Lab 1: Intro to HTML, CSS, and SVG

Jump to bottom

Alex Endert edited this page on Jan 11 · 1 revision

Learning Objectives

After completing this lab you will be able to:

- Use web development tools (Sublime and Chrome)
- Set up and modify HTML documents
- Define CSS rules to style a web page
- Create and style SVG element

Prerequisites

- Text Editor or Integrated Developer Environment installed on your laptop (Sublime recommended)
- Web Browser installed on your laptop (Google Chrome recommended)
- Download the corresponding lab from thecode repo (either using git, or downloading the folder) from the code of this repo (in the Code tab above)
- You have read Chapter 3 in D3 - Interactive Data Visualization for the Web by Scott Murray (stop at the Javascript section)

Useful Tutorial Videos

- Chrome Web Inspector
- HTML For Beginners
- CSS For Beginners
- SVG For Beginners

What to submit

1. You should have completed Activity 1, Activity 2, and Activity 3 (in each respective subfolder).
2. Rename your `lab1` folder to `LastName_FirstName_lab1`
3. Zip up `LastName_FirstName_lab1` as `LastName_FirstName_lab1.zip` and submit it to Canvas.

Grading

Your assignment will be graded on the following requirements:

- Created all 4 visuals
- Design of the charts match the description
- Did not use additional visualization libraries other than was is specified
- Your final html pages render according to the description

Tutorial 1: HTML

Let's start with the primary formatting language used to control what appears on webpages, HTML.

HTML is used to structure content for web browsers. It enables us to create a markup by adding additional elements (i.e., tags) to the content. Therefore we can differentiate, for example, between the headline and the body of a story.

A simple HTML document looks like this:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <h1>Page Title</h1>
        <p>This is a really interesting paragraph.</p>
    </body>
</html>
```

The core function of HTML is to enable you to "mark up content" - meaning giving your text, images, and other media structure. HTML is a tool for specifying semantic structure, or attaching hierarchy, relationships, and meaning to content.

Take a description of this course as an example:

```
CS 7450 – Information Visualization Instructor: John Stasko
Fall 2019 Tue, Thu 1:30 – 2:45 pm College of Business Room 300
Computer-based information visualization centers around helping people
explore or explain data through interactive software that exploits
the capabilities of the human perceptual system.
```

The raw text has no structure to it. Unstructured content like this makes it hard to identify titles or read lists.

By adding structure to the content with HTML it becomes easier to quickly glean info from a page:

> **CS 7450 - Information Visualization**

> *Instructor:* **John Stasko**
>
> **Fall 2019**
>
> - Tue
> - Thu
>
> **1:30 - 2:45 pm**
>
> *College of Business Room 300*
>
> Computer-based information visualization centers around helping people explore or explain data through interactive software that exploits the capabilities of the human perceptual system.

We can create this HTML with element tags around our content that "marks-up" the structure and how to present the information. All browsers have a default rendering of HTML. The above example could be given structure by marking it up with the following element tags:

```
<h1>CS 7450 - Information Visualization</h1>
<div>
    <em>Instructor:</em><strong>John Stasko</strong>
</div>
<div>
    <strong>Fall 2019</strong>
    <ul>
        <li>Tue</li>
        <li>Thu</li>
    </ul>
    <strong>1:30 - 2:45 pm</strong>
</div>
<div>
    <em>College of Business Room 300</em>
    <p>Computer-based information visualization centers around helping
    people explore or explain data through interactive software that
    exploits the capabilities of the human perceptual system.</p>
</div>
```

Attributes

All HTML elements can be assigned attributes by including property/value pairs in the opening tag.

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

Different kinds of elements can be assigned different attributes. For example, the a link tag can be given an href attribute, whose value specifies the URL for that link. (href is short for “HTTP reference.”)

```
<a href="http://d3js.org/">The D3 website</a>
```

Classes and IDs

Some attributes can be assigned to any type of element, such as class and id. Classes and IDs are useful when you want to manipulate the elements in page (e.g., changing the styles of specific elements). Consider these HTML elements:

```
<p>CS 7450</p>
<p>Information Visualization</p>
<div>Prof. John Stasko</div>
```

As there are multiple `p` elements, you will not be able to refer only to the second `p` element. To solve this issue, you can give the `p` element an `id`. `id` is like the unique name of an element. With the `id`, you can now change the style of the second `p` element by referring to its name.

```
<p>CS 7450</p>
<p id="info-vis">Information Visualization</p>
<div>Prof. John Stasko</div>
```

HTML also allows you to reference an arbitrary set of elements. This can be done with a `class` attribute. Let's say you want to reference the first and the third element. To do so, you can give the two elements the same class name.

```
<p class="some-info">CS 7450</p>
<p id="info-vis">Information Visualization</p>
<div class="some-info">Prof. John Stasko</div>
```

At a high level, `id` and `class` are for naming HTML elements so that you can specify the elements you want to manipulate. As a general rule, if there will be only one such element on the page, you can use an id. Otherwise, use a class.

Activity 1: Editing HTML

In this activity, you will edit an HTML file that quantifies your web Technology skills.

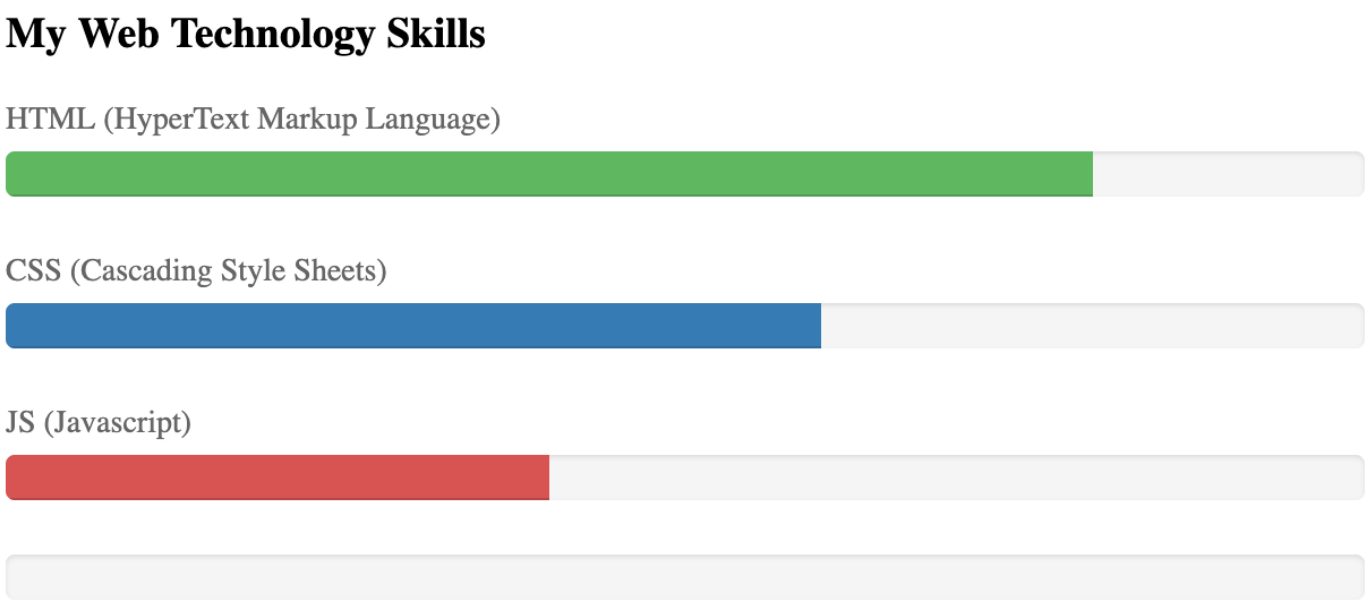
Open `\lab1\activity_1\index.html` in your code editor (e.g. Sublime).

Add a new `div` element within the main `div` that corresponds to your d3 skills.

```
<div id="main">
    ...
    <div class="skill">
```

```
        <p></p>
        <div class="bar-container">
            <div class="bar" style="width: 0%;"></div>
        </div>
    </div>
</div>
```

Save the file and open `\lab1\activity_1\index.html` in your browser. You should see a new empty bar.



We are going to add content to this to bar to quantify your knowledge/skills in d3.js. First we need to add a label to the bar. Edit the `p` element:

```
<p>D3.js (Data-Driven Documents)</p>
```

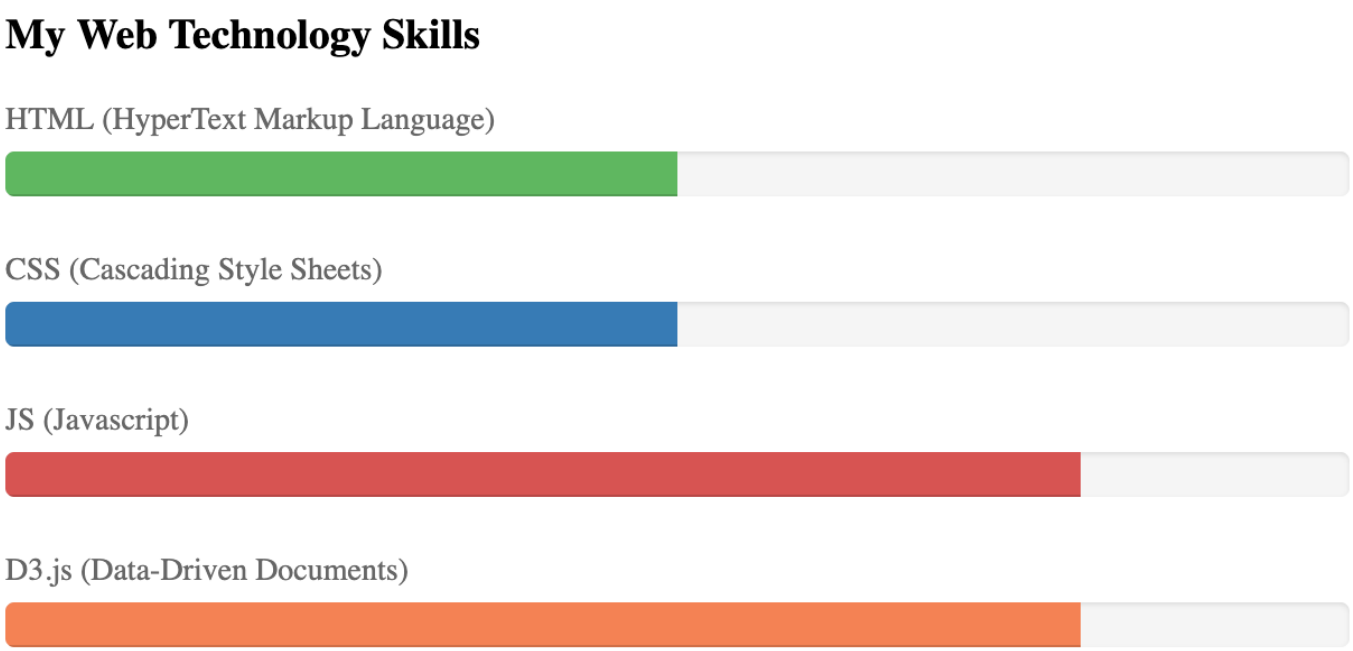
Make sure the bar is properly styled by giving it a unique `id`. Add the `id` attribute to the d3.js `div` you recently added:

```
<div id="skill-d3js" class="skill">
```

Change the `width` of the bar by changing the `width` style attribute. In this case we are using a `percentage` to define the width of this `div` element, however we can also use pixels `100px`.

```
<div class="bar" style="width: 45%;"></div>
```

Finally, you can edit the `width` style of the bars to reflect your own knowledge of web technologies. If your bars are on the lower side, don't worry - we all started out not knowing this stuff either! Was your new bar orange? If not, try to look at the css file. Your HTML page should look like this:



Tutorial 2: CSS

CSS Basics

With HTML, you define the structure and content of the page and with CSS, you set its style - things like fonts, colors, margins, backgrounds etc.

CSS styles consist of selectors and properties, like the following:

```
selector {
    property: value;
    property: value;
    property: value;
}
```

The same properties can be applied to multiple selectors at once by separating the selectors with a comma, as in the following:

```
selectorA,
selectorB,
selectorC {
    property: value;
    property: value;
    property: value;
}
```

For example, you might want to specify that both `p` and `li` elements should use the same font size, line height, and color.

```
p, li {
    font-size: 10px;
    font-weight: bold;
    color: black;
}
```

Selectors

Selectors identify specific elements to which styles will be applied. There are several different kinds of selectors.

Type selectors match elements with the same type:

```
h1          /* Selects all level 1 headings           */
p           /* Selects all paragraphs                 */
```

Descendant selectors match elements that are contained by (or "descended from") another element:

```
h1 em       /* Selects em elements contained in an h1 */
div p       /* Selects p elements contained in a div  */
```

Class selectors match elements of any type that have been assigned a specific class. Class names are preceded with a period, as shown here:

```
.bar        /* Selects elements with class "bar" */
.letter     /* Selects elements with class "letter" */
```

ID selectors match elements with a specific ID. IDs are preceded with a pound sign:

```
#vowel          /* Selects elements with the ID "vowel" */
#high-frequency  /* Selects elements with the ID "high-frequency" */
```

Properties and Values

Groups of property/value pairs cumulatively form the styles:

```
margin: 10px;
padding: 25px;
background-color: yellow;
```

```
    color: pink;
    font-family: Helvetica, Arial, sans-serif;
```

At the risk of stating the obvious, notice that each property expects a different kind of information. color wants a color, margin requires a measurement (here in px or pixels), and so on. You can find exhaustive lists of CSS properties online.

Referencing an external stylesheet from the HTML

A stylesheet are often stored in an external CSS file. Inside the CSS file are a list of CSS rules. To apply the CSS rules to your HTML elements, you need to include a line of code inside the

elements of your HTML file. For example, if your stylesheet is named `style.css`, you will need to include the following line inside .

```
<link rel="stylesheet" href="style.css">
```

Inline Styles

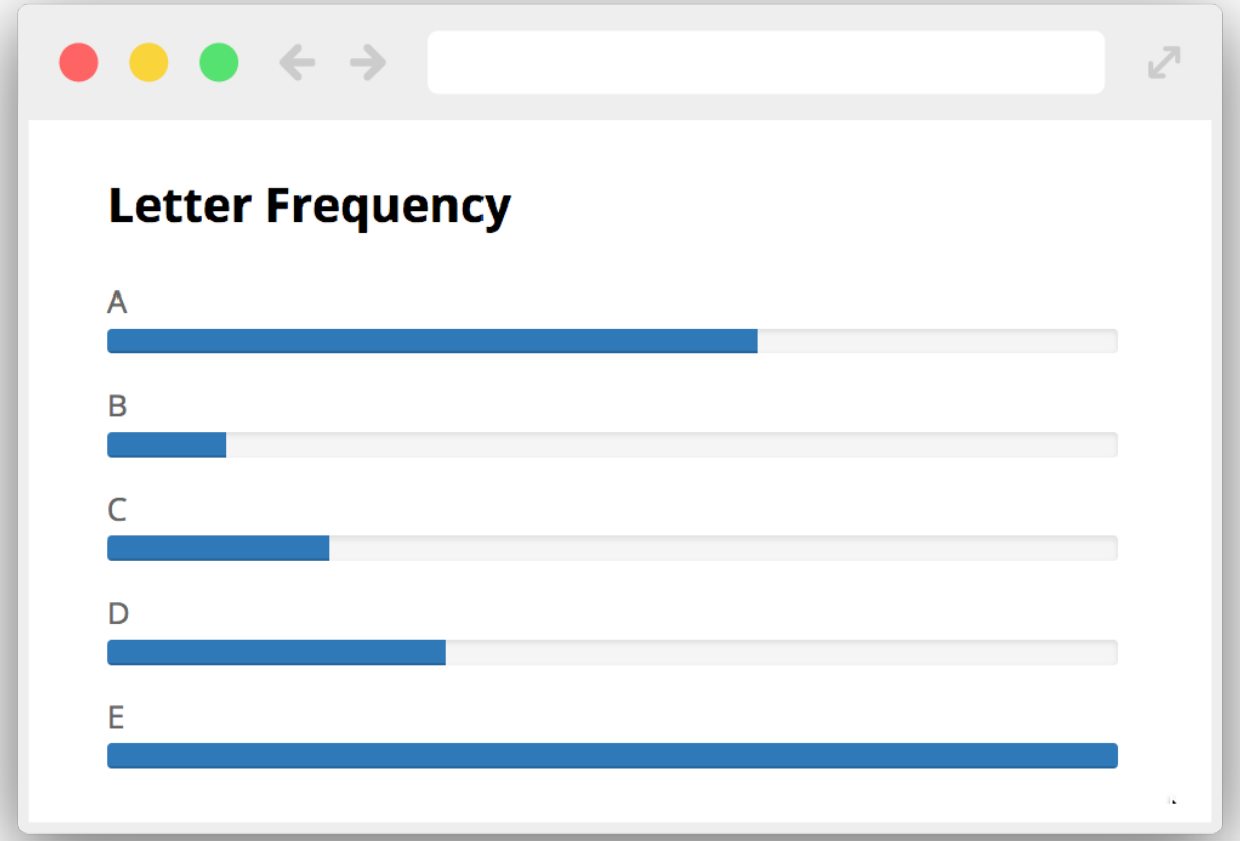
Beside using an external CSS file, you can directly apply styles to HTML elements using inline styles. To attach style rules inline directly to HTML elements, you can add a style attribute to any element. For example, the following `div` element have a background color of blue and an italic font style:

```
<div style="background-color: blue; font-style: italic;">
    Inline styles are kind of a hassle
</div>
```

Activity 2: Styling HTML Elements Using CSS

In this activity, you will use CSS rules to style a bar chart.

Open `\lab1\activity_2\index.html` in your browser. You should see a page with a bar chart representing letter frequency in the English language:

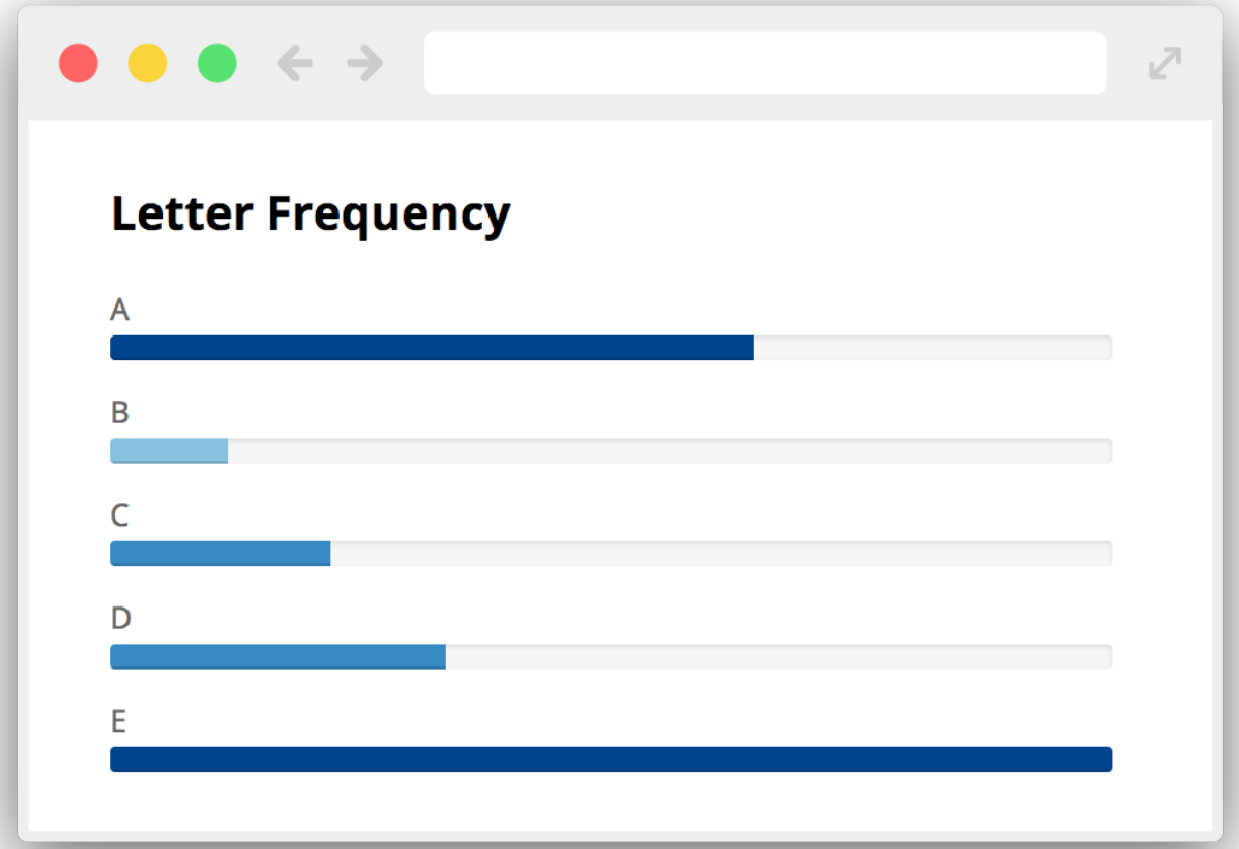


Now, open `\lab1\activity_2\index.html` in your code editor (e.g. Sublime). Each `div` element represents a bar in the chart and all of them have the class `letter`. The `letter` elements in the page include a class grouping them into high, mid, and low frequency (e.g., the class `high-frequency` indicates that a letter appear frequently).

Now, open `\lab1\activity_2\style.css` in your code editor. Add three new CSS rules to style the color of the bars based on frequency groups:

```
.letter.high-frequency .bar {
    background-color: #02458e;
}

.letter.mid-frequency .bar {
    background-color: #378ac4;
}

.letter.low-frequency .bar {
    background-color: #8ac0df;
}
```

Save the file and re-load your browser. You should see the following chart:



As a challenge, make changes to the `style.css` file to change the *Letter Frequency* bars. Try to make the following changes:

- Increase the `margin` between letter `divs`
- Re-style the `p` elements font
- Re-layout the letter `divs` into a 2x13 column and row layout

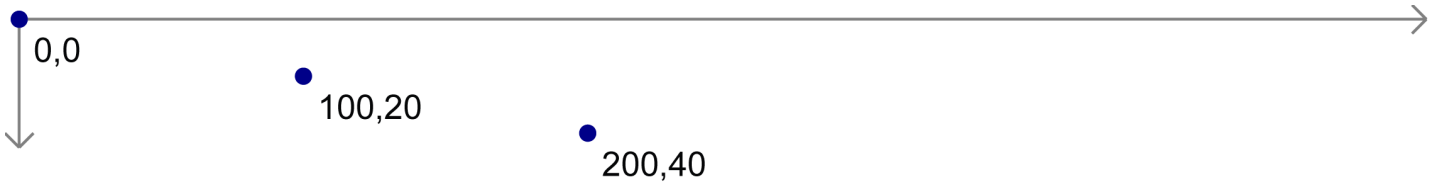
Tutorial 3: SVG

SVG is a subset of the HTML5 standard that will provide us with essentially all of our graphical needs. For most of the time, your visualizations are composed of SVG elements that are rendered using D3. You will learn about D3 in future tutorials, but for now let's learn the ins & outs of drawing with SVG.

The SVG Canvas

The canvas is the space or area where the SVG content is drawn. Conceptually, this canvas is infinite in both horizontal and vertical dimensions. The canvas can therefore be of any size. However, it is rendered on the screen as a finite region. SVG elements that lie beyond the canvas are clipped off and not visible.

In the SVG canvas, the origin (0, 0) is at the top left corner. The positive x-axis points towards the right and the positive y-axis points down. One unit in the coordinate system equals one "pixel". For example, the points (0, 0) and (0, 1) are one pixel apart.



An SVG drawing starts with an `svg` element, which requires width and height attributes:

```
<svg width="300" height="200">
</svg>
```

> Note that pixels are the default measurement units, so we can specify dimensions of `300 and 200`, not `300px and 200px`. We could have specified px explicitly, or any number of other supported units, including *em, pt, in, cm, and mm*

This results in a blank canvas, which is kind of boring. Notice though that we have to specify a height and width attribute for an `<svg>` element, it will not grow to fit the size of its enclosed content. Note: `<svg>` elements default to a size of `300px by 150px` but don't rely on this default.

Next, you'll learn how to add basic graphical shapes to the SVG canvas.

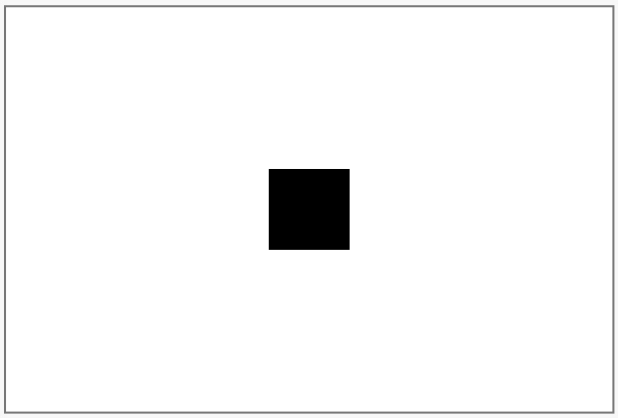
There are a number of visual elements that you can include between those svg tags, including *rect, circle, ellipse, line, path, and text*.

Rectangle

The x and y coordinates of a rectangle specify the position of its top left corner.

```
<svg width="300" height="200">
        <rect height="40" width="40" x="130" y="80"/>
```

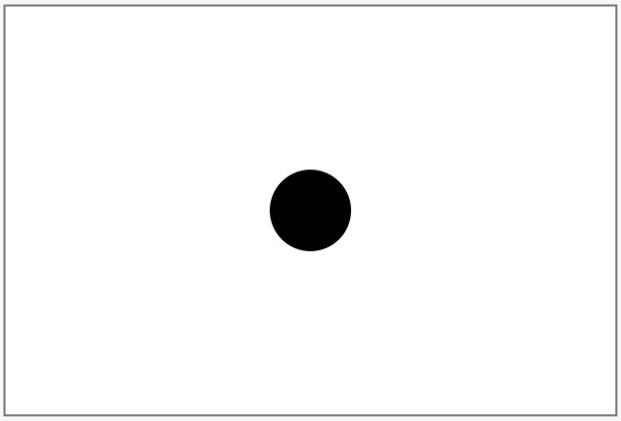
```
</svg>
```



Circle

The x and y coordinates of a circle specify the position of the center.

```
<svg width="300" height="200">
        <circle r="20" cx="150" cy="100"/>
</svg>
```



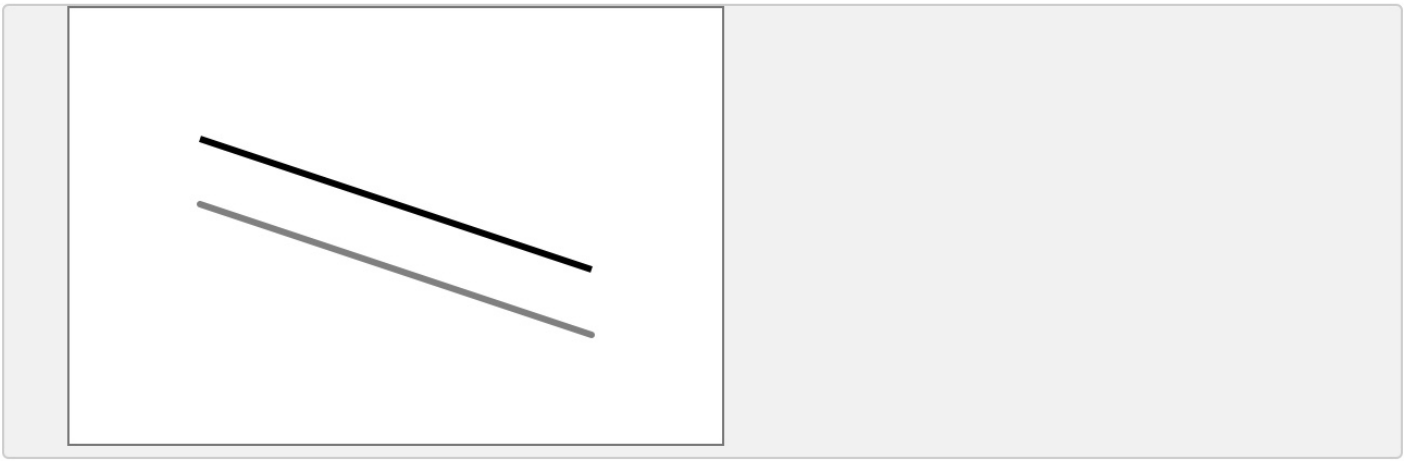
Ellipse

Common SVG properties are `fill`, `stroke`, `stroke-width`, and `opacity`. By default, SVG elements have a style of black fill with no stroke. The following shows you how to create a red ellipse with a gray stroke and a stroke width of 3px. The `rx` and `ry` attributes define the horizontal and vertical radii respectively

```
<svg width="300" height="200">
        <ellipse rx="40" ry="30" cx="150" cy="100"
                        style="fill: red; stroke: gray; stroke-width: 3px"/>
</svg>
```

Line

```
<svg width="300" height="200">
        <line x1="60" y1="60" x2="240" y2="120"
                  style="stroke: black; fill: none; stroke-width: 3px;"/>
        <line x1="60" y1="90" x2="240" y2="150"
                  style="stroke: gray; fill: none; stroke-width: 3px; stroke-linecap:
round;"/>
</svg>
```

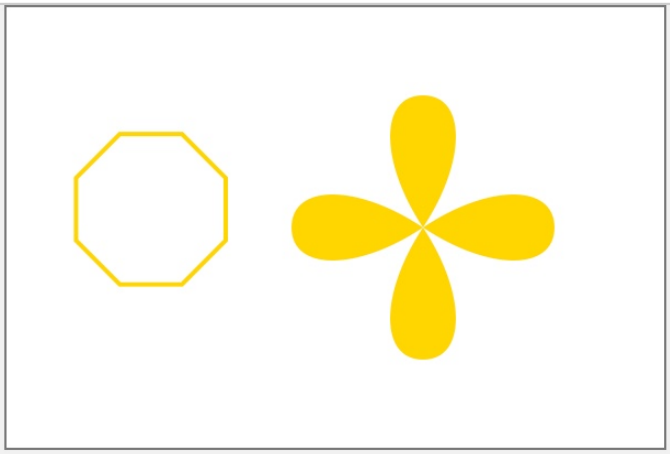


Path

The path element is how you "escape" the basic SVG shapes. In case none of the predefined shapes are good enough for you, you can draw any arbitrary shape you want using the path element. Paths are used in D3 to draw geo-shapes, glyphs and just about any creative design imaginable.

```
<svg width="300" height="200">
        <path d="M40,100 m40,-42.45 l20,20 v28.3 l-20,20h-28.3 l-20,-20 v-28.3 l20,-20z"
                  style="stroke: gold; fill: none; stroke-width: 2px;"/>
        <path d="M190,100 m0,-60 c20,0 20,30 0,60 c30,-20 60,-20 60,0 c0,20 -30,20 -60,0
c20,30 20,60 0,60 c-20,0 -20,-30 0,-60 c-30,20 -60,20 -60,0 c0,-20 30,-20 60,0 c-20,-30
-20,-60 0,-60z"
```

```
                    style="stroke: none; fill: gold;"/>
</svg>
```

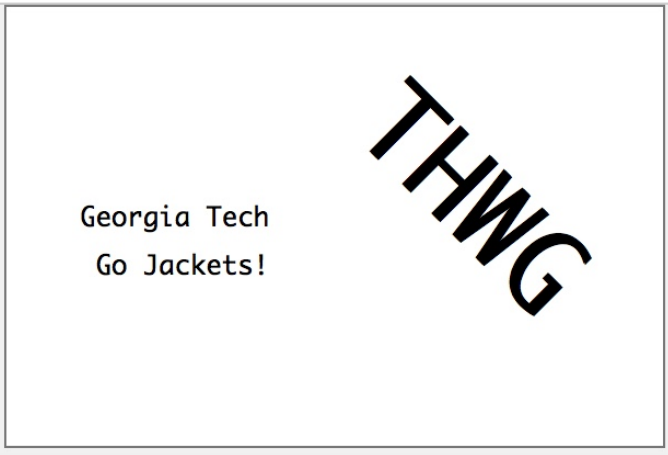


The path `d` attribute declares the points and line segments that define the shape of a path, its also really confusing to someone seeing if for the first time. If you ever find yourself wanting to learn how to create your own shapes, check out this great illustrated guide to the SVG path.

Text

By default, the x and y coordinates of a text element specify the position of its top left corner. The text "THWG" are translated and rotated. We'll get into how to translate and rotate elements in the following.

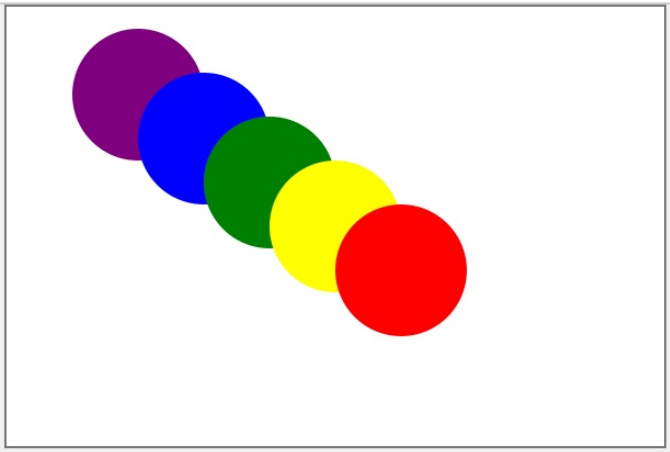
```
<svg width="300" height="200">
       <text x="120" y="100" style="text-anchor: end;">Georgia Tech</text>
       <text x="120" y="122" style="text-anchor: end;">Go Jackets!</text>

       <text transform=" translate(200,100)rotate(45)"
                style="text-anchor: middle; font-weight: bold; font-size:
50px;">THWG</text>
</svg>
```



Ordering

If you draw multiple shapes, they overlap based on the order presented in the code.

```
<svg width="300" height="200">
        <circle cx="60" cy="40" r="30" fill="purple"/>
        <circle cx="90" cy="60" r="30" fill="blue"/>
        <circle cx="120" cy="80" r="30" fill="green"/>
        <circle cx="150" cy="100" r="30" fill="yellow"/>
        <circle cx="180" cy="120" r="30" fill="red"/>
</svg>
```



In the above figure, the purple circle appears first in the code, so it is rendered first. Then, the blue circle is rendered "on top" of the purple one, then the green circle on top of that, and so on.

Grouping

`<g>` stands for group elements. The `<g>` element is used for logically grouping together sets of related SVG elements. Any styles you apply to the `<g>` element will also be applied to all of its descendants. This makes it easy to add styles, transformations, interactivity, and even animations to entire groups of objects.

For example, the following is an SVG version of Buzz. Buzz is made up of several shapes such as circles and paths:

Transforms

The SVG transformations that we will use are: *rotation, scaling, and translation*. In the example of buzz above, if we scale the entire `<g transform="scale(2)">` element then all the path and circle elements that make-up Buzz will also be scaled.

The transform attribute is used to specify one or more transformations on an element. An example of applying a transformation to an element may look like the following:

```
<g transform="translate(20, 20) rotate(40) translate(10)"></g>
```

This will transform the group in the following order:

- 20 pixels in the x-direction and 20 pixels in the y-direction
- rotate the group 40 degrees clockwise
- 10 pixels along the 40 degree line

Translation

To translate an SVG element, you can use the translate() function. The syntax for the translation function is:

```
translate(<tx>, [<ty>])
```

The `translate()` function takes one or two values which specify the horizontal and vertical translation values, respectively. tx represents the translation value along the x-axis; ty represents the translation value along the y-axis.

The ty value is optional; and, if omitted, it defaults to zero. The tx and ty values can be either space-separated or comma-separated, and they don't take any units inside the function—they default to the current user coordinate system units.

The following example translates an element by 100 user units to the right, and 300 user units to the bottom:

```
<circle cx="0" cy="0" r="100" transform="translate(100, 300)" />
```

Scaling

You can resize an SVG element by scaling it up or down using the scale() function transformation. The syntax for the scale transformation is:

```
scale(<sx>, [<sy>])
```

The `scale()` function takes one or two values which specify the horizontal and vertical scaling values, respectively. sx represents the scaling value along the x-axis, used to stretch or shrink the element horizontally; sy represents the scaling value along the y-axis, used to stretch or shrink the element vertically.

The sy value is optional; and, if omitted, it is assumed to be equal to sx. The sx and sy values can be either space-separated or comma-separated, and they are unitless numbers.

Rotation

You can rotate an SVG element using the rotate() function. The syntax for the function is:

```
rotate(<rotate-angle>, [<cx>, <cy>])
```

The `rotate()` function specifies a rotation by rotate-angle degrees about a given point. Unlike rotation transformations in CSS, you cannot specify an angle unit other than degrees. The angle value is specified unitless, and is considered a degrees value by default.

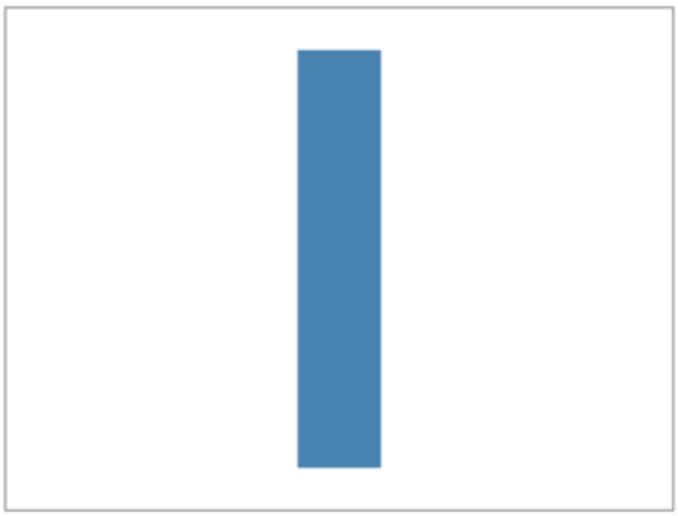
The optional cx and cy values represent the unitless coordinates of the point used as a center of rotation. If cx and cy are not provided, the rotation is about the origin of the current user coordinate system.

Activity 3: Creating Simple Visualizations Using SVG

In this activity, you will create simple visualizations using SVG.

Open `\lab1\activity_3\index.html` in your code editor (e.g. Sublime). Add the following code inside `<svg id="single-bar"></svg>` to create a single bar with the height of 250. The bar is colored steelblue.

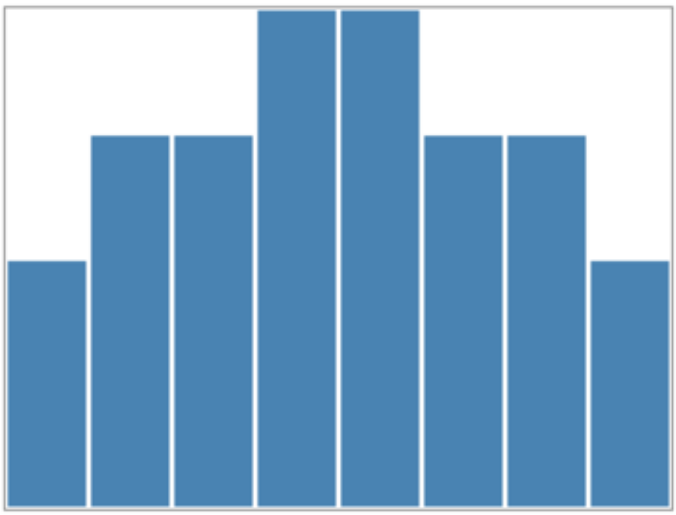
```
<rect x=175 y=25 width=50 height=250 fill="steelblue"></rect>
```



With this simple example, hopefully, you get a handle on how to create visualizations using SVG. Now, try to draw 1) a bar chart inside `<svg id="bar-chart"></svg>` , and 2) a line chart inside `<svg id="line-chart"></svg>` by following the descriptions below. You will be asked to submit the simple charts you created.

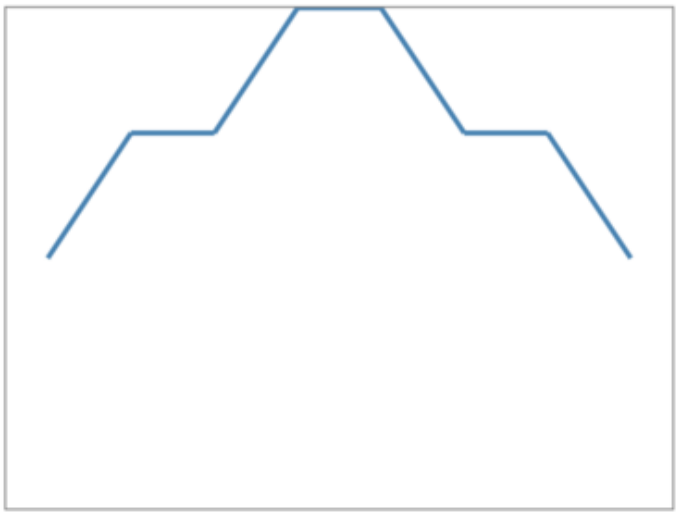
1) A bar chart inside `<svg id="bar-chart"></svg>`

All bars combined should have a total width of 400px. The individual bars should have the following height, in order: 150px, 225px, 225px, 300px, 300px, 225px, 225px, 150px. The bar charts will look something like this:

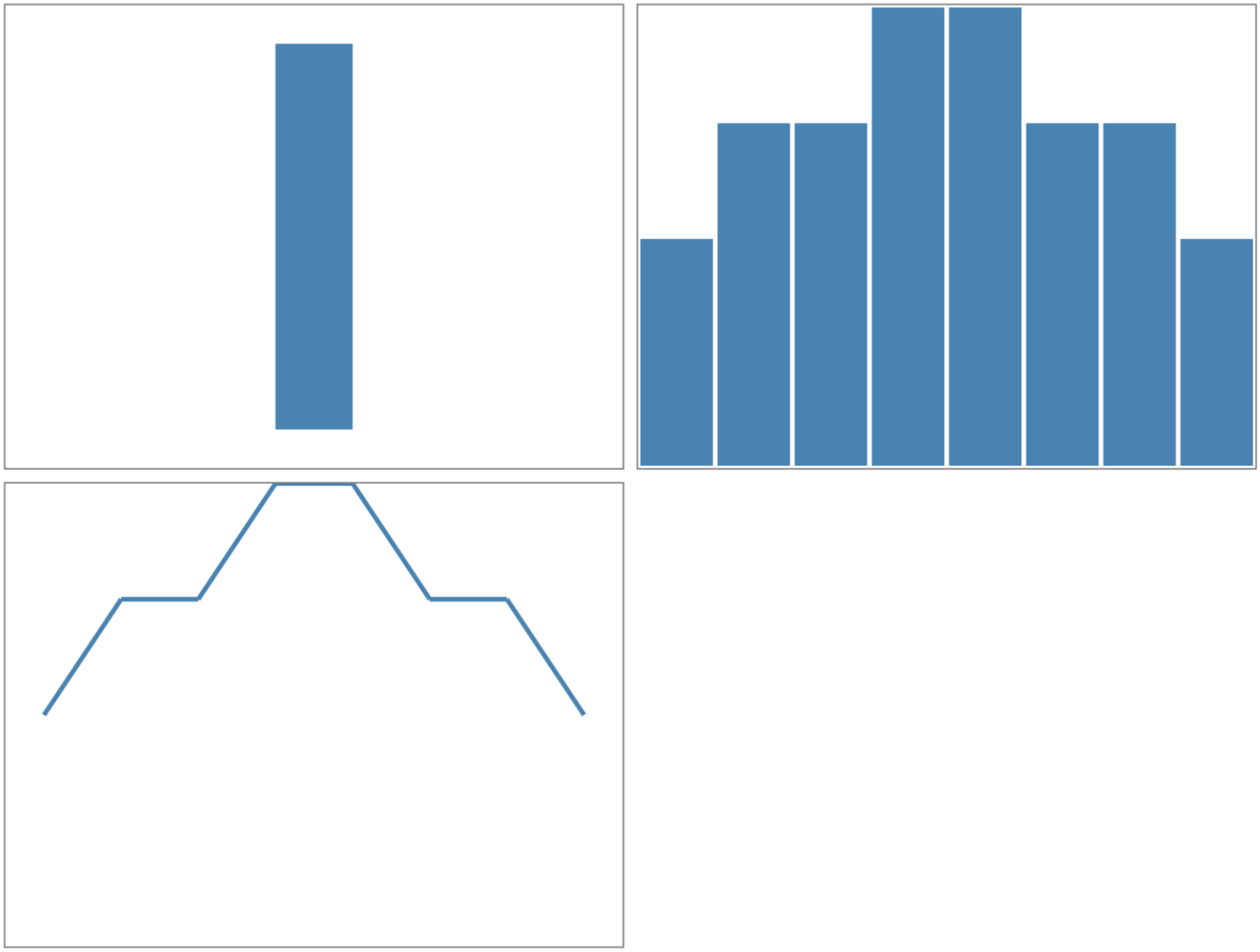


2) A line chart inside `<svg id="line-chart"></svg>`

Your line chart should be composed of the SVG line element. The heights of the vertices in the polyline should be the same as the heights of the individual bars for your bar charts. The resulting line chart should look close to the following figure:

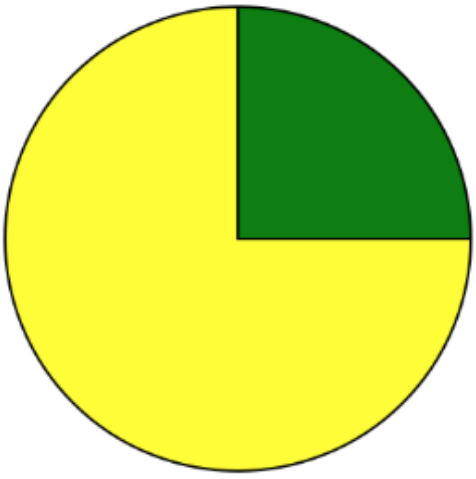


Your final output should look something like this:



3) A Pie Chart

Finally, following the same structure, add a pie chart to the bottom of your page. Your pie chart will have two wedges. The first wedge will span 90 degrees and will be green, and the second wedge will span the remaining 270 degrees, and will be yellow. From a data perspective, this means the green part is 25 percent and the yellow the other 75. Your pie chart should look like this:



This lab is based on the following material:

- Intro to Web Technologies (SVG Section) by Alex Lex of U. of Utah
- Understanding SVG Coordinate Systems and Transformations (Parts 1 + 2) by S. Soueidan
- Hanspeter Pfister's CS171 Lab Material (Harvard)
- Carlos Scheidegger's CS444 Course Assignment (Arizona)
- D3 - Interactive Data Visualization for the Web by Scott Murray

Grouping

Transforms

Activity 3: Creating Simple Visualizations Using SVG

1) A bar chart inside <svg id="bar-chart"></svg>

2) A line chart inside <svg id="line-chart"></svg>

3) A Pie Chart

Clone this wiki locally

```
https://github.gatech.edu/CS4460/Spring23-Labs-PUBLIC.wiki.git
```