Information Visualization

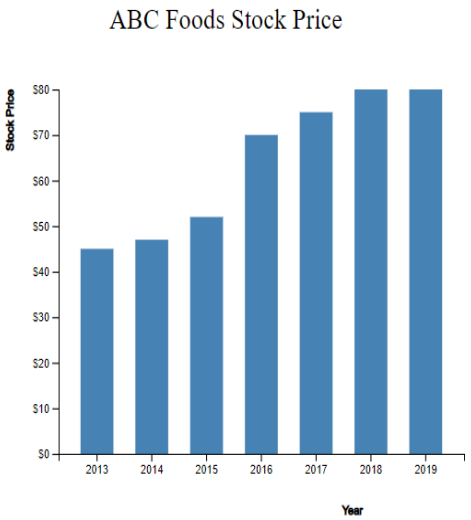
Course Module IN6221

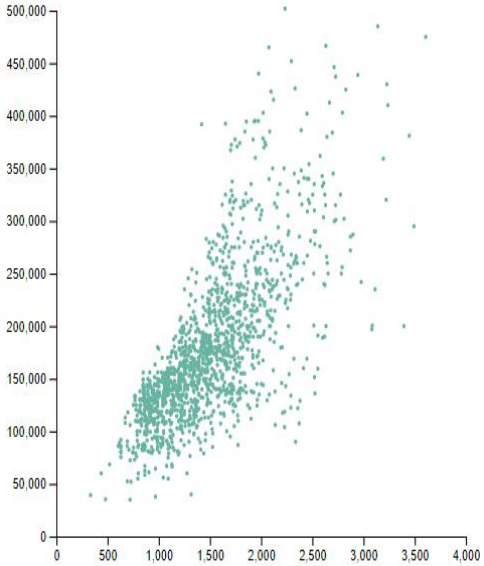
D3 Visualization Tool

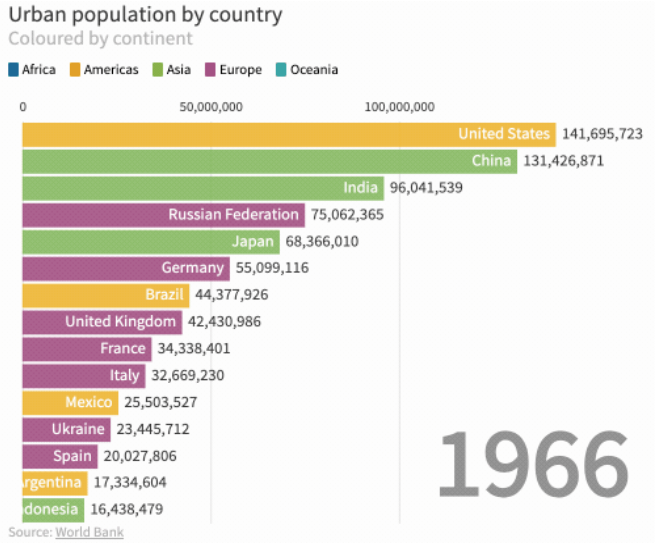
WKW School of Communication and Information,
NTU

We are going to:

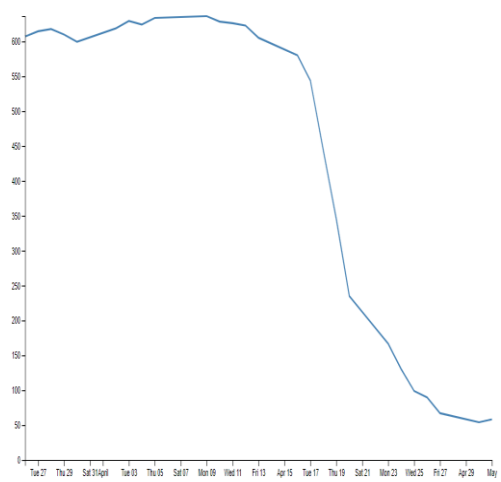
Create the Charts

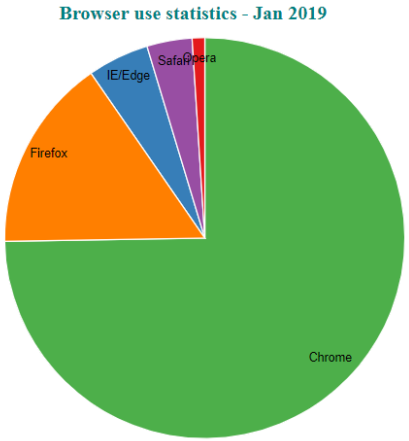




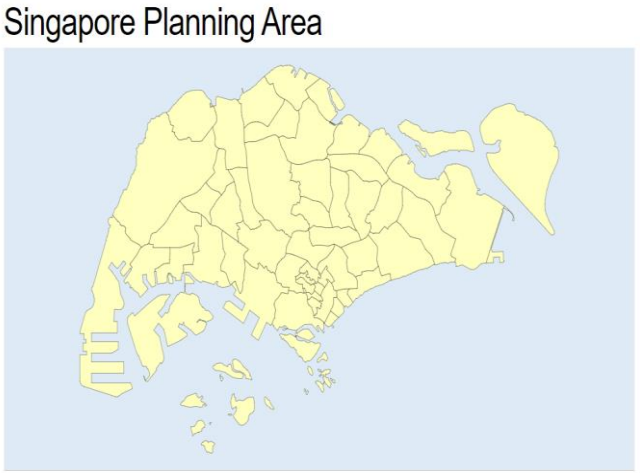


Create Animations





Draw Map Visualization



Setting up the Web Server

- To install Python (https://www.python.org/downloads/)
- To run a **Python web server**:

  1. Open a new terminal window.

  2. Via the command line, navigate to the directory that you want served. For example, if your project folder is in your Desktop folder, you could type: **cd ~/Desktop/project-folder**

  3. For Python 2.x enter >> python -m SimpleHTTPServer

  4. For Python 3.x enter >> **python3 -m http.server --bind localhost (or py –m http.server)**

- Switch back to the web browser and visit the URL: *http://localhost:8000/*

must start with this

SVG – Scalable Vector Graphics

可扩展的矢量图形

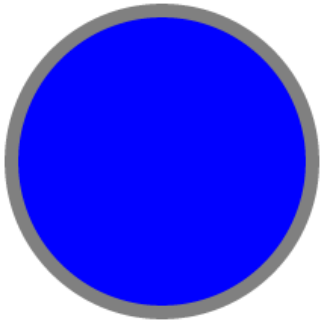
- D3 is most useful when used to generate and manipulate visuals as **Scalable Vector Graphics (SVG)** - SVG is used to define graphics for the Web - essentially, to **graphics** what **HTML** is to **text**.
- SVG is a **text-based image format**. Each SVG image is defined using **Extensible Markup Language (XML)** markup code.

Create **SVG element** of width=50, height=50 (in **pixels** by default)

Specify the width and height attributes values. Otherwise, SVG will behave like a greedy, block-level HTML element and **take up as much room** as it can

```
<svg width="50" height="50">
    <circle cx="25" cy="25" r="22" fill="blue" stroke="gray" stroke-width="2"/>
</svg>
```

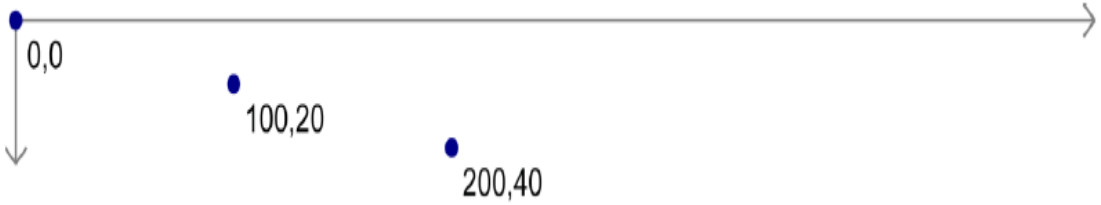
To create a circle centred at x=25, y=25 coordinates, radius=22, filled with blue, with gray stroke width=2



SVG – Simple Shapes - Rectangle

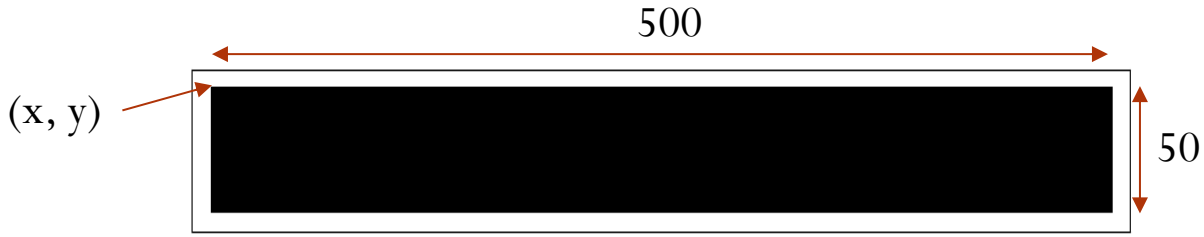
- **Visual elements** that can be created under **svg tags** include **rect, circle, ellipse, line, text, and path.**

- Pixel-based coordinates system with 0,0 at **top-left corner** of the drawing space. **Increasing x** values move to the **right**, while **increasing y** values move **down**



- To draw a **rectangle (rect)**, **x** and **y** specify the **coordinates** of the **upper-left corner**, and **width** and **height** specify the **dimensions**.

```
<rect x="0" y="0" width="500" height="50"/>
```



SVG – Simple Shapes - Circle

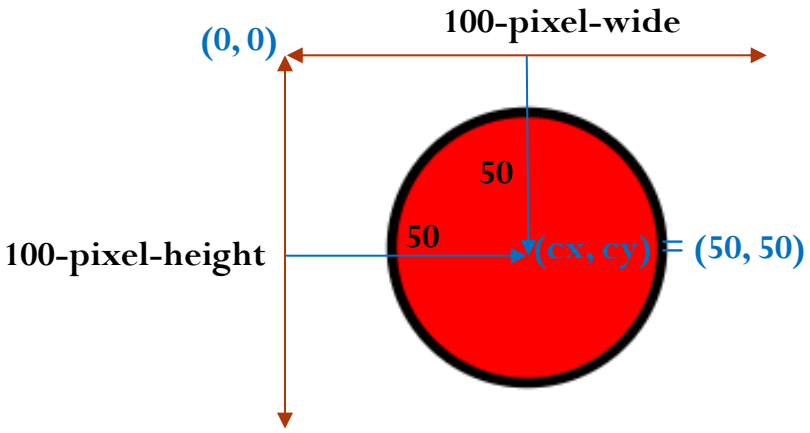
- Use **cx** and **cy** to specify the **coordinates** of the **center**, and **r** to specify the **radius**.

Stroke: 描边,
stroke-width 描边宽度

- Example circle is centered in the middle of **100-pixel height and width** SVG as its **cx** ("center-x") value is 50, **cy** ("center-y") value is 50.

```
<svg height="100" width="100">
  <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
  Sorry, your browser does not support inline SVG.
</svg>
```

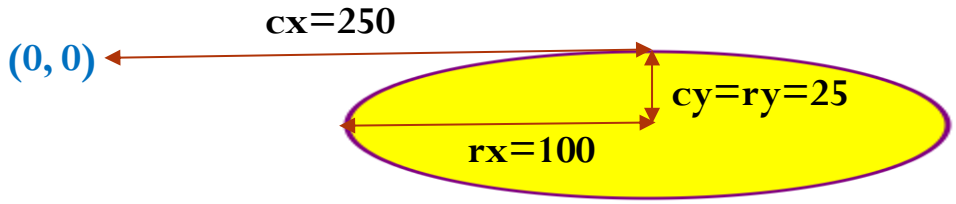
SVG display value when image cannot be created



SVG – Simple Shapes – Ellipse and Line

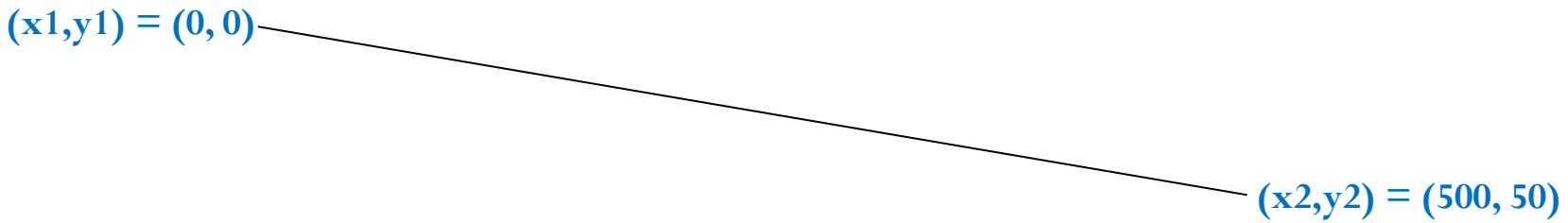
- **Ellipse** is similar to circle, but expects **separate radius values** for each axis. Instead of r, use **rx and ry**

```
<ellipse cx="250" cy="25" rx="100" ry="25"/>
```



- For a **line**, use **x1** and **y1** to specify the coordinates of **one end** of the line, and **x2** and **y2** to specify the coordinates of the **other end**. A **stroke color** must be specified for the **line to be visible**

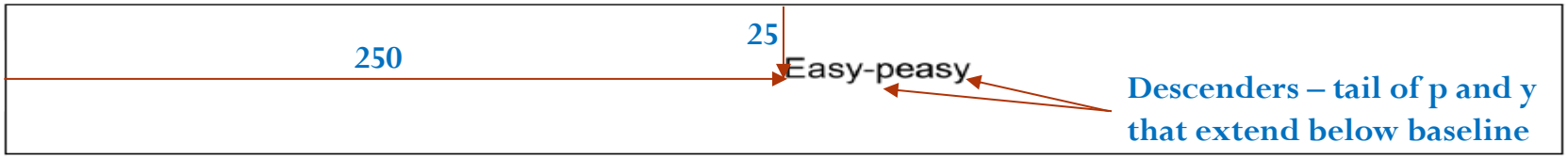
```
<line x1="0" y1="0" x2="500" y2="50" stroke="black"/>
```



SVG – Text

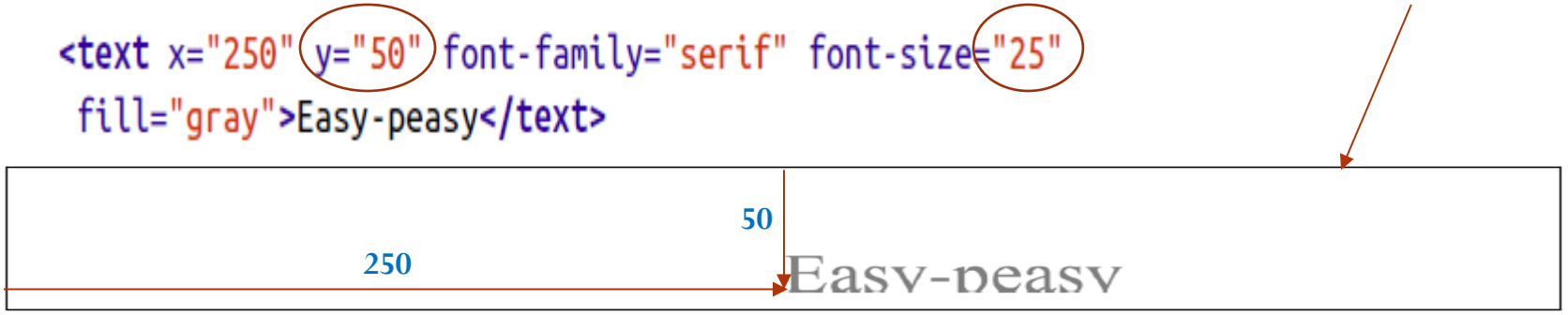
- The **<text>** element within **<svg>** tags defines a text. Use **x** to specify the position of the **left edge**, and **y** to specify the **vertical** position of the type's **baseline**.

```
<text x="250" y="25">Easy-peasy</text>
```



- (**Baseline** is a typographical term for the **invisible line** on which the letters **appear to rest**. Portions of letters such as "p" and "y" that **extend below the baseline** are called **descenders**.)
- When using text make sure the descenders **don't get cut off by svg borders**

```
<text x="250" y="50" font-family="serif" font-size="25"
 fill="gray">Easy-peasy</text>
```



SVG – Simple Shapes - Path

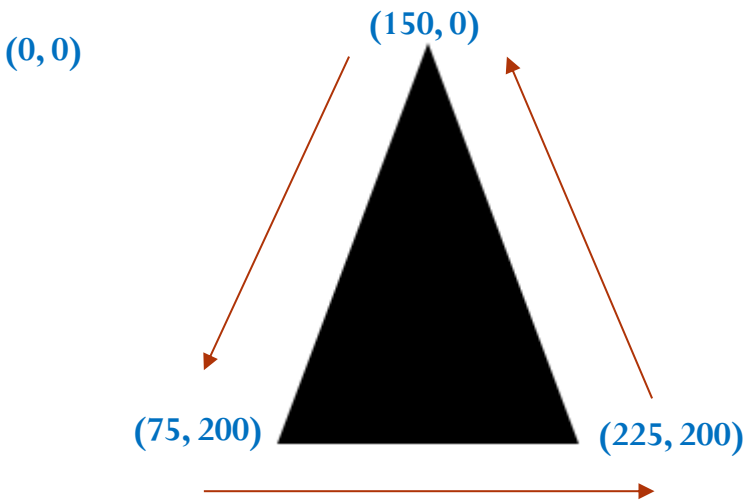
- **Path** is for drawing anything **more complex** than the preceding **shapes** (like country outlines for **geomaps**)

Define a path that starts at position **150, 0** with a line to position **75, 200** then from there, a line to **225, 200** and finally closing the path back to **150, 0**:

```
<!DOCTYPE html>
<html>
<body>

<svg height="210" width="400">
  <path d="M150 0 L75 200 L225 200 Z" />
  Sorry, your browser does not support inline SVG.
</svg>

</body>
</html>
```

The following commands are available for path data:

- M = moveto (without drawing)
- L = lineto
- H = horizontal lineto
- V = vertical lineto
- C = curveto
- S = smooth curveto
- Q = quadratic Bézier curve
- T = smooth quadratic Bézier curveto
- A = elliptical Arc
- Z = closepath



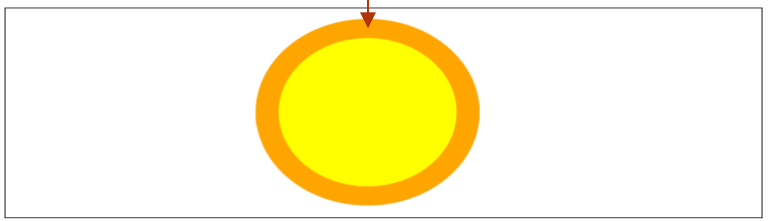
Styling SVG Elements

- SVG's **default style** is a **black** fill (of object) with **no stroke**. Examples of **SVG style properties**:
  - **fill** - A color value. Just as with CSS, colors can be specified as named colors, hex values, or RGB or RGBA (Alpha – transparency) values - https://www.w3schools.com/colors/colors_picker.asp
  - **stroke** - color value of stroke
  - **stroke-width** - A numeric measurement (typically in pixels). **Default 1px (0.26mm)**
  - **opacity** - A numeric value between 0.0 (**completely transparent**) and 1.0 (**completely opaque**).

```
<circle cx="25" cy="25" r="22"
 fill="yellow" stroke="orange" stroke-width="5"/>
```

Fill color value

Stroke color value



Styling SVG Elements

- We can strip the style attributes and **assign the circle a class.**

```
<circle cx="25" cy="25" r="22" class="pumpkin"/>
```

- then put the fill, stroke, and stroke-width rules into a **CSS style** that **targets the new class**:

```
.pumpkin {
    fill: yellow;
    stroke: orange;
    stroke-width: 5;
}
```

- **CSS/Class** approach benefits:
  - specify a style once and have it applied to **multiple elements**.
  - CSS code is **easier to read** than inline attributes.
  - more **maintainable** and make **design changes faster** to implement.

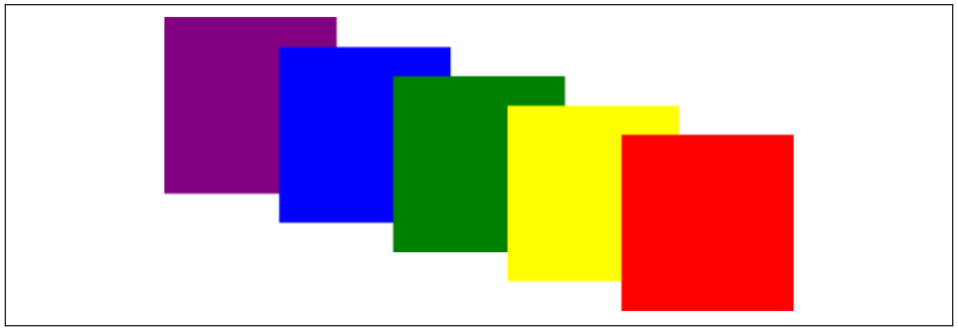
Styling SVG Elements - Layering and Drawing Order

- There are **no "layers" in SVG** and no real concept of depth. Shapes can be arranged only within the **two-dimensional x/y plane**.
- If multiple shapes are drawn, they can overlap:

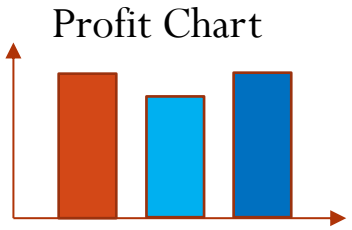
```
<rect x="0" y="0" width="30" height="30" fill="purple"/>
<rect x="20" y="5" width="30" height="30" fill="blue"/>
<rect x="40" y="10" width="30" height="30" fill="green"/>
<rect x="60" y="15" width="30" height="30" fill="yellow"/>
<rect x="80" y="20" width="30" height="30" fill="red"/>
```

The **order in which elements are coded** determines their **depth order**.

元素编码的顺序决定了它们的深度顺序。



E.g., **axes** and **labels** should be added to the SVG **last**. **Why?**



Styling SVG Elements - Transparency

- **Transparency** - used when elements **overlap** but must remain **visible** or used to **deemphasize** some elements while **highlighting** others.
- Two ways to apply transparency: use an **RGB color with alpha (rgba)**, or set an **opacity** value.

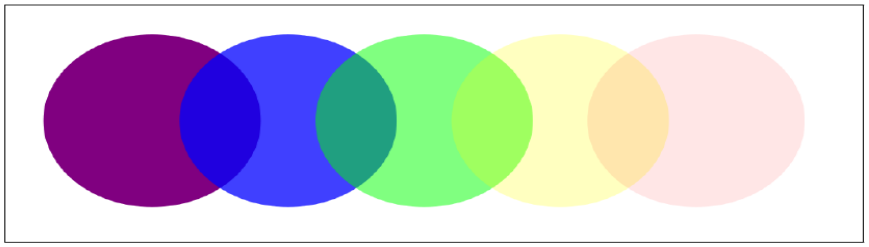
```
<circle cx="150" cy="50" r="40" fill="black" opacity="0.3" />
```

- **rgba()** expects three values between 0 and 255 for red, green, and blue, plus an **alpha (transparency)** value between 0.0 (full transparency) and 1.0 (full opaque)

```
<circle cx="25" cy="25" r="20" fill="rgba(128, 0, 128, 1.0)"/>   → purple
<circle cx="50" cy="25" r="20" fill="rgba(0, 0, 255, 0.75)"/>    → blue
<circle cx="75" cy="25" r="20" fill="rgba(0, 255, 0, 0.5)"/>     → green
<circle cx="100" cy="25" r="20" fill="rgba(255, 255, 0, 0.25)"/> → yellow
<circle cx="125" cy="25" r="20" fill="rgba(255, 0, 0, 0.1)"/>    → red
```

Different center points

Alpha transparency value



Styling SVG Elements - Transparency

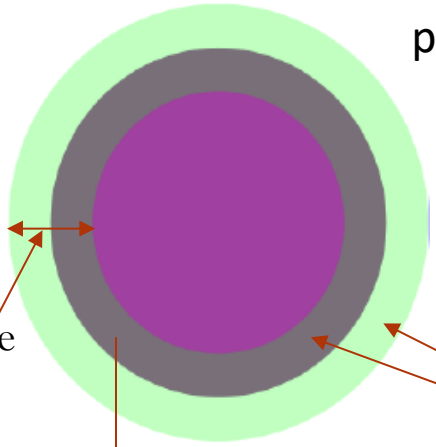
- With **rgba()**, transparency is applied **independently** to the **fill** and **stroke** colors.

red + blue => purple

```
<circle cx="25" cy="25" r="20" fill="rgba(128, 0, 128, 0.75)"
        stroke="rgba(0, 255, 0, 0.25)" stroke-width="10"/>
```

Circle fill is 75 percent opaque

Green stroke is 25 percent opaque



Full stroke = 10px

Green (**half of stroke**) mix with **purple** => gives **grayish** ½ stroke

The **strokes** are **aligned** to the **center of** each shape's **edge** => stroke is neither fully inside nor outside the object, but is both **half in** and **half out**. As a result of this transparency, the 10px stroke looks more like **two separate 5px strokes**.

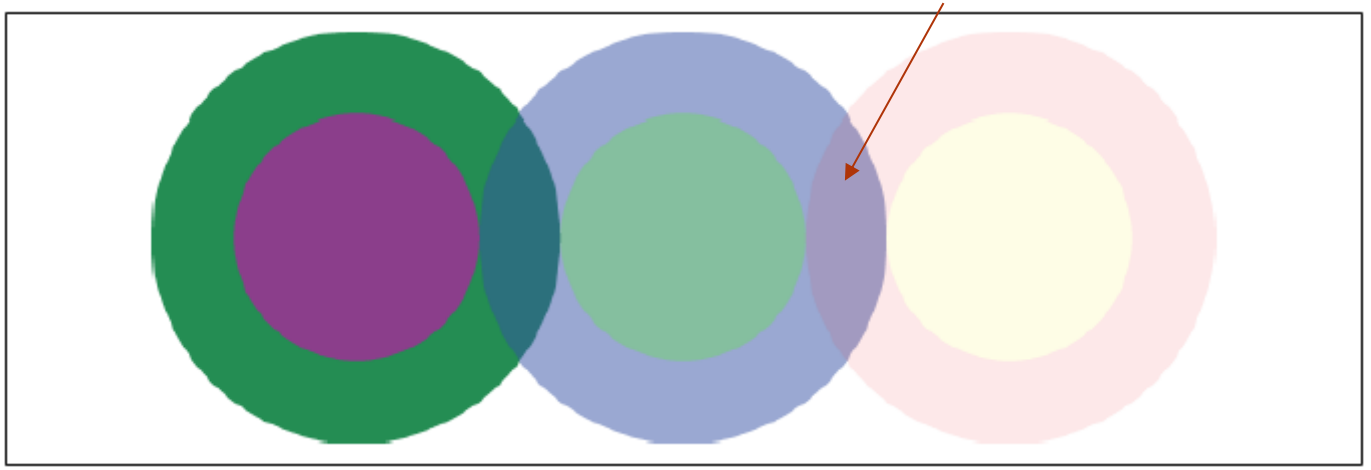
笔画与每个形状边缘的中心对齐=>笔画既不完全在物体内部，也不完全在物体外部，而是一半在和一半外。由于这种透明度，10px笔画看起来更像两个单独的5px笔画。

Styling SVG Elements - Transparency

- To apply transparency to an **entire element**, set an **opacity** attribute.

```
<circle cx="25" cy="25" r="20" fill="purple" stroke="green" stroke-width="10"
      opacity="0.9"/>
<circle cx="65" cy="25" r="20" fill="green" stroke="blue" stroke-width="10"
      opacity="0.5"/>
<circle cx="105" cy="25" r="20" fill="yellow" stroke="red" stroke-width="10"
      opacity="0.1"/>
```

Transparency (opacity=0.1) of third element allows second element to be seen



Stroke follows entire element opacity => no border line formed

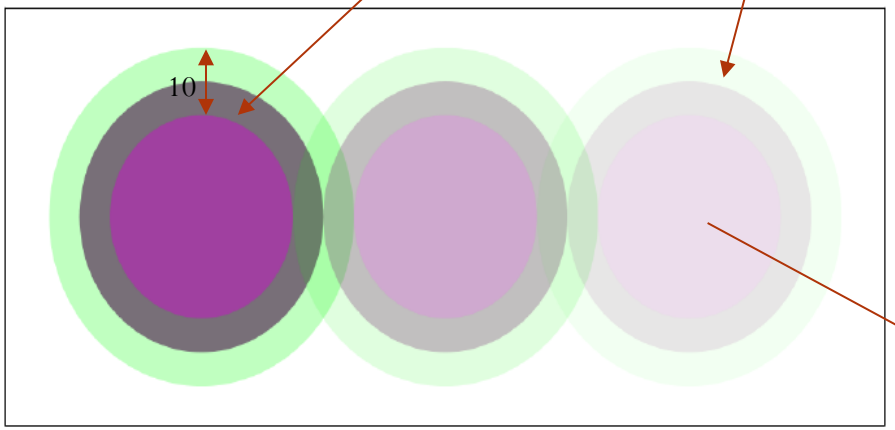
Styling SVG Elements - Transparency

- Employ **opacity** on an element that also has **colors set with rgba()**. When doing so, the **transparencies** are **multiplied**.

```
<circle cx="25" cy="25" r="20" fill="rgba(128, 0, 128, 0.75)"
        stroke="rgba(0, 255, 0, 0.25)" stroke-width="10"/>
<circle cx="65" cy="25" r="20" fill="rgba(128, 0, 128, 0.75)"
        stroke="rgba(0, 255, 0, 0.25)" stroke-width="10" opacity="0.5"/>
<circle cx="105" cy="25" r="20" fill="rgba(128, 0, 128, 0.75)"
        stroke="rgba(0, 255, 0, 0.25)" stroke-width="10" opacity="0.2"/>
```

10-width stroke – mid inside circle - Green combine with purple – give grey

Stroke opacity => 0.25 x 0.2 = 0.05



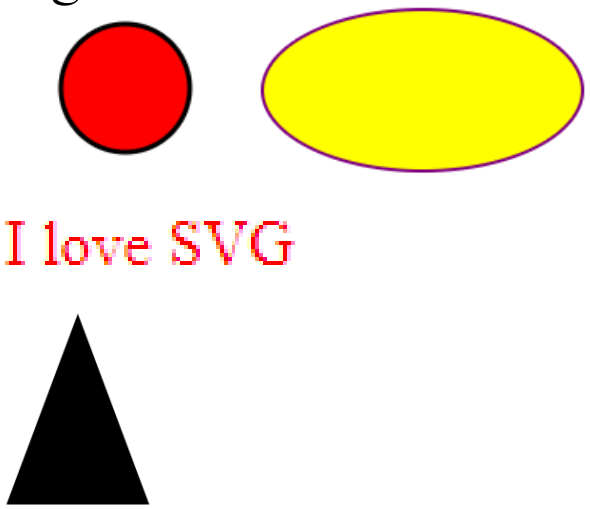
The third circle's **opacity** is 0.2, or 20 percent. Yet its purple **fill** already has an **alpha** value of 0.75, or 75 percent. The purple area, then, has a final transparency of **0.2 x 0.75 = 0.15**, or 15 percent.

第三个圆的不透明度为0.2，或20%。然而，它的紫色填充物已经具有0.75或75%的α值。然后，紫色区域的最终透明度为0.2 x 0.75 = 0.15，或15%。

Class Exercise - SVG

- Open any text editor – e.g., Notepad++

- Edit the codes in following HTML files to draw the following SVG:
  - 12_SVG_Circle.html
  - 13_SVG_Ellipse.html
  - 14_SVG_Text.html
  - 15_SVG_Path.html



SVG Attributes Reference

https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute

```
<svg height="100" width="100">
  <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
  Sorry, your browser does not support inline SVG.
</svg>
```

```
<svg height="140" width="500">
  <ellipse cx="200" cy="80" rx="100" ry="50" style="fill:yellow;stroke:purple;stroke-width:2" />
  Sorry, your browser does not support inline SVG.
</svg>
```

```
<svg height="60" width="200">
  <text x="0" y="15" fill="red">I love SVG</text>
  Sorry, your browser does not support inline SVG.
</svg>
```

Need text

```
<svg height="210" width="400">
  <path d="M150 0 L75 200 L225 200 Z" />
  Sorry, your browser does not support inline SVG.
</svg>
```

need both X-axis and Y-axis

What is D3?

- **D3** also referred to as **d3.js** is a **JavaScript library** for producing **highly customizable (bespoke)**, dynamic, and interactive data visualizations in web browsers based on web standards like **SVG, HTML** and **CSS**

- D3 refers to **Data-Driven Documents – data** driven - uses static data or fetch from remote server in formats such as Arrays, Objects, CSV, JSON, XML, etc for Web-based **documents** (HTML and SVG) that can be rendered by a web browser and D3 does the **driving** – it **connects the data to the documents**.

D3是指数据驱动文档——数据驱动——使用静态数据或以数组、对象、CSV、JSON、XML等格式从远程服务器获取基于Web的文档（HTML和SVG），这些文档可以由Web浏览器渲染，D3进行驱动——它将数据连接到文档。

- Entirely open source and freely available on Github - https://github.com/d3. Official website is https://d3js.org/.

- D3 facilitates generation and manipulation of web documents with data by
  - **Loading data** into browser's memory
  - **Binding data to elements** within the document, creating new elements as needed
  - **Transforming (map)** those **elements** by interpreting each element's bound datum and setting its **visual properties** accordingly

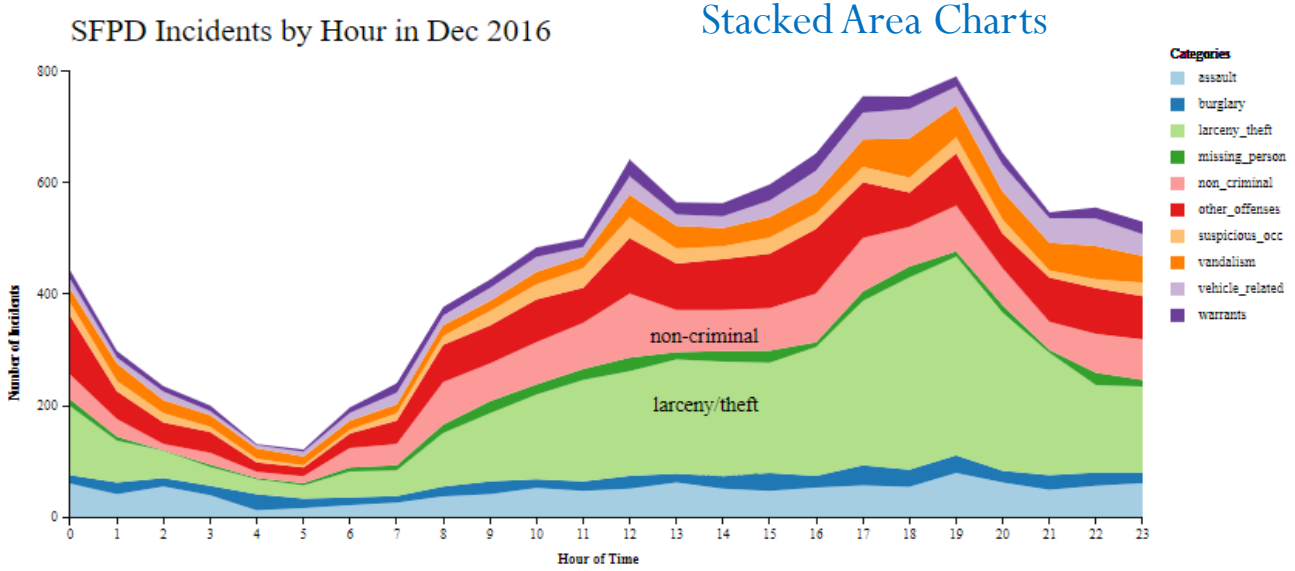
What is D3?

- D3 does not generate predefined visualizations – **no templates** or chart wizards (though there are **tools built on top of D3** that provides access to preconfigured charts e.g., Plotly)
  D3不生成预定义的可视化——没有模板或图表向导（尽管在D3之上构建了一些工具，可以访问预配置的图表，例如Plotly）
- D3 is intended primarily for **highly customized** visualization work - designing one-off explanatory charts or complex, interactive, exploratory tools.
  D3主要用于高度定制的可视化工作-设计一次性解释图表或复杂、交互式、探索性工具。
- Powerful tool for visualization on the web specifically because it enables you to develop whatever you can imagine from **scratch** – edge over tools like Tableau and QlikView.
  网络上可视化的强大工具，特别是因为它使您能够从头开始开发任何你能想象到的东西——Tableau和QlikView等边缘工具。
- D3 code is executed on the **client** side (web browser and not on the web server), data to be visualized **must be sent to the client** – alternative when data cannot be shared – **prerender** visualizations as **static images** and send to browser.
  D3代码在客户端（Web浏览器而不是在Web服务器上）执行，要可视化的数据必须发送到客户端——当数据无法共享时——将可视化预渲染为静态图像并发送到浏览器。

Then again, if data cannot be shared, why provide visualizations?

Visualization Examples

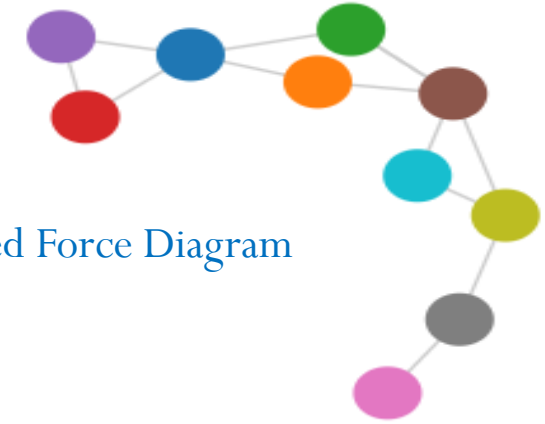
Stacked Area Charts



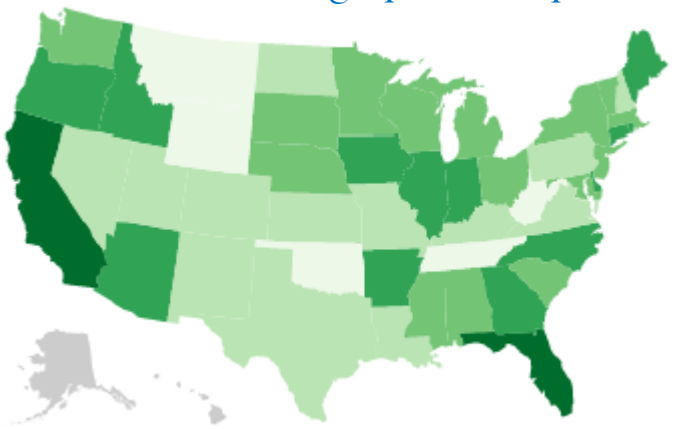
This is a plot of the 10 most frequent incidents over a 24-hour period in San Francisco during December 2016. Interestingly enough, most categories follow a similar structure: the occurences dip in the early morning and peak during the lunch and early evening hours.

By Anaelia Ovalle

Animated Force Diagram



Geographical Maps

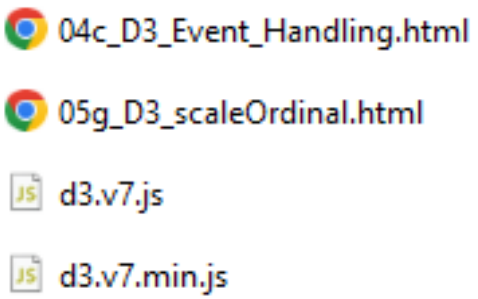


Referencing D3

Go to https://d3js.org/d3.v7.js to download the latest version of D3 – **click save as in the browser**. Use https://d3js.org/d3.v7.min.js for the minified version

Move the file **d3** files into the lab folder.

```
<script src="d3.v7.js"></script>
```



d3.v7.min.js - "minified" version with **whitespace in file removed** for smaller file sizes and faster load times.

Same functionality for both files - use standard version while working on a project (**friendlier debugging**), switch to minified version to launch the project (for **optimized load times**).

Alternatively to **link directly to latest release** use:

https://d3js.org/d3.v7.js

https://d3js.org/d3.v7.min.js

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/d3/7.8.5/d3.min.js"></script>
```

Referencing D3

- Create a simple HTML page named **index.html** within the project folder.

```
project-folder/
    d3.js
    d3.min.js (optional)
    index.html
```

The meta tag **identifies the encoding** for this file as **utf-8** => ensure browser can parse D3's functions and data properly.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>D3 Page Template</title>
        <script type="text/javascript" src="d3.js"></script>
    </head>
    <body>
        <script type="text/javascript">
            // Your beautiful D3 code will go here
        </script>
    </body>
</html>
```

First script tag sets the reference to **d3.js**. Edit file path if located somewhere other than the **project-folder** directory.

Second script tag in the body is where the D3 code will be.

Make a copy of this template for each new project down the line.

Chain Syntax

D3 object

```
d3.select("body")
    .append("p")
    .text("New paragraph!");
```

D3 employs a technique called **chain syntax** allowing **several actions in a single line** of code.

- Reference the D3 **object** to access its **methods**.
  - **.select("body")** - gives the **select() method** a CSS selector ("body" in this case) as input, and it will **return a reference to the first element**. Use **selectAll()** when selecting **more than one** element. In example, **reference to body** is handed off to the **next method (append)** in the chain.
  - **.append("p")** – append() **creates DOM element** specified, in this case "p" as the input argument. The method also **reference "body" passed down** the chain from the select() method => **empty p paragraph is *appended* to the body**.
  - **.text("New paragraph!")** - text() **takes a string and inserts it** between the opening and closing tags. As previous method passed down a reference to p, **new text is inserted between and **.

Drawing SVGs

- **Scalable Vector Graphics** (SVG) is a way to render images on the web page. SVG is not a direct image but a way to create images using text. It **scales itself according to the specified size**, so resizing the browser will not distort the image.
- SVG elements specify all of their properties as **attributes**.

```
<element property="value"></element>
```

<svg height="50" width="500"></svg>

element property value

Creating the SVG element (**the d3 way**)

```
var svgContainer = d3.select("body")
                     .append("svg")
                     .attr("width", 1000)
                     .attr("height", 1000);
```

find the document's body and **append a new svg element** just before the closing </body> tag.

SVG provides different shapes like **lines <line>**, **rectangles <rect>**, **circles <circle>**, **ellipses <ellipse>**, etc.

SVG - Text Element

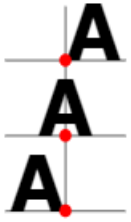
- **Text** is the only element that is **neither a shape nor translates to a path** in the background like the other elements.

```
svgContainer.append('text')
            .text('A picture!')
            .attr("x", 10)
            .attr("y", 150)
            .attr("text-anchor", 'start');
```

A picture!

Append a **text element** to the svg element => then define its **actual text**, add **attributes** to **position** the text at the (x, y) point, and anchor the text at the **start**.

The **text-anchor** attribute defines the **horizontal positioning (start, middle, and end)** of rendered text in relation to the anchor point defined by (x, y).



Start

Middle

End

SVG Shapes – Line

```
svgContainer.append('line')
            .attr("x1", 10)
            .attr("y1", 10)
            .attr("x2", 100)
            .attr("y2", 100)
            .attr("stroke", 'blue')
            .attr("stroke-width", 3);
```

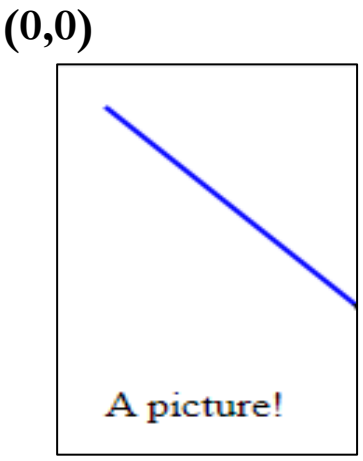
start

end

Take **svg** element, append a **line**, and define following attributes:

Define **start** and **end** points of line: (x1, y1) and (x2, y2).

To make the line visible, define the **stroke color** and **stroke-width** attributes



Line points downwards as y2 is bigger than y1 => the origin lies in the top-left corner => **(x=0, y=0)** defines the **top-left** corner of the image.

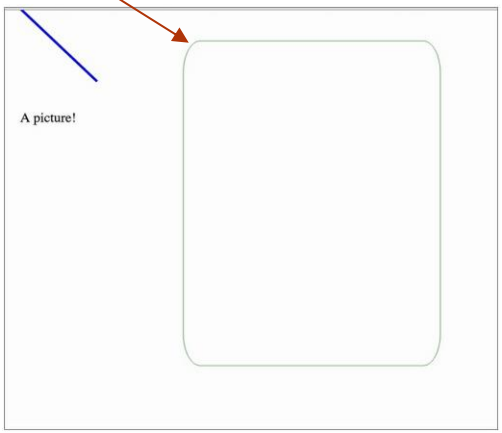
SVG Shapes - Rectangle

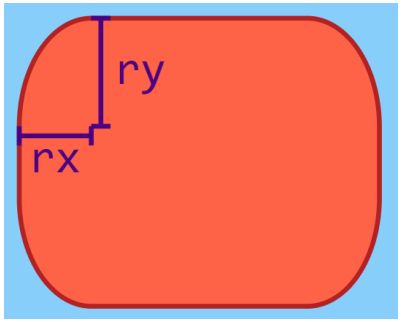
- To draw a rectangle, use the **rect element**:

```
svgContainer.append('rect')
            .attr("x", 200)
            .attr("y", 50)
            .attr("width", 300)
            .attr("height", 400)
            .attr("stroke", "green")
            .attr("stroke-width", 0.5)
            .attr("fill", "white")
            .attr("rx", 20)
            .attr("ry", 40);
```

Append a **rect element** to the **svg element** and define the attributes.

A rectangle is defined by its **top-left corner** ((**x**, **y**)), **width**, and **height**. Define rectangle with 3 more properties (**stroke-width**, **fill**, **rounding radius**)





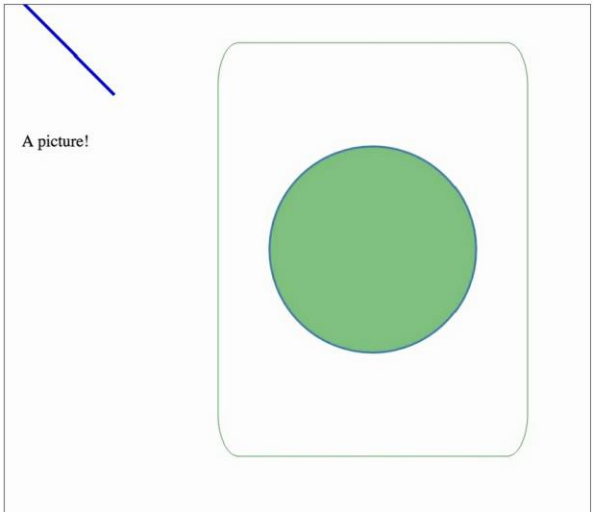
rx defines the x-axis radius used to **round off** the **corners** of the rectangle, **ry** defines the radius at y-axis

SVG Shapes - Circle

```
svgContainer.append('circle')
            .attr("cx", 350)
            .attr("cy", 250)
            .attr("r", 100)
            .attr("fill", "green")
            .attr("fill-opacity", 0.5)
            .attr("stroke", "steelblue")
            .attr("stroke-width", 2);
```

Append a **circle** element to the **svg element** and define the attributes.

A circle is defined by a **central point**, (**cx, cy**), and a **radius**, **r**.



Shows a green circle with a steel blue outline in the middle of the rectangle.

The **fill-opacity** attribute made the circle to be slightly transparent so that it doesn't look too strong against the light rectangle

SVG Shapes - Ellipse

Green ellipse

```
svgContainer.append("ellipse")
            .attr("cx", 350)
            .attr("cy", 250)
            .attr("rx", 150)
            .attr("ry", 70)
            .attr("fill", "green")
            .attr("fill-opacity", 0.3)
            .attr("stroke", "steelblue")
            .attr("stroke-width", 0.7);
```

```
svgContainer.append('ellipse')
            .attr("cx", 350)
            .attr("cy", 250)
            .attr("rx", 20)
            .attr("ry", 70);
```

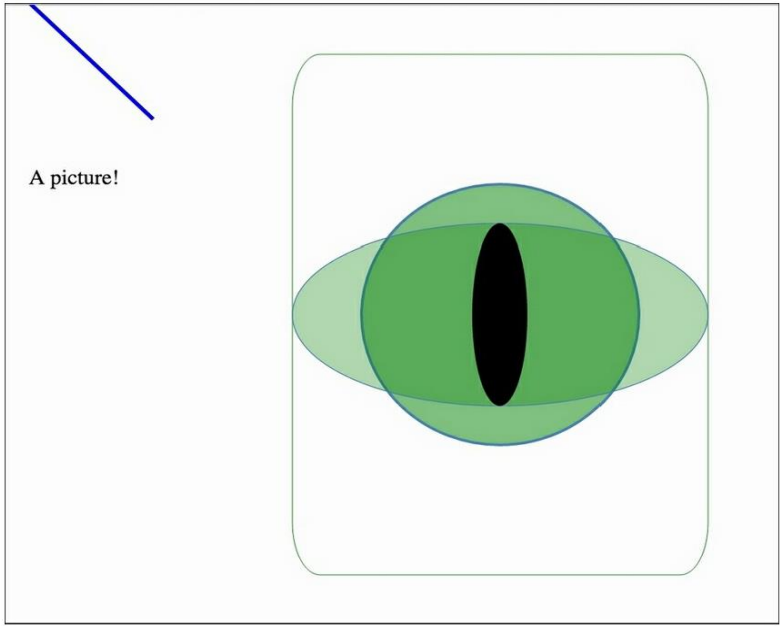
Black ellipse

`rx` is smaller than `ry` => create a **vertical** ellipse.

Append an **ellipse** element to the **svg element** and define the attributes.

The ellipse shape needs a **central point** ((**cx, cy**)) and **two radii** (**rx and ry**).

Setting a low fill-opacity attribute makes the circle visible **under** the ellipse:
(order of placement)

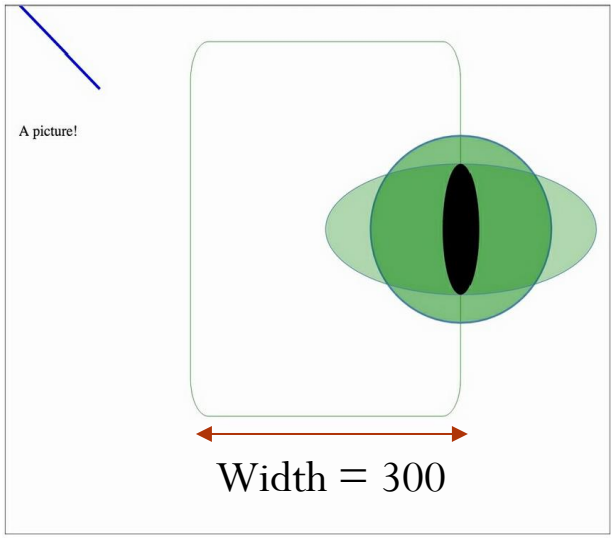


SVG Transformation - Translate

- Affine transformations (geometric transformation that **preserves lines and parallelism**) of coordinate systems used by shapes
- Set of predefined transformations, namely **translate(), path(), scale(), rotate(), skewX(),** and **skewY().**
- Apply transformations with the **transform attribute**.

```
var svgTranslate = svgContainer.selectAll('ellipse, circle')
                               .attr('transform', 'translate(150,0)');
```

Select the **two ellipses** and a **circle** and then apply the **translate** transformation.

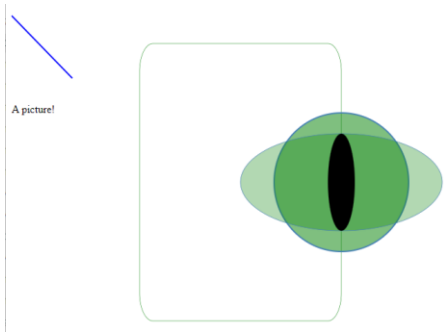


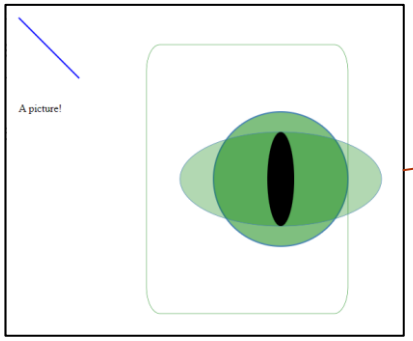
⇒ move the shape's origin **along the (150, 0) vector**, moving the shape 150 pixels to the right and 0 pixels down.

SVG Transformation - Translate

- Applying **new transformations**.

```
var svgTranslate = svgContainer.selectAll('ellipse, circle')
    .attr('transform', 'translate(150,0)');
var svgTranslate = svgContainer.selectAll('ellipse, circle')
    .attr('transform', 'translate(50,0)');
```





Select the **two ellipses** and a **circle** and then apply the **translate** transformation.

⇒ move the shape's origin **along the (150, 0) vector**, moving the shape 150 pixels to the right and 0 pixels down.

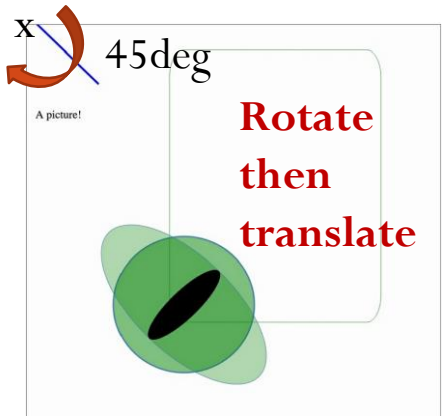
⇒ **moving it again**, new transformations are applied according to the **original state** of the shape => **only one transform attribute per shape**

再次移动它，根据形状的原始状态应用新的变换=>每个形状只有一个变换属性

SVG Transformation - Rotations

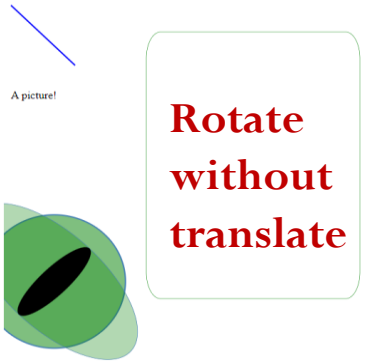
- **Rotate** the eye by 45 degrees:

```
svg.selectAll('ellipse, circle')
  .attr('transform', 'translate(150, 0) rotate(45)');
```



Rotate or translate first? Run code left to right or right to left?

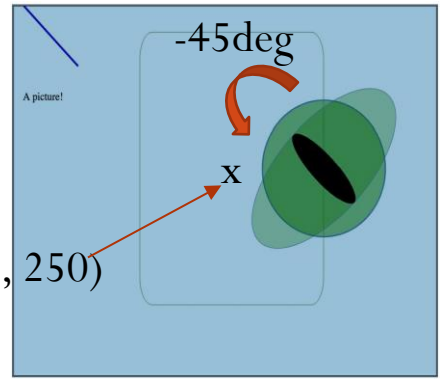
Rotation happened **around the origin** of the **entire image** and not the shape!



To **rotate about the shape** – need to define the **axis of rotation**

```
svg.selectAll('ellipse, circle')
  .attr('transform', 'translate(150, 0) rotate(-45, 350, 250)');
```

Add two more arguments (x, y) to define the rotation axis **around shape –**

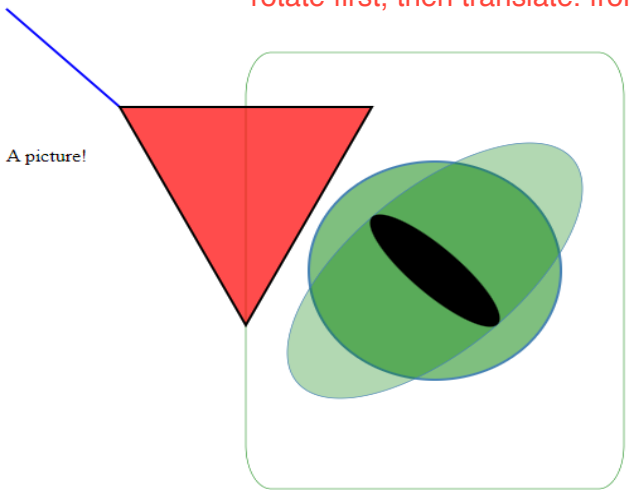


Class Exercise

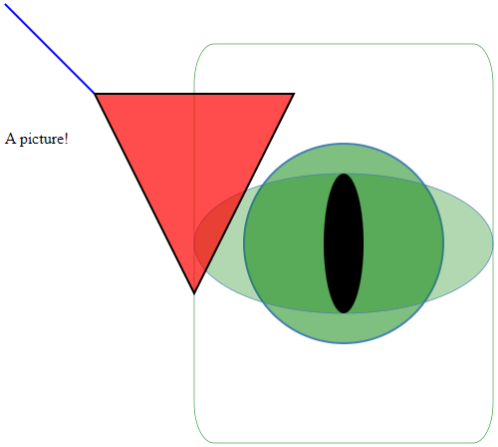
- Open any text editor – e.g., Notepad++
- Open the file 01_SVG.html in the Lab folder.
- Edit the codes to **determine whether codes run from left to right or right to left**

```
svg.selectAll('ellipse, circle')
  .attr('transform', 'translate(150, 0) rotate(45)');
```

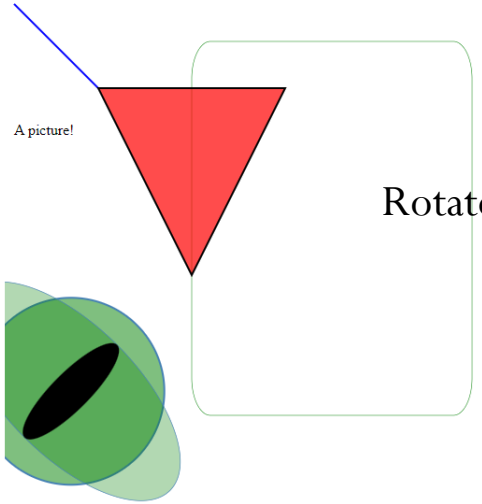
rotate first, then translate. from back to front



Original Location

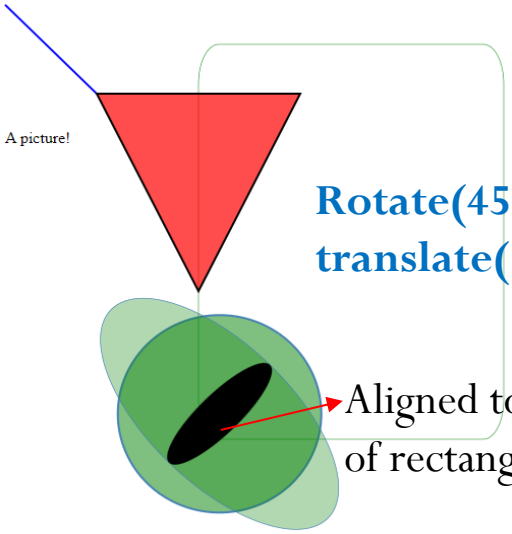



```
var svgTranslate = svgContainer.selectAll('ellipse, circle')
    .attr('transform', 'rotate(45)');
```



Rotate(45)

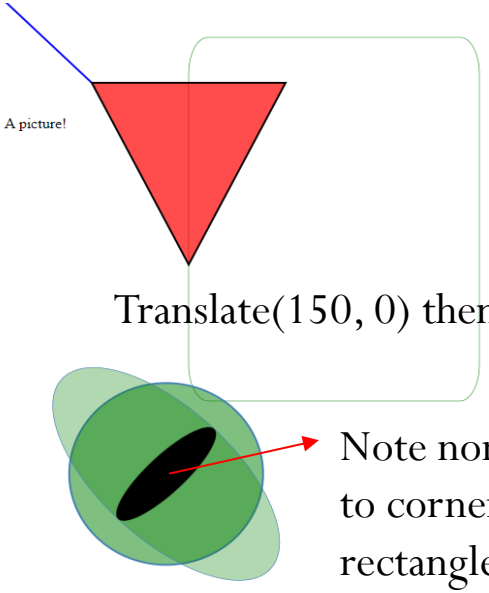
```
var svgTranslate = svgContainer.selectAll('ellipse, circle')
    .attr('transform', 'translate(150,0) rotate(45)');
```



Rotate(45) then translate(150, 0)

Aligned to corner of rectangle

```
var svgTranslate = svgContainer.selectAll('ellipse, circle')
    .attr('transform', 'rotate(45) translate(150,0)');
```



Translate(150, 0) then rotate(45)

Note non-alignment to corner of rectangle

Binding Data

- Data visualization is a process of **mapping data to visuals**. Data in, visual properties out. In D3, data input values are **bind to elements** in the DOM.
  数据可视化是将数据映射到视觉效果的过程。数据进入，视觉属性出。在D3中，数据输入值绑定到DOM中的元素。
- **Binding** - attach or associate data to specific elements – for **referencing** when applying mapping rules.

```
<!doctype html>
<html>
<head>
<script src="https://d3js.org/d3.v4.min.js"></script>
</head>
<body>
<p>Test paragraph</p>
    <script>

        var myData = ["Hello World!"];

        var p = d3.select("body")
                .selectAll("p")
                .data(myData)
                .text(function (d) {
                    return d;
                });

    </script>
</body>
</html>
```

Result:

Hello World!

must array

data array named 'myData' with a single string "Hello World"- to bind to the <p> element.

Data() function **requires a data array**. Will not work on single constant value.

selects the **Body** element.

selects **multiple paragraph** element

data() method - **binds data** array 'myData' to **<p> elements**.

text function adds the **data as text** to each of the **selected ("p") element**. Each **array value** is passed as the first argument **(d)** to **text function**, which replace existing text ('Test paragraph') from paragraph element with the first array value 'Hello World'.

Binding Data – Data Arrays

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>D3: Data Arrays</title>
        <script type="text/javascript" src="d3.js"></script>
    </head>
    <body>
        <script type="text/javascript">

            var dataset = [ 5, 10, 15, 20, 25 ];

            d3.select("body").selectAll("p")
                .data(dataset)
                .append("p")
                .text(function (d) {
                    return d;
                });
        </script>

    </body>
</html>
```

data array named 'dataset' with 5 values.

data() method - **binds** data array 'dataset' to <p> element.

However there are (not enough) no p elements and the text output will not be processed.

Binding Data – Placeholder Element

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>D3: Place Holders</title>
        <script type="text/javascript" src="d3.js"></script>
    </head>
    <body>
        <script type="text/javascript">

            var dataset = [ 5, 10, 15, 20, 25 ];

            d3.select("body").selectAll("p")
                .data(dataset)
                .enter()
                .append("p")
                .text(function (d) {
                    return d;
                });
        </script>
    </body>
</html>
```

5
10
15
20
25

enter() creates a **placeholder element**, if there are more data values than elements

d3.select("body") - **Finds the body** in the DOM and hands off a reference to the next step in the chain.

.selectAll("p") - Selects **all paragraphs** in the DOM. **None exist yet (not defined in this example)**, returns an **empty selection**.

.data(dataset) - **Counts and parses** the five data values in dataset array.

.text("New paragraph!") - reference the **newly created p** and inserts a text value.

Binding Data – Placeholder Element

The **data()** function **maps selection elements to array values**. **What if number of elements and data values don't match?** - cannot use select() and selectAll() - either return lesser elements than dataset, or no selections at all, if element is not defined.

Data（）函数将选择元素映射到数组值。如果元素数量和数据值不匹配怎么办？-不能使用select（）和selectAll（）-如果元素未定义，要么返回比数据集更少的元素，要么根本没有选择。

The **enter()** method dynamically creates **placeholder** references **corresponding to the number of data values.** Output of enter() is fed to **append()** method, which **create** the **elements** when there are **no corresponding** elements on the page.

Enter（）方法动态创建与数据值数量相对应的占位符引用。Enter（）的输出被输入到append（）方法，当页面上没有相应的元素时，append（）方法会创建元素。

What happens if enter() is not used? D3 will simply update the existing nodes. If there are **no existing** nodes **available** to bind the data to, there **won't be any updates**.

```
<body>
    <script type="text/javascript">

        var dataset = [ 5, 10, 15, 20, 25 ];

        d3.select("body").selectAll("p")
            .data(dataset)
            .enter()
            .append("p")
            .text("New paragraph!");

    </script>
</body>
</html>
```

.enter() - **create new, data-bound elements**. If there are more data values than corresponding DOM elements, then enter() **creates a new placeholder element**. It then hands off a reference to this new placeholder to the next step in the chain.

Bound and Determined

Named function

```
var doSomething = function() {
    //Code to do something here
};
```

- How to retrieve the data values?

```
var dataset = [ 5, 10, 15, 20, 25 ];

d3.select("body").selectAll("p")
    .data(dataset)
    .enter()
    .append("p")
    .text(function (d) {
        return d;
    });
```

5
10
15
20
25

Anonymous functions are arguments being passed to **higher-order functions**, or used for constructing the result of a **higher-order function** that needs to **return a function**.

匿名函数是传递给高级函数的参数，或用于构建需要返回函数的高级函数的结果。

```
function(d) {
    return d;
}
```

- Using chain methods after calling data(), create an **anonymous function** that **accepts d (data)** as input to **return d value** to **higher-order (text)** function.
- **data()** and **enter()** methods ensure that **d** is **set** to the **corresponding value** in the original **dataset**, given the **current element** at hand.
- Value of "the current element" **changes over time** as D3 loops through each element. E.g., third loop => dataset[2] => 15

在调用data（）后使用链方法，创建一个匿名函数，该函数接受d（数据）作为输入，将d值返回到更高阶（文本）函数。
data（）和enter（）方法确保d设置为原始数据集中的相应值，给定手头的当前元素。
随着D3循环每个元素，"当前元素"的值会随着时间的推移而变化。例如，第三个循环 => 数据集[2] => 15

Bound and Determined

- **.text(), .attr()**, and other D3 methods can **take a function as an argument**.
- **text()** can take either simply a **static string** of text as an argument:

```
.text("someString")
```

- or the **result of a function**:

```
.text(someFunction())  // Presumably, someFunction() would return a string
```

- or an **anonymous function** itself can be the argument

```
.text(function(d) {
    return d;
})
```

Why use function(d) { … } instead of just d on its own?
E.g., this won't work: .text("I can count up to " + d) - **without being wrapped in an anonymous function, d is undefined and hence error.**

为什么要使用函数(d) { ... }，而不是单独使用d？例如，这行不通：.text（“我可以数到”+ d）-如果没有被包装在匿名函数中，d是未定义的，因此是错误的。

Data-Driven Shapes

```
var dataset = [ 5, 10, 15, 20, 25 ];
var circles = svg.selectAll("circle")
                 .data(dataset)
                 .enter()
                 .append("circle");
```

selectAll() will return empty references to all circles (which don't exist yet)

data() binds (dataset) to the elements (iterate through each data point)

enter() returns a **placeholder** reference to the new element

append() adds a circle to the DOM => but attributes need to be defined.

Defines circles attributes

data index

```
circles.attr("cx", function(d, i) {
            return (i * 50) + 25;
        })
        .attr("cy", h/2)
        .attr("r", function(d) {
            return d;
        });
```

cx position based on index

Match r to corresponding d value

Sets the cx attribute for each circle (cx is the **x-position** value of the center of the circle)

i index acts as a **counter** (0,1,2,3,4) => automatically populated

h = height of svg element

For each circle, the value **d matches the corresponding value in original dataset**

Data-Driven Shapes

i is the **numeric index value** of the current element (**starts at 0**)

```
circles.attr("cx", function(d, i) {
            return (i * 50) + 25;
        })
        .attr("cy", h/2)
        .attr("r", function(d) {
            return d;
        })
        .attr("fill", "yellow")
        .attr("stroke", "orange")
        .attr("stroke-width", function(d) {
            return d/2;
        });
```

```
(0 * 50) + 25   //Returns 25
(1 * 50) + 25   //Returns 75
(2 * 50) + 25   //Returns 125
(3 * 50) + 25   //Returns 175
(4 * 50) + 25   //Returns 225
```

Center **x-position** value of each circle

cy is the **y-position** value of the center of each circle. Setting cy to h/2 => aligns all circles in the vertical center of the image as **h stores the height of the entire SVG element**.

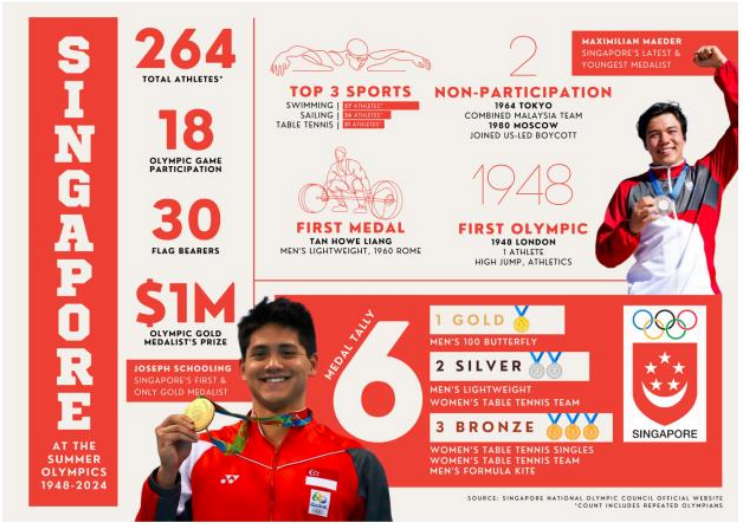
radius r of each circle is simply set to **d**, the corresponding data value.

Set **color fills** and **strokes** using attributes

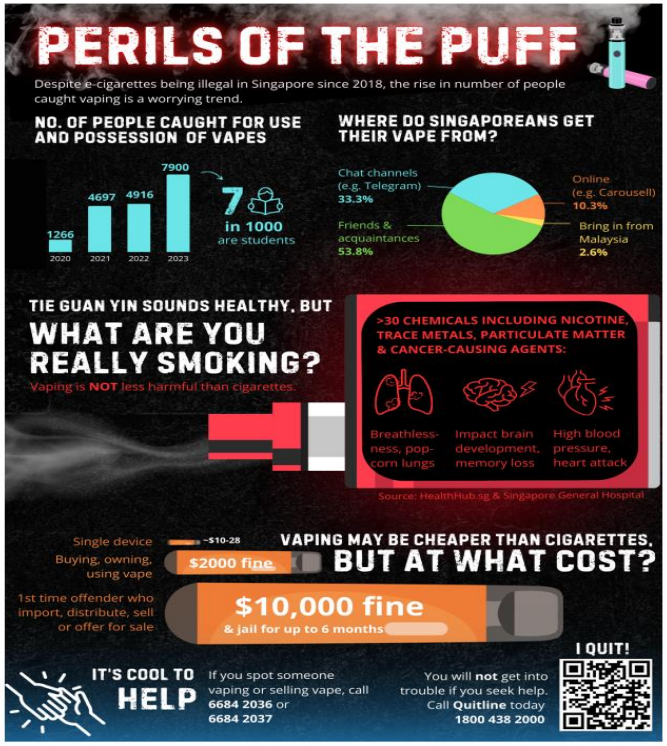
The top and bottom edges of the far-right circle are cut off where they **exceed the boundaries** of the **SVG image**.

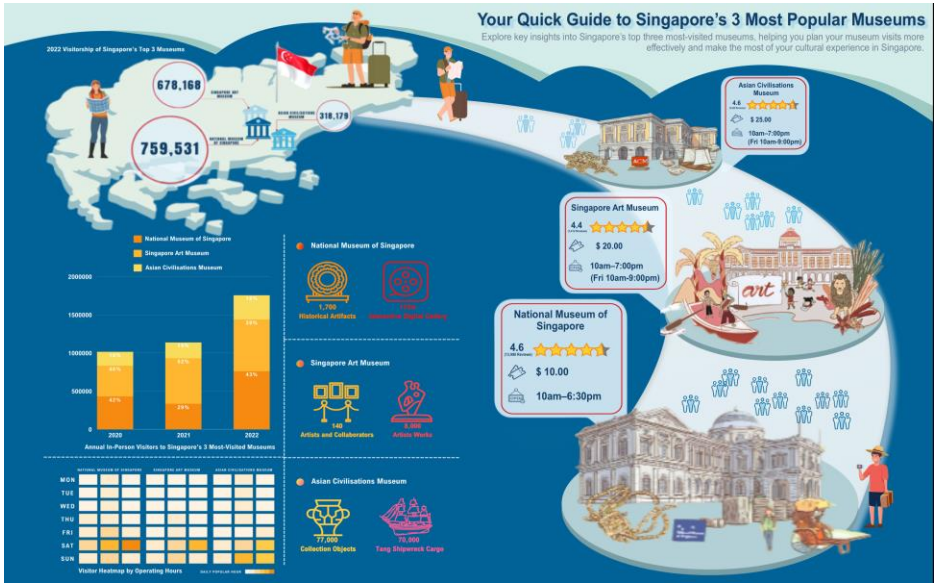
Top 3 voting results (Assignment 1)

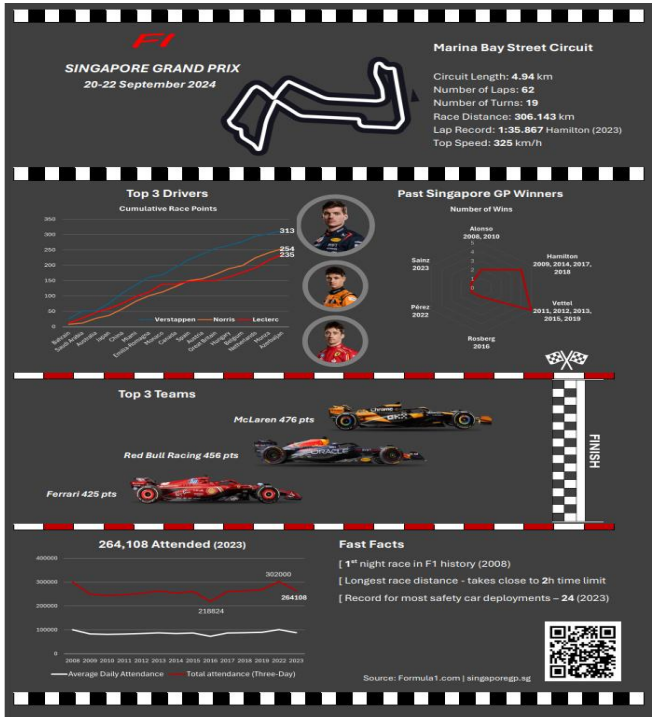
Title: SINGAPORE at the Summer Olympics (1948 – 2024)

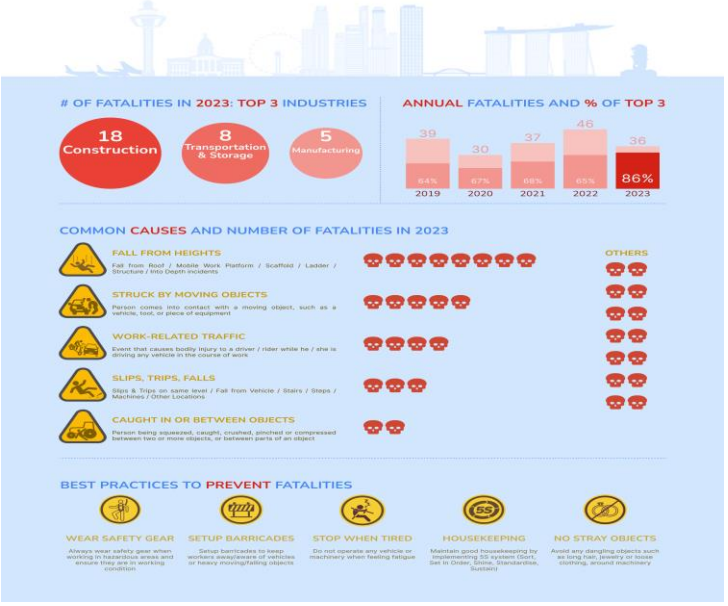


Infographic: Perils of the Puff




Your Quick Guide to Singapore's 3 Most Popular Museums
Explore key insights into Singapore's top three most-visited museums, helping you plan your museum visits more effectively and make the most of your cultural experience in Singapore.
2022 Visitorship of Singapore's Top 3 Museums
678,168
759,531
318,179
National Museum of Singapore
Singapore Art Museum
Asian Civilisations Museum
Annual In-Person Visitors to Singapore's 3 Most-Visited Museums
Visitor Heatmap by Operating Hours
1,700 Historical Artifacts
140 Artists and Collaborators
77,000 Collection Objects
70,000 Tang Shipwreck Cargo
Asian Civilisations Museum
4.6
$ 25.00
10am–7:00pm (Fri 10am-9:00pm)
Singapore Art Museum
4.4
$ 20.00
10am–7:00pm (Fri 10am-9:00pm)
National Museum of Singapore
4.6
$ 10.00
10am–6:30pm


F1
SINGAPORE GRAND PRIX
20-22 September 2024
Marina Bay Street Circuit
Circuit Length: 4.94 km
Number of Laps: 62
Number of Turns: 19
Race Distance: 306.143 km
Lap Record: 1:35.867 Hamilton (2023)
Top Speed: 325 km/h
Top 3 Drivers
Cumulative Race Points
313
254
235
Verstappen
Norris
Lecterc
Past Singapore GP Winners
Number of Wins
Alonso 2008, 2010
Hamilton 2009, 2014, 2017, 2018
Vettel 2011, 2012, 2013, 2015, 2019
Sainz 2023
Pérez 2022
Rosberg 2016
Top 3 Teams
McLaren 476 pts
Red Bull Racing 456 pts
Ferrari 425 pts
FINISH
264,108 Attended (2023)
362000
264108
218824
Average Daily Attendance
Total attendance (Three-Day)
Fast Facts
[1st night race in F1 history (2008)
[Longest race distance - takes close to 2h time limit
[Record for most safety car deployments – 24 (2023)
Source: Formula1.com | singaporegp.sg

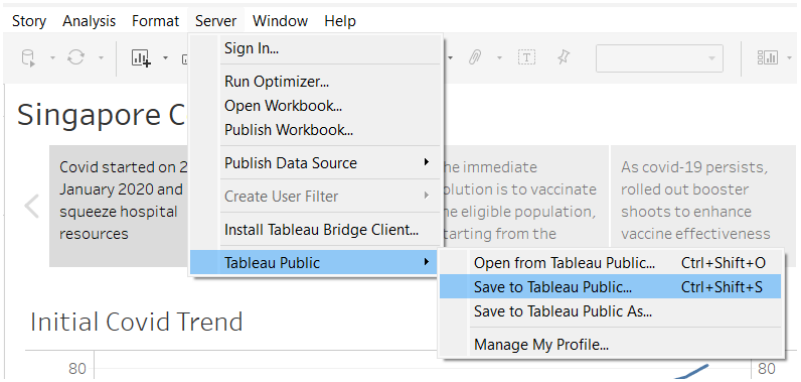

WORKPLACE FATALITY IN SINGAPORE
Devastating accidents in the workplace can happen to anyone at anytime. This infographic aims to instill awareness on workplace safety based on facts reported by the Ministry of Manpower between 2019 to 2023.
OF FATALITIES IN 2023: TOP 3 INDUSTRIES
18 Construction
8 Transportation & Storage
5 Manufacturing
ANNUAL FATALITIES AND % OF TOP 3
39
30
37
46
36
64%
67%
68%
65%
86%
2019
2020
2021
2022
2023
COMMON CAUSES AND NUMBER OF FATALITIES IN 2023
FALL FROM HEIGHTS
Fall from Roof / Mobile Work Platform / Scaffold / Ladder / Structure / Into Depth incidents
STRUCK BY MOVING OBJECTS
Person comes into contact with a moving object, such as a vehicle, tool, or piece of equipment
WORK-RELATED TRAFFIC
Event that causes bodily injury to a driver / rider while he / she is driving any vehicle in the course of work
SLIPS, TRIPS, FALLS
Slips & Trips on same level / Fall from Vehicle / Stairs / Steps / Machines / Other Locations
CAUGHT IN OR BETWEEN OBJECTS
Person being squeezed, caught, crushed, pinched or compressed between two or more objects, or between parts of an object
OTHERS
BEST PRACTICES TO PREVENT FATALITIES
WEAR SAFETY GEAR
Always wear safety gear when working in hazardous areas and ensure they are in working condition
SETUP BARRICADES
Setup barricades to keep workers away/aware of vehicles or heavy moving/falling objects
STOP WHEN TIRED
Do not operate any vehicle or machinery when feeling fatigue
HOUSEKEEPING
Maintain good housekeeping by implementing 5S system (Sort, Set in Order, Shine, Standardise, Sustain)
NO STRAY OBJECTS
Avoid any dangling objects such as long hair, jewelry or loose clothing, around machinery

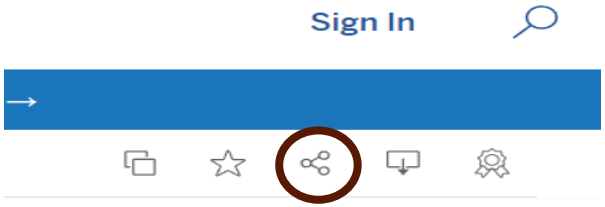
Assignment 2 - Option to publish Tableau Workbook to Tableau Public

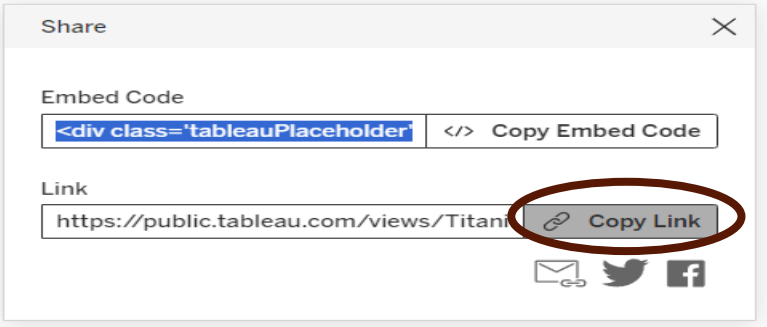
- https://help.tableau.com/current/pro/desktop/en-us/publish_workbooks_tableaupublic.htm

Save a workbook

1. With your workbook open in Tableau Desktop, select **Server** > **Tableau Public** > **Save to Tableau Public**.







Data Loading

- D3 can handle different kinds of data, including **array** ('lists') of numbers, **strings**, or **objects** ('dictionary') (containing other arrays or key/value pairs).
- It can also handle CSV, TSV, JSON, XML, and has built-in methods to load data from these files.

d3.csv() function takes two arguments: a **string** representing the **path of the CSV file**, and an **anonymous function** to be used as a **callback function** - **"called" only *after* the CSV file has been loaded into memory** – so that by the time the callback is called, d3.csv() is **done executing**.

```
<script>
d3.csv("/data/employees.csv", function(data) {
    for (var i = 0; i < data.length; i++) {
        console.log(data[i].Name);
        console.log(data[i].Age);
    }
});
</script>
```

loop based on **data length** (number of rows)

data is loaded into an **array of objects**

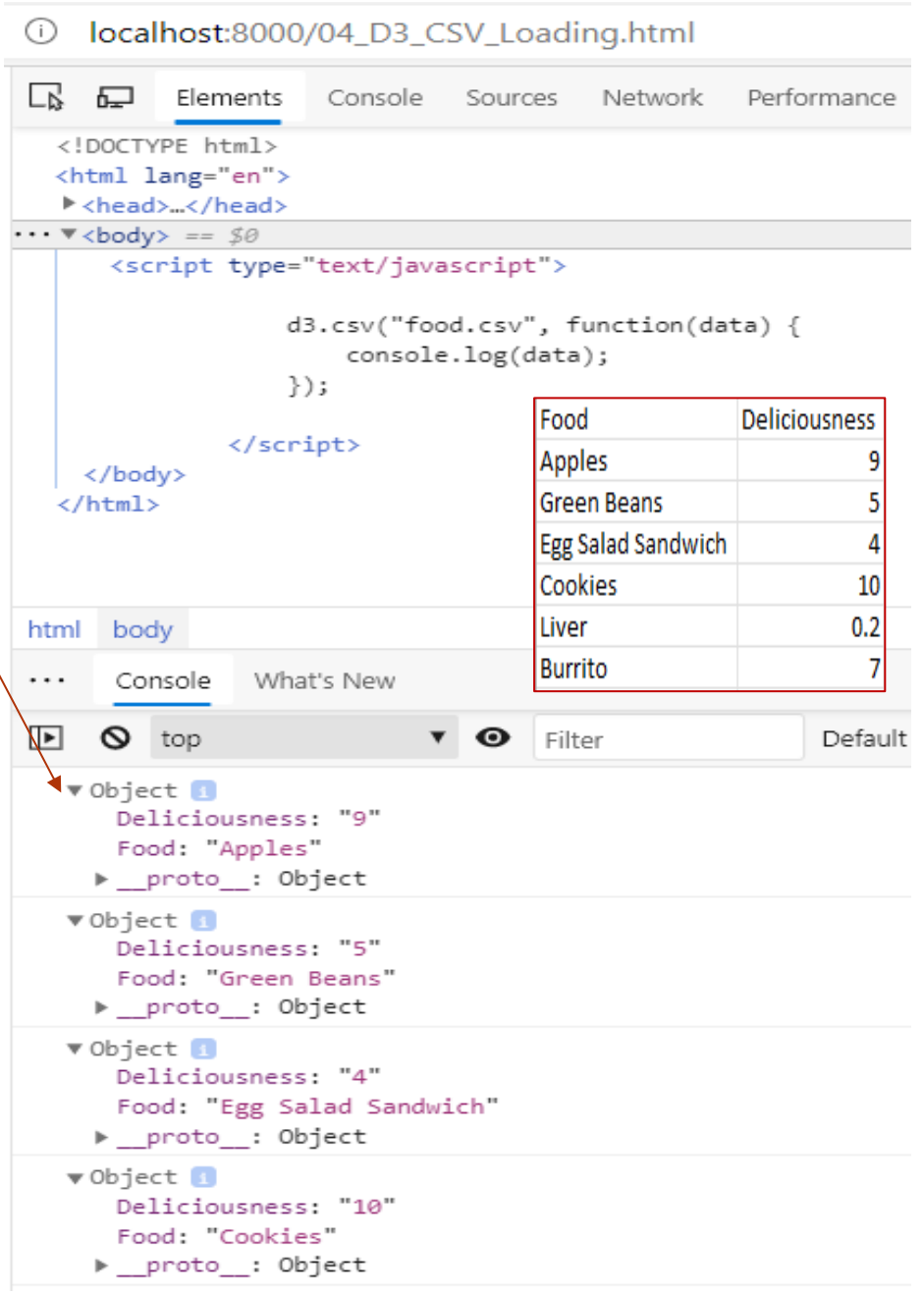
Name	Age
John	30
Jane	32

data object uses **first row column names** as the object's properties and hence are converted to object keys.

Data Loading - CSV

```
var dataset;  //Declare global var

d3.csv("food.csv", function(data) {
  console.log(data);

  //Hand CSV data off to global var,
  //so it's accessible later.
  dataset = data;

  //Call some other functions that
  //generate your visualization, e.g.:
});
```

Print data in console



Food	Deliciousness
Apples	9
Green Beans	5
Egg Salad Sandwich	4
Cookies	10
Liver	0.2
Burrito	7

```
d3.csv("/data/employees.csv", function(data) { })
```

is the same as

```
d3.csv("/data/employees.csv")
  .get(function(data) {
      console.log(data);
  });
```

chain method

Data Loading – JSON (JavaScript Object Notation)

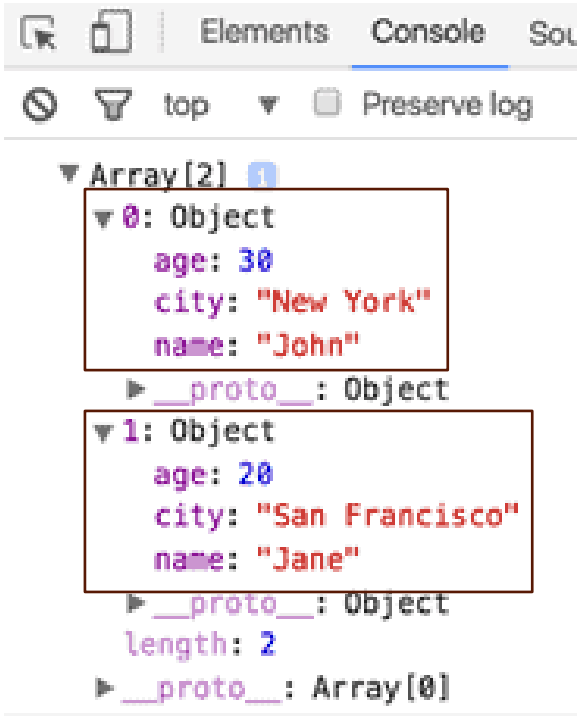
JSON Object

```
var nameObj = {
    "name": "John",
    "age": 30,
    "city": "New York"
};
```

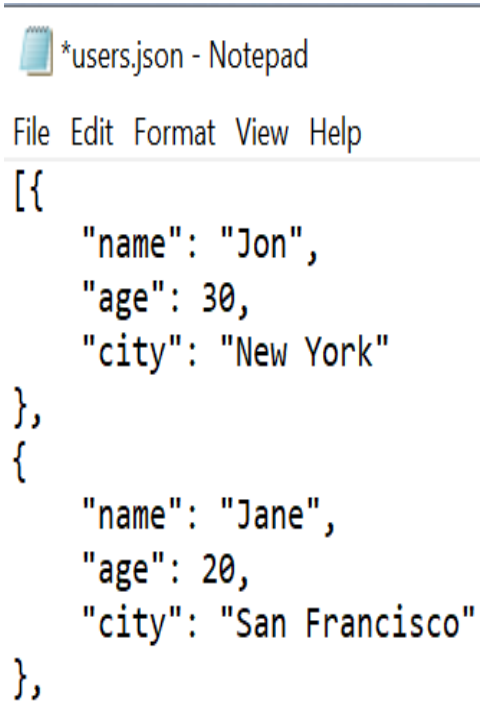
JSON Array of Objects

```
var nameArray = [{
    "name": "John",
    "age": 30,
    "city": "New York"
},
{
    "name": "Jane",
    "age": 20,
    "city": "San Francisco"
}];
```

```
d3.json("/data/users.json", function(data) {
    console.log(data);
});
```



json file contains array (list) of objects



Data Loading – TSV & XML

employees.tsv

Name	Age
John	30
Jane	32

TSV (Tab Separated Values) File Loading

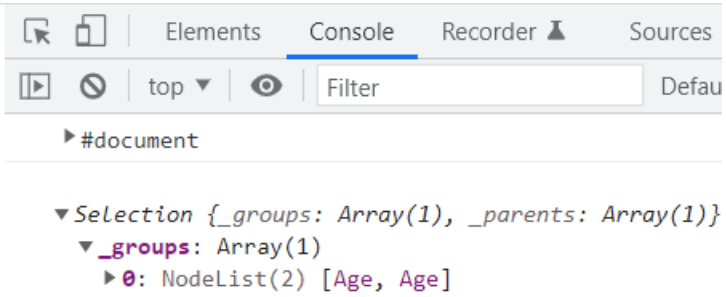
```
<script type="text/javascript">

    d3.tsv("employees.tsv", function(data) {
        console.log(data);
    });

</script>
```

Parse and traverse the XML based on tag name "**Age**"

XML File Loading

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<row>
    <Name>John</Name>
    <Age>30</Age>
</row>
<row>
    <Name>Jane</Name>
    <Age>32</Age>
</row>
</root>
```

```
d3.xml("employees.xml", function(data) {
    console.log(data);

    let xmlData = d3.select(data.documentElement);

    age = xmlData.selectAll("Age");
    console.log(age);
});
```

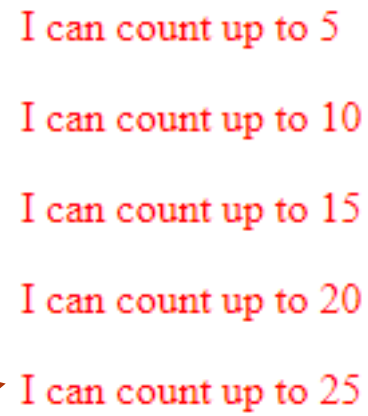


Styling – Style()

- D3's other methods, like **attr()** and **style()**, allow the setting of HTML attributes and CSS properties on selections, respectively.

E.g., adding color to the output:

```
<body>
    <script type="text/javascript">

        var dataset = [ 5, 10, 15, 20, 25 ];

        d3.select("body").selectAll("p")
            .data(dataset)
            .enter()
            .append("p")
            .text(function(d) {
                return "I can count up to " + d;
            })
            .style("color", "red");

    </script>
</body>
```



Styling – Style()

- Use a **custom function** to make the text red only if the current datum exceeds a certain threshold.

仅当当前基准超过一定阈值时，才使用自定义函数使文本变成红色。

```
<body>
    <script type="text/javascript">

        var dataset = [ 5, 10, 15, 20, 25 ];

        d3.select("body").selectAll("p")
            .data(dataset)
            .enter()
            .append("p")
            .text(function(d) {
                return "I can count up to " + d;
            })
            .style("color", function(d) {
                if (d > 15) {   //Threshold of 15
                    return "red";
                } else {
                    return "black";
                }
            });

    </script>
</body>
```

I can count up to 5

I can count up to 10

I can count up to 15

I can count up to 20

I can count up to 25

Styling – Attr()

Another way to style elements is by using **selection.attr** to set the **attribute**.
.attr() is used to change an element's attributes.

selection.attr("attribute string", new value) **circleDemo.attr("r", 40);**

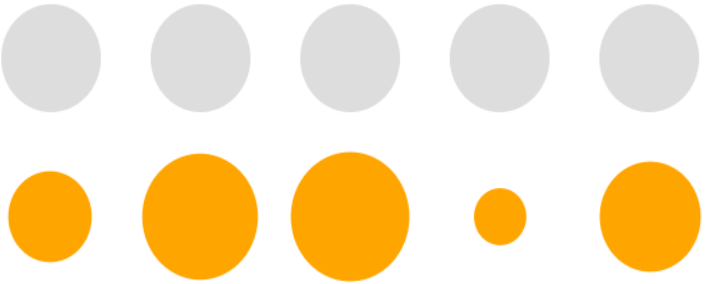
.attr() requires two values: **attribute to be changed**, and **new value** for specified attribute.

```
<circle id = "myCircle" cx="50" cy="50" r="30" ></circle>
var circleDemo = d3.select("#myCircle");
circleDemo.attr("r", 40);
```

selection.attr used to change the attribute

Use function to pass **customized** values to .attr

```
d3.selectAll('circle')
  .style('fill', 'orange')
  .attr('r', function() {
    return 10 + Math.random() * 40;
  });
```



Modifying Elements

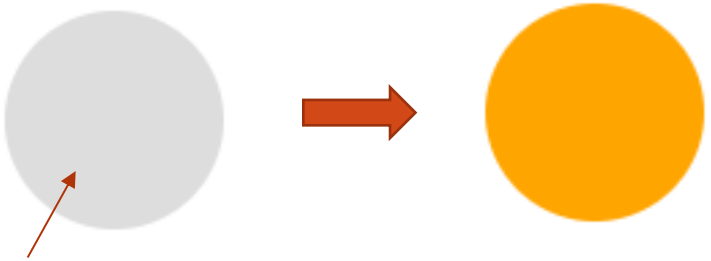
Name	Behaviour	Example
.style	Update the style	d3.selectAll('circle').**style**('fill', 'red')
.attr	Update an attribute	d3.selectAll('rect').**attr**('width', 10)
.classed	Update a class attribute	d3.select('.item').**classed**('selected', true)
.property	Update an element's property	d3.selectAll('checkbox').**property**('checked', false)
.text	Update the text content	d3.select('div.title').**text**('My new book')
.html	Change the html content	d3.select('.legend').**html**(' 0 - 10 ')

Event Handling

Add **event handlers** to selected elements using **.on** which expects a **callback function** with **arguments d** and **i**.

使用.on将事件处理程序添加到选定的元素中，该元素期望具有参数d和i的回调函数。

```
d3.selectAll('circle')
  .on('click', function(d, i) {
    d3.select(this)
      .style('fill', 'orange');
  });
```



Click on the circle => on click fill the circle orange

Event name	Description
click	Element has been clicked
mouseenter	Mouse pointer has **moved onto** the element (must **exit** and **enter** again to retrigger event)
mouseover	Mouse pointer has **moved onto** the element or its **children**
mouseleave	Mouse pointer has **moved off** the element
mouseout	Mouse pointer has **moved off** the element or its **children**
mousemove	Mouse pointer has **moved over** the element (**no need to exit** element to retrigger event)

Class Exercise – D3

- Open the file 04c_D3_Event_Handling.html
- Edit the codes to:
  - Create 5 circles that are placed 120px apart.
  - Create an event upon clicking the circle that will fill the circle with the color orange.

Click on a circle

```
<script>
d3.selectAll('circle')
  .on('click', function(d, i) {
    d3.select(this)
      .style('fill', 'orange');
  });
</script>
```

D3 - Scales

- **Scales** are **functions** that map from an **input domain** to an **output range**.

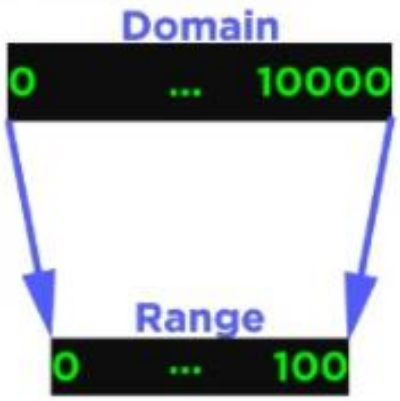
- Values in any dataset are **unlikely to correspond** exactly to pixel measurements in the visualization. Scales provide a convenient way to **map data values to new values** useful for visualization purposes.

任何数据集中的值都不太可能与可视化中的像素测量完全对应。缩放提供了一种将数据值映射到用于可视化目的的新值的便捷方式。

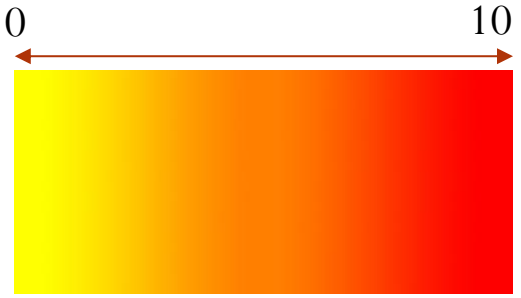
- A scale's **input domain** is the range of **possible input data values**. E.g., given the data, input domain is between 0 and 10000 (the minimum and maximum values of the dataset) and specified as 0 and 100 for the corresponding range in the visual output.

尺度的输入域是可能的输入数据值的范围。例如，给定数据，输入域在0到10000（数据集的最小值和最大值）之间，并为视觉输出中的相应范围指定为0和100。

- A scale's **output range** is the range used as display values in **continuous** or **discrete** values. Output range is arbitrary (by choice) => e.g., [0] => (255,255,0) yellow, [10] => (255,0,0) red




```
var linearScale = d3.scaleLinear()
  .domain([0, 10])
  .range(['yellow', 'red']);

linearScale(0);   // returns "rgb(255, 255, 0)"
linearScale(5);   // returns "rgb(255, 128, 0)"
linearScale(10);  // returns "rgb(255, 0, 0)"
```

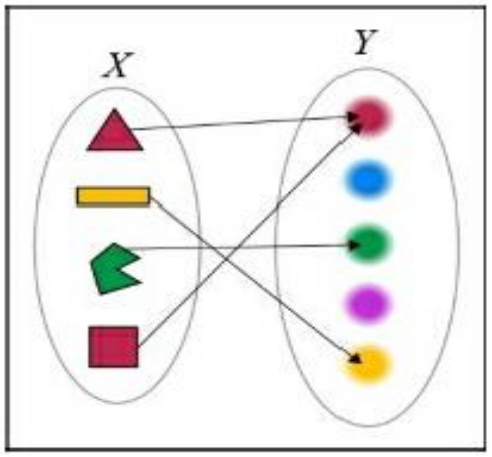


D3 - Scales

- **Scale functions** take an **input domain** (usually a number, date or category) and return a **range value** (such as a coordinate, a colour, a length or a radius) - used to **transform** (or 'map') **data** values into **visual variables** (such as position, length and colour).

缩放函数获取输入域（通常是数字、日期或类别）并返回范围值（如坐标、颜色、长度或半径）-用于将（或"映射"）数据值转换为视觉变量（如位置、长度和颜色）。

```
let scale = d3.scale.ordinal()
      .domain(['triangle', 'line', 'pacman', 'square'])
      .range(['red', 'yellow', 'green', 'red']);
```



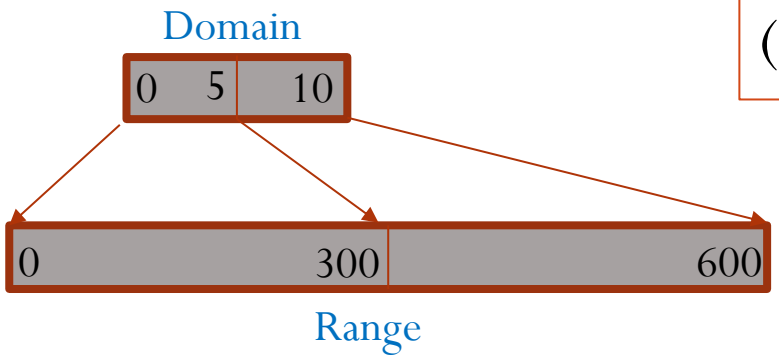
- D3 Scales can be classified into 4 groups:
  - Scales with **continuous** input and **continuous** output
  - Scales with **continuous** input and **discrete** output
  - Scales with **discrete** input and **discrete** output
  - Scales with **discrete** input and **continuous** output

Scales - Continuous Input and Output

- **Linear scales** are commonly used scale type for **transforming data values** into **positions and lengths**. 线性比例尺是将数据值转换为位置和长度的常用比例尺类型。
- **scaleLinear** - use a **linear** function (**y = mx + b**) to interpolate across the domain and range.

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

linearScale(0);   // returns 0
linearScale(5);   // returns 300
linearScale(10);  // returns 600
```

```
var linearScale = d3.scaleLinear()
  .domain([0, 10])
  .range(['yellow', 'red']);

linearScale(0);   // returns "rgb(255, 255, 0)"
linearScale(5);   // returns "rgb(255, 128, 0)"
linearScale(10);  // returns "rgb(255, 0, 0)"
```



$$\left(\frac{\text{Input} - \text{MinDomain}}{\text{MaxDomain} - \text{MinDomain}}\right) * (\text{MaxRange} - \text{MinRange})$$

Scales – Extent() and Nice()

程度

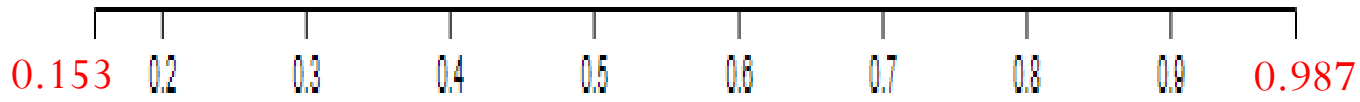
- If the **domain** is computed from real data, the start and end values **might not** be **sorted in order** or in **round figures**.

如果域是根据真实数据计算的，则起始值和终点值可能不会按顺序或圆形数字排序。

```
var data = [0.243, 0.584, 0.987, 0.153, 0.433];
var extent = d3.extent(data);

var linearScale = d3.scaleLinear()
  .domain(extent)
  .range([0, 100]);
```

extent() - returns the **minimum** and **maximum** value in given array.

Output [0.153, 0.987]



- Use **.nice()** to round the **domain** to 'nice' round values:

```
linearScale.nice();
```

Scales – Clamp()

夾具

- By default **scaleLinear**, **scaleSqrt**, **scaleTime** allow input **outside the domain**.

```
var linearScale = d3.scaleLinear()
  .domain([0, 10])
  .range([0, 100]);

linearScale(20);  // returns 200
linearScale(-10); // returns -100
```

Extended **beyond max** domain value 10

Outputs are **beyond range** values - **extrapolated** values outside the domain.

Extended **beyond min** domain value 0

- Use **clamp()** to **restrict** input values inside the **domain**.

```
linearScale.clamp(true);

linearScale(20);  // returns 100
linearScale(-10); // returns 0
```

.clamp(false) – to off clamping

Extended domain values – limited to **max and min range** values respectively

Scales - Continuous Input and Output

- **scaleSqrt** interpolates using a **power** ($y = mx^k + b$) function where **(k=0.5)** => used to size circles by **area** instead of radius.

- As **area** of **circle** is **proportional** to **square** of the **radius** ($A = \pi R^2$), conversely the **radius is proportional to the square root of the area**.

Using scaleSqrt => input **radius** as **domain** give **range** proportional to **area**

```
var sqrtScale = d3.scaleSqrt()
  .domain([0, 100])
  .range([0, 30]);

sqrtScale(0);   // returns 0
sqrtScale(50);  // returns 21.21...
sqrtScale(100); // returns 30
```

$$\sqrt{\left(\frac{50 - \text{MinDomain}}{\text{MaxDomain} - \text{MinDomain}}\right)} * (\text{MaxRange} - \text{MinRange})$$

$$\text{sqrt}(50/100) * 30 = 21.21$$

Scales - Continuous Input and Output

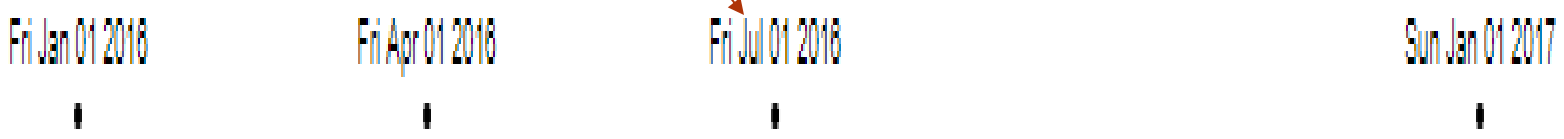
- **scaleTime()** is similar to **scaleLinear()** except the **domain** is expressed as an **array of dates**.

Expressed as array of dates using **Date()** function with arguments as inputs

```
timeScale = d3.scaleTime()
  .domain([new Date(2016, 0, 1), new Date(2017, 0, 1)])
  .range([0, 700]);

timeScale(new Date(2016, 0, 1));   // returns 0
timeScale(new Date(2016, 6, 1));   // returns 348.00...
timeScale(new Date(2017, 0, 1));   // returns 700
```

Month (0 to 11)

Range values determined by length of axis and number of partitions required



Scales - Continuous Input and Discrete Output

- **scaleQuantize()** accepts **continuous** numeric input and outputs a number of **discrete** **quantities** defined by the **range**.

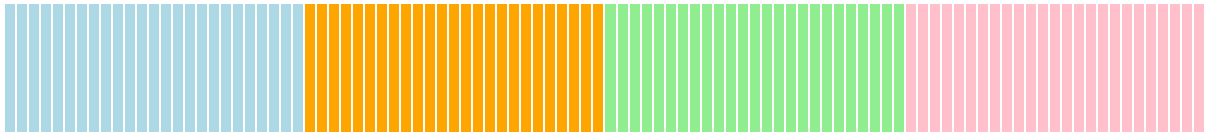
```
var quantizeScale = d3.scaleQuantize()
  .domain([0, 100])
  .range(['lightblue', 'orange', 'lightgreen', 'pink']);

quantizeScale(10);   // returns 'lightblue'
quantizeScale(30);  // returns 'orange'
quantizeScale(90);  // returns 'pink'
```

Continuous input

Discrete output

0 ≤ *u* < 25 is mapped to 'lightblue'
25 ≤ *u* < 50 is mapped to 'orange'
50 ≤ *u* < 75 is mapped to 'lightgreen'
75 ≤ *u* ≤ 100 is mapped to 'pink'

where ***u*** is the **input value**



Scales - Continuous Input and Discrete Output

- **scaleQuantile()** maps **continuous** **numeric** input to **discrete** values. The **domain** is defined by an **array** of numbers.

  In this case an array of values

scaleQuantile（）将连续数字输入映射到离散值。域由数字数组定义。

Array of 15 input values

```
var myData = [0, 5, 7, 10, 20, 30, 35, 40, 60, 62, 65, 70, 80, 90, 100];

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

quantileScale(0);   // returns 'lightblue'
quantileScale(20);  // returns 'lightblue'
quantileScale(30);  // returns 'orange'
quantileScale(65);  // returns 'lightgreen'
```

3 **equal quantiles**
output of 5 values each

the first 5 values are mapped to ‘lightblue’
the next 5 values to ‘orange’ and
the last 5 values to ‘lightgreen’.

3 Groups

- The (sorted) domain array is **divided** into ***n*** intervals of (roughly) **equal frequency**, where ***n*** is the **number** of **range** values.

What happens when #domain values not divisible by n? (5/3)

(Quantile1)
5 domain values

(Quantile2)
5 domain values

(Quantile3)
5 domain values

```
var myData = [0, 5, 7, 9, 10];
```

group into roughly equal frequency

Scales - Continuous Input and Discrete Output

- **scaleThreshold()** maps **continuous numeric** input to **discrete** values **defined by the range**. **n-1 domain split points** are specified where **n** is the **number of range values**.

scaleThreshold（）将连续数字输入映射到范围定义的离散值。指定了n-1域拆分点，其中n是范围值的数量。

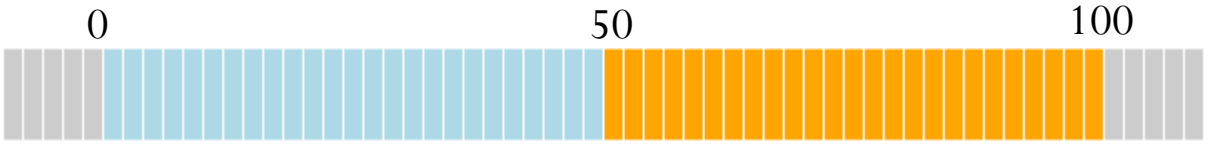
```
var thresholdScale = d3.scaleThreshold()
  .domain([0, 50, 100])
  .range(['#ccc', 'lightblue', 'orange', '#ccc']);

thresholdScale(-10);  // returns '#ccc'
thresholdScale(20);   // returns 'lightblue'
thresholdScale(70);   // returns 'orange'
thresholdScale(110);  // returns '#ccc'
```

4 range values

3 split points of **domain** at **0,50,100** **defined by the 4 range values**

$u < 0$ is mapped to '#ccc'
$0 \le u < 50$ to 'lightblue'
$50 \le u < 100$ to 'orange'
$u \ge 100$ to '#ccc'

u is the input value.



Specifying the **number of range values (n)** allows number of split points to be chosen => if there are **more than n-1 domain values**, **only the first n-1 domain values** will be used as split points

Scales - Discrete Input and Output

- **scaleOrdinal** maps (**ordered**) **discrete** domain values (specified by **array of values**) to **discrete** range values (specified by **array of values**). The **range** array will **repeat** if it's **shorter** than the **domain** array.

scaleOrdinal映射（有序）离散域值（由值数组指定）到离散范围值（由值数组指定）。如果范围数组比域数组短，它将重复。

```
var myData = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

var ordinalScale = d3.scaleOrdinal()
  .domain(myData)
  .range(['black', '#ccc', '#ccc']);

ordinalScale('Jan');  // returns 'black';
ordinalScale('Feb');  // returns '#ccc';
ordinalScale('Mar');  // returns '#ccc';
ordinalScale('Apr');  // returns 'black';
```

Array of values

Range value repeats after one cycle

Ordered Scale

scaleOrdinal

<g> SVG **element** is a container used to group other SVG elements

SVG元素是用于对其他SVG元素进行分组的容器

```
<body>
  <svg width="800" height="60">
    <g id="wrapper" transform="translate(100, 40)">
    </g>
  </svg>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/4.2.2/d3.min.js"></script>
  <script>
var myData = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

var linearScale = d3.scaleLinear()
    .domain([0, 11])
    .range([0, 600]);

var ordinalScale = d3.scaleOrdinal()
    .domain(myData)
    .range(['black', '#ccc', '#ccc']);

d3.select('#wrapper')
    .selectAll('text')
    .data(myData)
    .enter()
    .append('text')
    .attr('x', function(d, i) {
        return linearScale(i);
    })
    .text(function(d) {
        return d;
    })
    .style('fill', function(d) {
        return ordinalScale(d);
    });

  </script>
</body>
```

Declare Linear scale
=> 12 domain values, 600 px range

Declare **ordered** data in array

Declare Ordinal scale → 'black', '#ccc', '#ccc'

x position attribute of text based on **linear** scale **index** setting

add text based on data

style text based on **ordinal** scale setting

Scales - Discrete Input and Continuous Output

- **scaleBand** – determines the **geometry** of the **bars**, including **padding** in **bar charts**. The **domain** is specified as **array** of values (**one value for each band**) and **range** as **minimum** and **maximum extents** of the bands (**total width** of bar chart).

scaleBand – 确定条形图的几何形状，包括条形图中的填充。域被指定为值数组（每个波段一个值），范围被指定为波段的最小和最大范围（条形图的总宽度）。

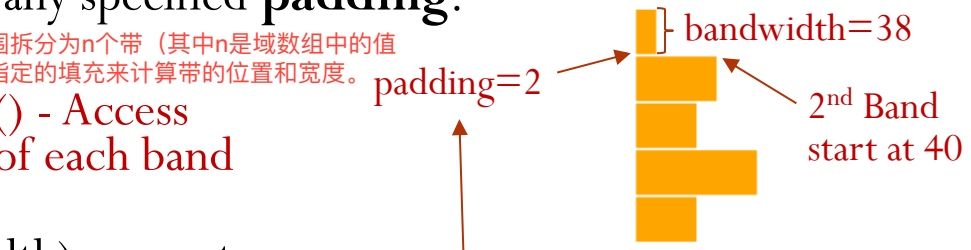
```
var bandScale = d3.scaleBand()
  .domain(['Mon', 'Tue', 'Wed', 'Thu', 'Fri'])
  .range([0, 200]);

bandScale('Mon'); // returns 0
bandScale('Tue'); // returns 40
bandScale('Fri'); // returns 160
bandScale.bandwidth();  // returns 40
```

scaleBand() - **split** the **range** into ***n* bands** (where ***n*** is the **number of values** in the **domain** array) and compute the **positions** and **widths** of the bands taking into account any specified **padding**.

Text

scaleBand（）-将范围拆分为n个带（其中n是域数组中的值数），并考虑到任何指定的填充来计算带的位置和宽度。

.bandwidth() - Access the **width** of each band

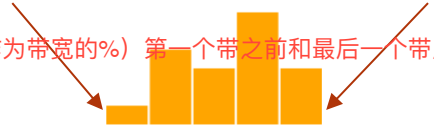


paddingInner - specifies (as % of band width) amount of padding **between each** band.

paddingInner-指定（作为带宽的%）每个带之间的填充量。

paddingOuter - specifies (as % of band width) amount of padding **before first** band and **after last** band.

paddingOuter-指定（作为带宽的%）第一个带之前和最后一个带之后的填充量。



```
bandScale.paddingInner(0.05);→ 0.05*40=2

bandScale.bandWidth();  // returns 38.
bandScale('Mon');       // returns 0
bandScale('Tue');       // returns 40.
```

Scales - Discrete Input and Continuous Output

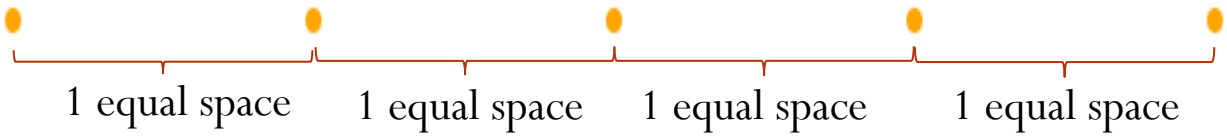
- **scalePoint** – maps from a **discrete** set of values (**n**) to **equally spaced points** along the specified **range**.

```
var pointScale = d3.scalePoint()
  .domain(['Mon', 'Tue', 'Wed', 'Thu', 'Fri'])
  .range([0, 500]);

pointScale('Mon');  // returns 0
pointScale('Tue');  // returns 125
pointScale('Fri');  // returns 500

pointScale.step();  // returns 125
```

Equally spaced

$$\frac{(maxRange - minRange)}{(n-1)} = \text{step length}$$

Distance between the points accessed using **.step()**



(500-0)/(**5-1**) = 125

D3 - Scales ☆

Scale	Input Domain	Output Range	Remarks
scaleLinear	**Continuous**	**Continuous**	Numbers to numbers
scaleSqrt	**Continuous**	**Continuous**	Translate **radius** to area of **circle**
scaleTime	**Continuous** (array of dates)	**Continuous**	Domain contains **date array**
scaleQuantize	**Continuous**	**Discrete** (n range values)	Divide **continuous input** into **equal n intervals**
scaleQuantile	**Continuous** (array of numbers)	**Discrete** (n range values)	Divide **array** into **equal frequency** intervals
scaleThreshold	**Continuous** (array of numbers)	**Discrete** (n range values)	Allow us to **specify** the **cut values** that separate the intervals
scaleOrdinal	**Discrete** (array of values)	**Discrete** (array of values)	**Repeat range array** if shorter than **domain** array of **ordered values**
scaleBand	**Discrete** (array of values)	**Continuous** (min, max extent)	Determines **geometry** of **bar**
scalePoint	**Discrete** (array of values)	**Continuous**	Map domain values to **equally spaced points** with specified **range**

Setting Up Dynamic Scales

Dataset of an array of arrays.

First value (**x-axis**) Second value (**y-axis**)

```
var dataset = [
    [5, 20], [480, 90], [250, 50], [100, 33], [330, 95],
    [410, 12], [475, 44], [25, 67], [85, 21], [220, 88]
];
```

- Map first value in each array onto **x-axis**, second value onto **y-axis.**
- **x values** range from 5 to 480 => instead of specifying input domain as [0, 500] use the functions **d3.min()** and **d3.max()** to analyze the dataset on the fly.

Reference array of values to be evaluated

```
d3.max(dataset, function(d) {
    return d[0];  //References first value in each subarray
});
```

anonymous function hands off each value in the data array, one at a time, as d (position [0]).

d[0] => x value, d[1] => y value

max() function **loops through** each value in the array, and **identifies the largest** (480).

Setting Up Dynamic Scales

Alternatively, could use d3.**min()** to calculate a dynamic value.

References **d[0]**, **x value** of each dataset subarray

```
var xScale = d3.scaleLinear()
                .domain([0, d3.max(dataset, function(d) { return d[0]; })])
                .range([0, w]);
```

Input **data** values →

Output **pixel** values →

Upper end of domain is set to the **maximum** value in dataset

Output range is set to the SVG's **width**.

max() function references **d[1]**, **y value** of each dataset subarray

```
var yScale = d3.scaleLinear()
                .domain([0, d3.max(dataset, function(d) { return d[1]; })])
                .range([0, h]);
```

Upper end of y scale range() is set to **h** instead of w => range value is moving from top to bottom. (.range([**h, 0**]); to move from bottom to top)

Margin Convention

1) Define the **margin**

```
var margin = {top: 20, right: 10, bottom: 20, left: 10};
```

2) Define **width** and **height** as the **inner dimensions** of the chart area.

```
var width = 960 - margin.left - margin.right,
    height = 500 - margin.top - margin.bottom;
```

3) Define svg **g** element (**container used to group other SVG elements**) that translates origin to top-left corner of chart area

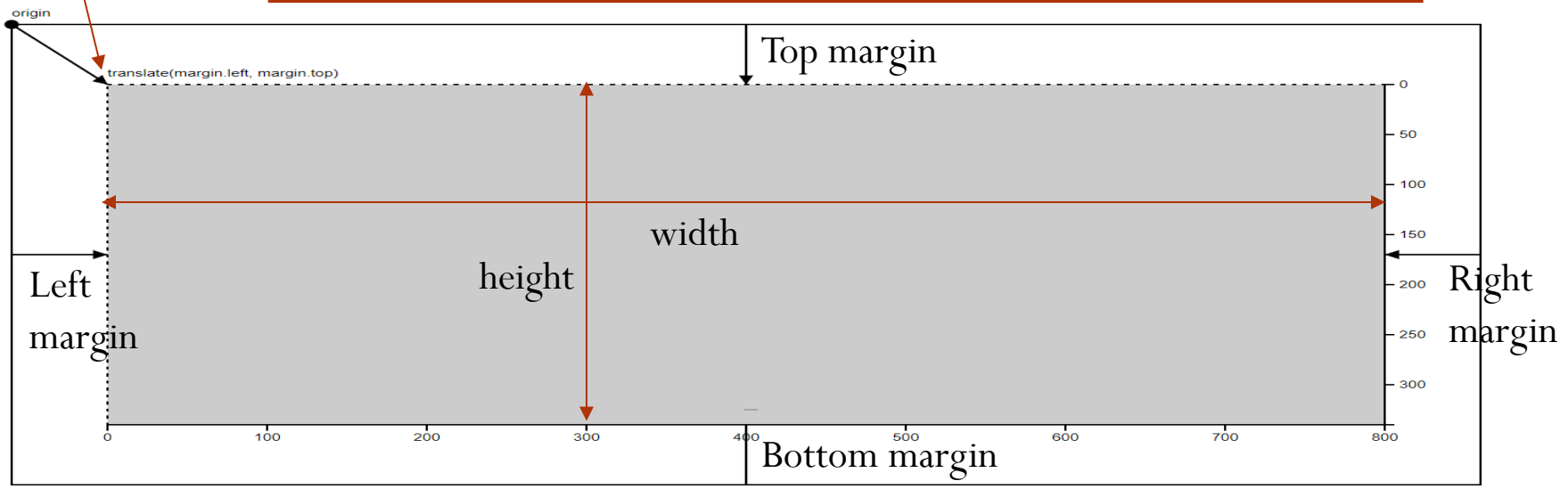
```
var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height)
    .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
```

Define svg

Define width

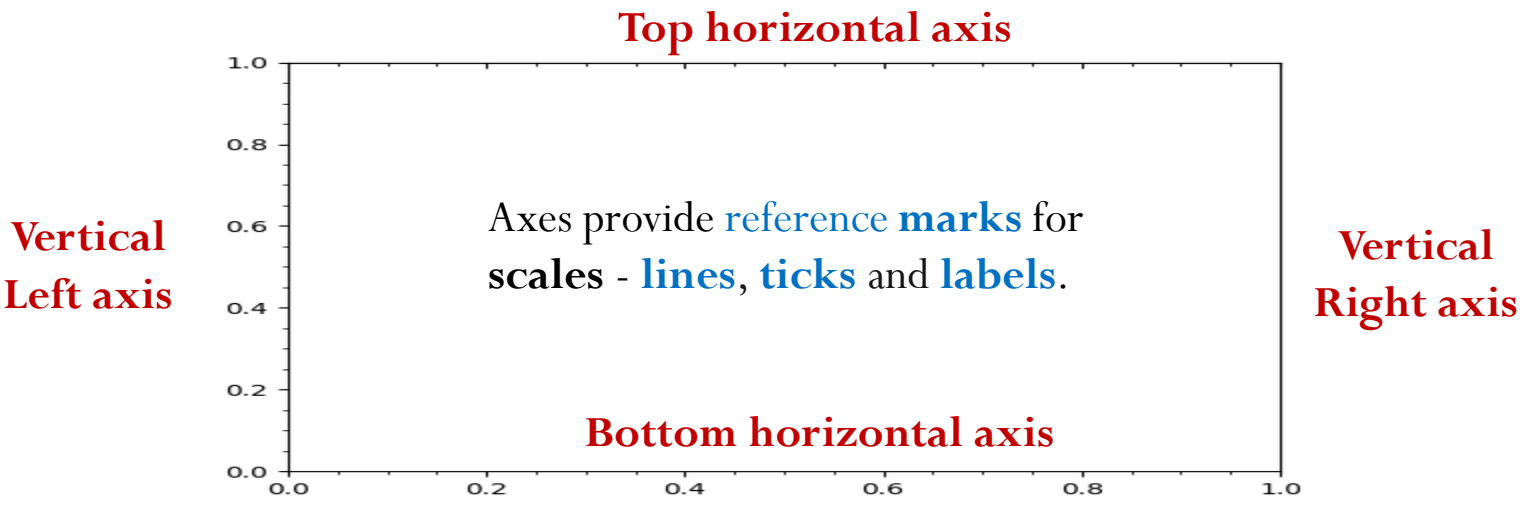
Define height

(relocate the svg)



D3 - Axes

- 2-D Graphs have **two axes**: the horizontal axis (x-axis) and vertical axis (y-axis). An **axis** uses **scale**, so each axis will need to be given a scale to work with.



Axis Methods used to create axes

d3.**axisTop()**	Creates top horizontal axis.
d3.**axisRight()**	Creates vertical right-oriented axis.
d3.**axisBottom()**	Creates bottom horizontal axis.
d3.**axisLeft()**	Creates left vertical axis.

D3 – X and Y Axes

```
<script>
    var width = 400, height = 400;
    var xdata = [10, 15, 20, 25, 30];
    var ydata = [10, 55, 70, 85, 95];

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

    var xscale = d3.scaleLinear()
        .domain([10, d3.max(xdata)])
        .range([0, width - 100]);

    var yscale = d3.scaleLinear()
            .domain([0, d3.max(ydata)])
            .range([height/2, 0]);

    var x_axis = d3.axisBottom()
            .scale(xscale);

    var y_axis = d3.axisLeft()
            .scale(yscale);

        svg.append("g")
           .attr("transform", "translate(50, 10)")
           .call(y_axis);

    var xAxisTranslate = height/2 + 10;

        svg.append("g")
                .attr("transform", "translate(50, " + xAxisTranslate  +")")
                .call(x_axis)
</script>
```

Create SVG element using D3 object

Create **x-axis** **Linear Scale**

(10, 30)

(0, 300)

Margin space 50 on each side

Create **y-axis** **Linear Scale**

(0, 95)

(200, 0)

Create **bottom x-axis** with defined **x scale**

Create **left y-axis** with defined **y scale**

Append **group element** - all components of **y-axis** will be grouped under the group element.

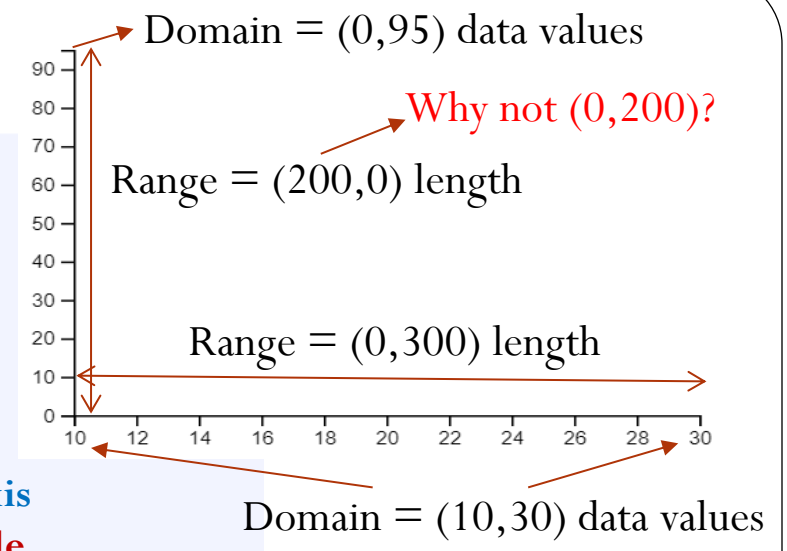
Apply translate transformation to align y-axis to **50px right** and **10px bottom of the origin** – margin gives better visual representation on screen.

Call **y_axis** function => **render y-axis**

210

Move down **to y coordinate of x-axis**

Apply translate transformation to align x-axis with start point of chart (**bottom left** x, y intersection)



Class Exercise - Scales

- The d3.scaleLinear() function generates a scale.

```
var scale = d3.scaleLinear()
            .domain([100, 500])
            .range([10, 350]);
```

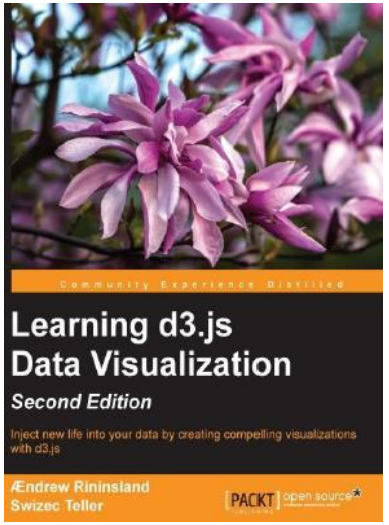
```
scale(100);  => returns? 10
scale(300);  => returns? 180
scale(500);  => returns? 350
```

- Edit the **05g_D3_scaleOrdinal.html** file to make the ordinal scale work.

Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

References

- Rininsland, A. (2016). Learning d3.js Data Visualization 2nd Ed. Packt Publishing.



- Murray, S.(2017) Interactive Data Visualization for the Web, 2nd Ed. O'Reilly.