groups to avoid code repetition in the eyes.

Codepen

```
16  const eyeSpacing = 100;
17  const eyeYOffset = -70;
18  const eyeRadius = 30;
19
20  const leftEye = g.append('circle')
21    .attr('r', eyeRadius)
22    .attr('cx', -eyeSpacing)
23    .attr('cy', eyeYOffset)
24    .attr('fill', 'black');
25
26  const rightEye = g.append('circle')
27    .attr('r', eyeRadius)
28    .attr('cx', eyeSpacing)
29    .attr('cy', eyeYOffset)
30    .attr('fill', 'black');
```

```
16  const eyeSpacing = 100;
17  const eyeYOffset = -70;
18  const eyeRadius = 30;
19
20  const eyeGroup = g.append('g')
21    .attr('transform', `translate(0,${eyeYOffset})`);
22
23  const leftEye = eyeGroup.append('circle')
24    .attr('r', eyeRadius)
25    .attr('cx', -eyeSpacing);
26
27  const rightEye = eyeGroup.append('circle')
28    .attr('r', eyeRadius)
29    .attr('cx', eyeSpacing);
```

D3 Let's make a face!

- Let's add eyebrows to our face

```
31  const eyebrowWidth = 50;
32  const eyebrowHeight = 20;
33
34  const leftEyebrow = eyeGroup.append('rect')
35    .attr('width', eyebrowWidth)
36    .attr('height', eyebrowHeight);
```

```
31  const eyebrowWidth = 60;
32  const eyebrowHeight = 15;
33
34  const leftEyebrow = eyeGroup.append('rect')
35    .attr('width', eyebrowWidth)
36    .attr('height', eyebrowHeight)
37    .attr('x', -eyeSpacing - eyebrowWidth/2)
38    .attr('y', -60);
```

[Codepen]

A **very** quick intro to D3 transitions

D3 Let's make a face!

- We can change the selection to perform a transition.
- Also see the new indentation pattern

```
34  const leftEyebrow = eyeGroup
35    .append('rect')
36      .attr('width', eyebrowWidth)
37      .attr('height', eyebrowHeight)
38      .attr('x', -eyeSpacing - eyebrowWidth/2)
39      .attr('y', eyebrowYOffset)
40    .transition()
41      .attr('y', eyebrowYOffset-30);
```

```
34  const leftEyebrow = eyeGroup
35    .append('rect')
36      .attr('width', eyebrowWidth)
37      .attr('height', eyebrowHeight)
38      .attr('x', -eyeSpacing - eyebrowWidth/2)
39      .attr('y', eyebrowYOffset)
40    .transition().duration(2000)
41      .attr('y', eyebrowYOffset-30);
```

Codepen

D3 Let's make a face!

- To move both eyebrows, first let us create an eyebrow group. We can get rid of the "y" attr.

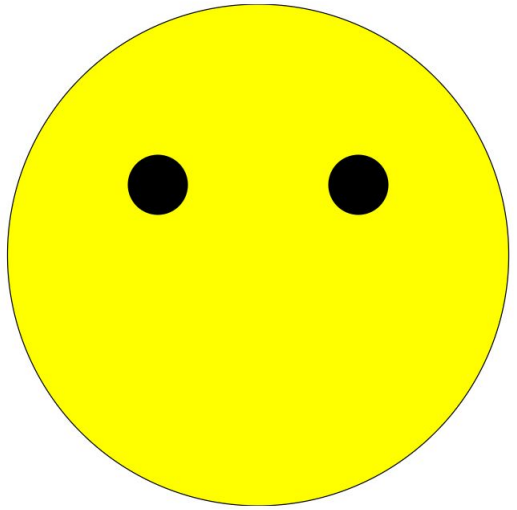
```
40  const eyebrowGroup = eyeGroup
41    .append('g')
42      .attr('transform', `translate(0,${eyebrowYOffset})`);
43
44  const leftEyebrow = eyebrowGroup
45    .append('rect')
46      .attr('width', eyebrowWidth)
47      .attr('height', eyebrowHeight)
48      .attr('x', -eyeSpacing - eyebrowWidth/2);
49
50  const rightEyebrow = eyebrowGroup
51    .append('rect')
52      .attr('width', eyebrowWidth)
53      .attr('height', eyebrowHeight)
54      .attr('x', eyeSpacing - eyebrowWidth/2);
```

[Codepen]

D3 Let's make a face!

- Now let's transition the entire group. We need to consider "transform" instead of "y" now.
- Note we can chain transitions too.

```
40  const eyebrowGroup = eyeGroup
41    .append('g')
42      .attr('transform', `translate(0,${eyebrowYOffset})`)
43    .transition().duration(2000)
44      .attr('transform', `translate(0,${eyebrowYOffset-30})`)
45    .transition().duration(2000)
46      .attr('transform', `translate(0,${eyebrowYOffset})`)
47
48  const leftEyebrow = eyebrowGroup
49    .append('rect')
50      .attr('width', eyebrowWidth)
51      .attr('height', eyebrowHeight)
52      .attr('x', -eyeSpacing - eyebrowWidth/2);
```



Codepen

D3 Let's make a face!

- Isolate eyebrow group

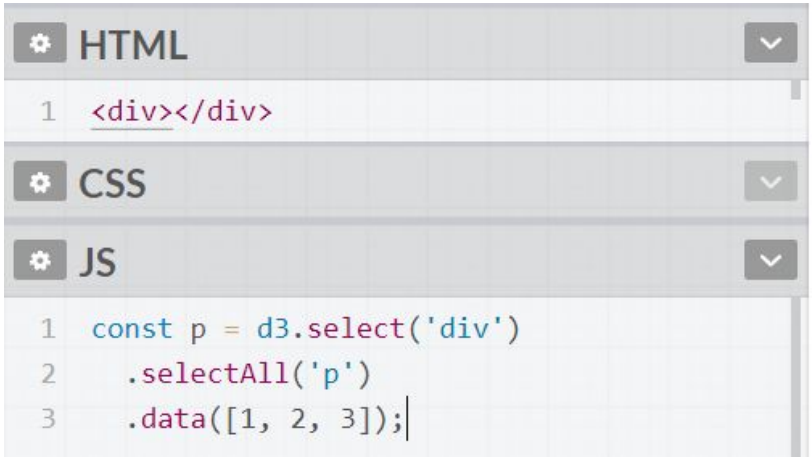
```
40  const eyebrowGroup = eyeGroup
41    .append('g')
42      .attr('transform', `translate(0,${eyebrowYOffset})`);
43
44  eyebrowGroup
45    .transition().duration(2000)
46      .attr('transform', `translate(0,${eyebrowYOffset-30})`)
47    .transition().duration(2000)
48      .attr('transform', `translate(0,${eyebrowYOffset})`)
49
50  const leftEyebrow = eyebrowGroup
51    .append('rect')
52      .attr('width', eyebrowWidth)
53      .attr('height', eyebrowHeight)
54      .attr('x', -eyeSpacing - eyebrowWidth/2);
```

[Codepen]

Data binding with D3

D3 Data Bind

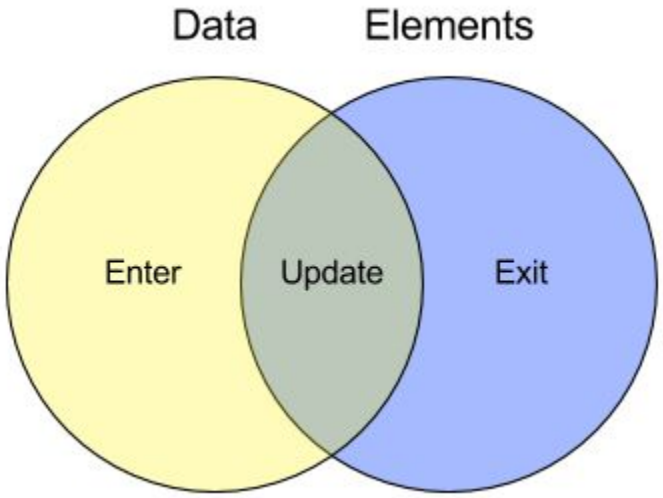
- We can Bind data to a D3 selection by calling the method “data” (even if the DOM elements does not exist yet):



- When data is bound to a selection, each element in the data array is paired with the corresponding node in the selection.

D3 Data Join

- D3 uses a declarative style of programming to bind data to DOM elements and returns three virtual selections:
  - **Enter:** New data, no selection elements
  - **Update:** Data item mapped to existing selection element
  - **Exit:** Selection elements with no data item mapped to them



https://www.amphinicy.com/blog/post/manipulating-svg-using-d3js-library

D3 Data Join: enter

- Since we don’t have any “p” inside our “div”, the join must happen at the “enter” stage:

HTML

```
1  <div></div>
```

CSS

JS

```
1  const p = d3.select('div')
2    .selectAll('p')
3    .data([1, 2, 3])
4    .enter().append('p')
5      .text(d => `Paragraph ${d}`);
```

Paragraph 1

Paragraph 2

Paragraph 3

[Codepen]

D3 Data Join: update

- What if we had one “p” inside our “div”?

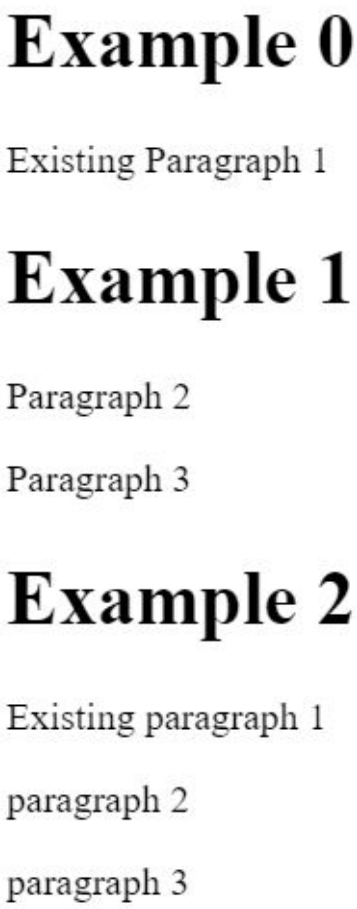
HTML

```html
<h1>Example 0</h1>
<div id="ex0">
  <p></p>
</div>

<h1>Example 1</h1>
<div id="ex1">
  <p></p>
</div>

<h1>Example 2</h1>
<div id="ex2">
  <p></p>
</div>
```

JS

```js
const ex0 = d3.select('#ex0')
  .selectAll('p')
  .data([1, 2, 3])
    .text(d => `Existing Paragraph ${d}`);

const ex1 = d3.select('#ex1')
  .selectAll('p')
  .data([1, 2, 3])
  .enter().append('p')
    .text(d => `Paragraph ${d}`);

const ex2 = d3.select('#ex2')
  .selectAll('p')
  .data([1, 2, 3])
    .text(d => `Existing paragraph ${d}`);

ex2.enter().append('p')
  .text(d => `paragraph ${d}`);
```

[Codepen]



D3 Data Join: exit

- What if we had more than three “p”s inside our “div”?

HTML

```
<div>
  <p></p>
  <p></p>
  <p></p>
</div>
```

JS

```
const div = d3.select('div')
  .selectAll('p')
  .data([1, 2, 3])
    .text(d => `Paragraph ${d}`);
```

Paragraph 1

Paragraph 2

Paragraph 3

```
<div>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  <p>Paragraph 3</p>
  <p></p>
</div>
```

JS

```
const div = d3.select('div')
  .selectAll('p')
  .data([1, 2, 3])
    .text(d => `Paragraph ${d}`);

div.exit().remove();
```

Paragraph 1

Paragraph 2

Paragraph 3

```
<div>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  <p>Paragraph 3</p>
</div>
```

Codepen