CSC 595 Lab 1

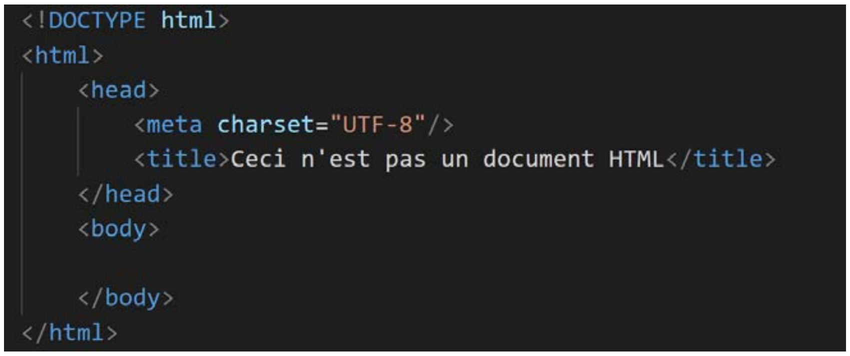
We will work on this in class, including a demonstration. The part at the end labeled "Assignment" must be completed and turned in on Dropbox by April 20.

Note heavy thanks to Carlos Scheidegger's materials, which I encourage you to read for a bit more on the subject. He links to Mozilla's references which are also great.

Scheidegger: https://cscheid.net/courses/fal17/csc544/lecture2.html

Mozilla Developer Network: https://developer.mozilla.org/en-US/

First off, creating a webpage is as simple as creating any text file. Open up an editor. You want to save it with a *html* extension so your operating system recognizes it as a web file. Then we can just open the file in a browser to be rendered. To put some actual HTML in a file, copy the below code (by typing it yourself, this is an image).



```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8"/>
        <title>Ceci n'est pas un document HTML</title>
    </head>
    <body>

    </body>
</html>
```

This is boilerplate code that you should copy for just about any web page. Note the overall structure:

- First is the simple declaration that this is an html file
- Second is the opening html tag. We open a tag of type X by writing <X> and close it with </X>
- There are two sections to the html document, each marked by its own tag, head and body
- The header includes the title, marked with its own tag, and a character set (encoding) setting telling the browser this file can contain Unicode characters (like non-English letters and emoji).
- The meta tag has an argument, the 'charset=' part, but it does not have contents. Therefore, there is no closing tag, it just ends with a />.

Content that we want on the page will be put in the body section. Since this is markup language, we can use different tags to markup our text. For example, let's mark some things as headers, bold and italics:

```
<h1><b>This</b> is my <u>title</u></h1>
```

Open the page in Chrome to see how this is rendered.

One especially useful type of html element is an SVG area, defined with an SVG tag. Include this in the body:

```
<svg width="400" height="400">

</svg>
```

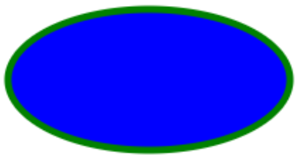
This creates a blank SVG canvas you can draw on by inserting graphical elements. When we create real visualizations, we will add the elements by code. Here are some examples from Scheidegger's materials:

CIRCLE

```
<svg width="400" height="200">
    <circle cx="200" cy="100" r="50"
             style="fill:yellow; stroke:brown; stroke-width:5px"/>
</svg>
```

ELLIPSE

```
<svg width="400" height="200">
    <ellipse cx="200" cy="100" rx="100" ry="50"
             style="fill:blue; stroke:green; stroke-width:5px"/>
</svg>
```

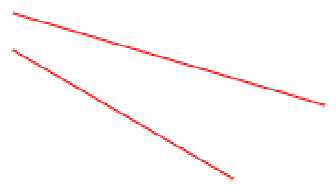


RECT (RECTANGLE)

```
<svg width="400" height="200">
    <rect x="50" y="50" width="200" height="100"
          style="fill:red; stroke:black; stroke-width:3px"/>
</svg>
```

LINE

```
<svg width="400" height="200" style="stroke: red">
    <line x1="30" y1="30" x2="200" y2="80"/>
    <line x1="30" y1="50" x2="150" y2="120"/>
</svg>
```



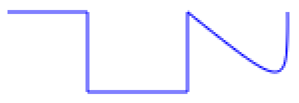
TEXT

```
<svg width="400" height="100">
    <text x="30" y="30">Some text</text>
    <text x="30" y="50">Some more text</text>
</svg>
```

Some text
Some more text

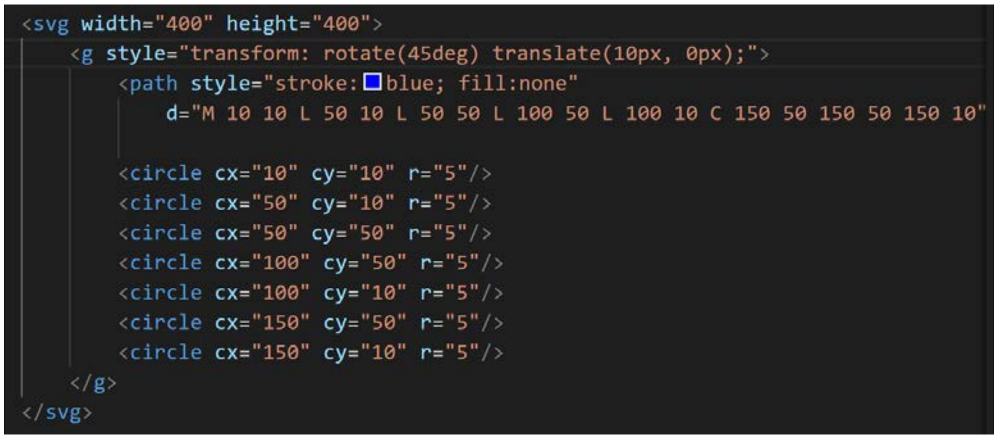
PATH

```
<svg width="200" height="60" style="stroke:blue; fill:none">
    <path d="M 10 10 L 50 10 L 50 50 L 100 50 L 100 10 C 150 50
    150 50 150 10"/>
</svg>
```



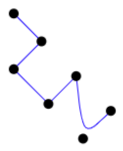
Path is especially powerful and hard to use. You can draw whatever you want with it, so it is worthwhile to play around with it a bit. Try putting some of these shapes together in your file. Note that the coordinate axes do not work as you expect – **the y axis points downward.** Finally, there is one error in this code that means it still works but won't perform as expected when more is added to the canvas – can you spot it?

Two more useful things: (1) you can apply transformations to these tags, like translating, rotating and skewing. This means if you wanted to draw that path above, but tilted 45 degrees, you don't have to calculate the positions where the points would be after rotation. Instead you can apply the 45 degree rotation to the path object. And (2), you can group tags together so that you can apply the same transformations to all of them at once. This is extremely useful, especially when we start talking about interactions. Here is an example where I have added a circle at each control point of the path and then used a group (*g* tag) to make them one group which I rotate. The rotation is covered on this page of the Mozilla documentation (https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/transform). Note that there are optional arguments to the rotation to specify what point to rotate around and the translation (moving left and right) happens before the rotation, i.e. moving to the right means moving along the 45 degree line in this case.



```
<svg width="400" height="400">
    <g style="transform: rotate(45deg) translate(10px, 0px);">
        <path style="stroke:blue; fill:none"
            d="M 10 10 L 50 10 L 50 50 L 100 50 L 100 10 C 150 50 150 50 150 10"

        <circle cx="10" cy="10" r="5"/>
        <circle cx="50" cy="10" r="5"/>
        <circle cx="50" cy="50" r="5"/>
        <circle cx="100" cy="50" r="5"/>
        <circle cx="100" cy="10" r="5"/>
        <circle cx="150" cy="50" r="5"/>
        <circle cx="150" cy="10" r="5"/>
    </g>
</svg>
```



Assignment

The assignment component is very simple – you will create two pages, each with different 'drawings' made from SVG elements. Submit the two HTML files.

I recommend drawing on paper first and thinking about how to calculate the coordinates.

1. Draw a bar chart
    a. You can make up a handful (5) of data points.
    b. Use rectangles, lines and text to create axes and labels along with the
2. Draw a smiley face ☺ !
    a. Use circles and paths
    b. Add a few lines of hair (use paths so they curve)