Data Visualization (CSCI 627/490)

Web Programming

Dr. David Koop

Languages of the Web

- HTML
- CSS
- SVG
- JavaScript
  - Versions of Javascript: ES6, ES2015, ES2020…
  - Specific frameworks: react, jQuery, bootstrap, D3

Hyper Text Markup Language (HTML)

- Markup languages allow users to encode the **semantics** of text
- Elements structure a document
  - Elements delineated by tags: `<h1>An element</h1>`
  - Document Object Model (DOM)
  - We can **navigate** this tree
- Identifying and Classifying elements: `id` and `class` attributes
  - `id` identifies a **single** element—use for a unique case
  - `class` may identify **multiple** elements—use for common cases
  - Each element may have multiple classes, separate by spaces
  - Use normal identifiers: don't start the name with a number

Cascading Style Sheets (CSS)

- Separate style from content, just specifies how to style the content
- Style information appears in three places: external, head, ~~individual elements~~
- Statement: `<selectors>: { <style definitions> }`
- Cascading:
  - use inheritance idea
  - properties that apply to children cascade down
- Selectors: element types (`strong`), ids (`#main-section`), classes (`.cool`)
  - Can combine to be more specific
    - `#main-section em, .cool > strong, p.cool`
  - Can group: `#main-section, p.cool { font-size: 16pt; }`

Example CSS

```
body {
   font-face: sans-serif;
   font-size: 12pt;
}
em { color: green; }
em u { color: red; }
em > strong { color: blue; }
img { border: 4px solid red; }
```

- What colors are displayed for this HTML (with the above stylesheet)?
  - `<em>This is <strong>cool</strong>. What about <u><strong>this?</strong></u></em>`

CSS Specificity

- Example:
  - CSS:

```
p.exciting { color: red; }
p { color: blue; }
```

  - What is the color of the paragraph

```
<p class="exciting">Cool</p>?
```

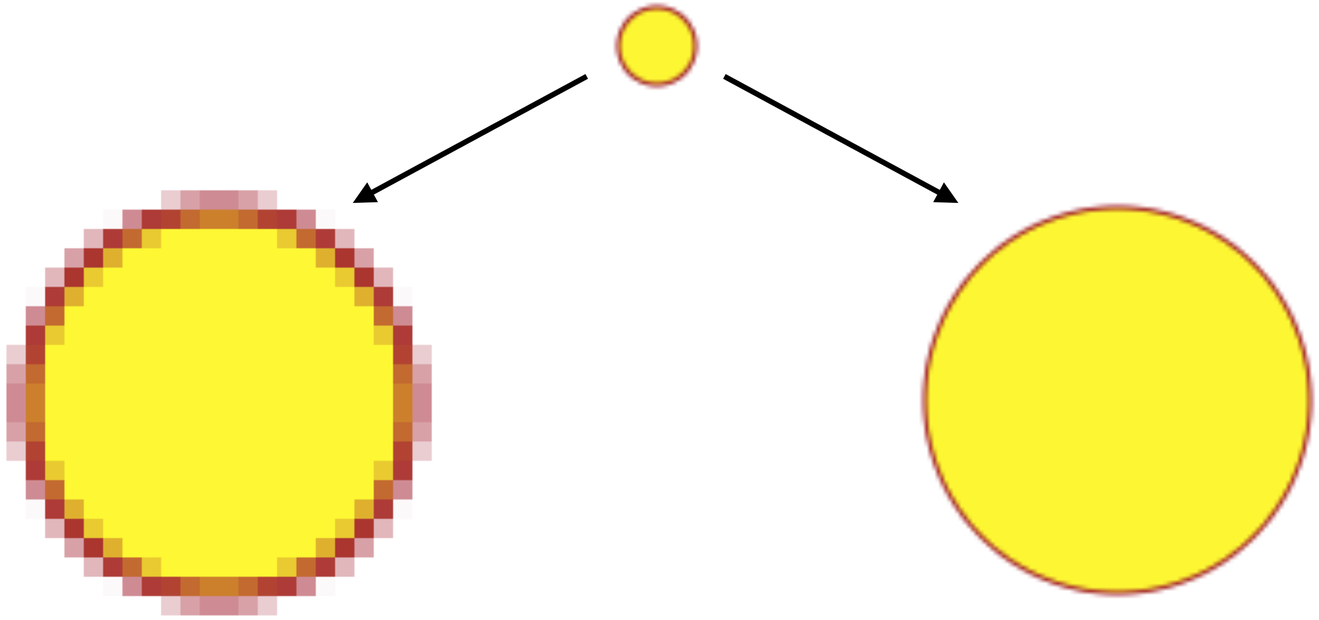
- Generally, last rule listed overrides previous rules
- …but anytime a selector is **more specific**, it has precedence
- `p.exciting` is a more specific selector
- When in doubt, **test it** in a browser

How to add CSS to HTML

- External: a separate file via a link element (in the `<head>` section):
  - `<link rel="stylesheet" href="styles.css">`
- Embedded: in the header:
  - `<style type="text/css"> … </style>`
- Inline: for a specific element: (**Discouraged!**)
  - `<p style="font-weight: bold;">Some text</p>`

Scalable Vector Graphics (SVG)

- Vector graphics vs. Raster graphics
- Drawing commands versus a grid of pixels
- Why vector graphics?



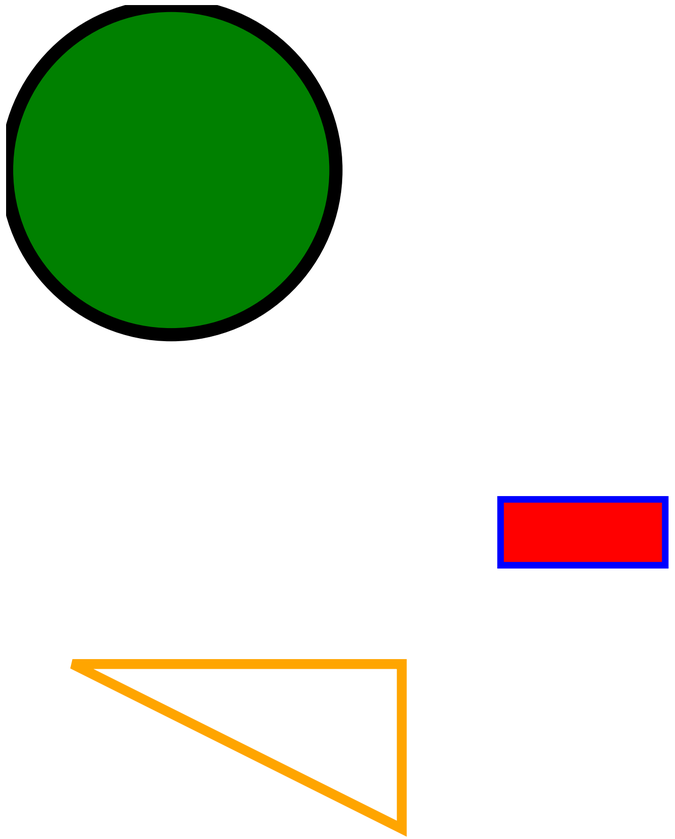
Raster

Vector

SVG Elements

- Markup language like HTML with similar XML syntax
- Pixel Coordinates: **Top-left** origin
- Drawing primitives:
  - Lines, Circles, Rects, Ellipses, Text, Polylines, Paths
  - Work by specifying information about **how** to draw the shape
  - Lots more: see MDN Documentation
- Ordering/Stacking:
  - SVG Elements are drawn in the order they are specified
- Paths: directions for drawing
  - https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths

SVG Example

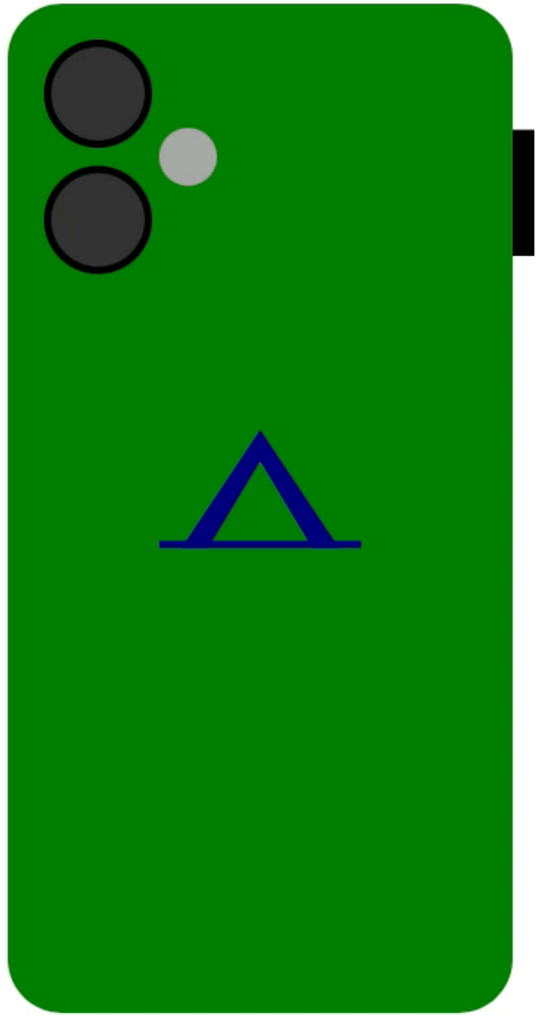


```
<svg id="mysvg" width="300" height="600">
  <circle cx="50" cy="50" r="50"/>
  <rect class="lego" x="150" y="150"
        width="50" height="20"/>
  <path id="triangle" d="M 20 200
        L 120 200 L 120 250 Z"/>
</svg>
```

```
circle { fill: green; stroke: black;
         stroke-width: 4px; }
.lego { fill: red; stroke: blue;
        stroke-width: 2px; }
#triangle { fill: none; stroke: orange;
            stroke-width:3px; }
```

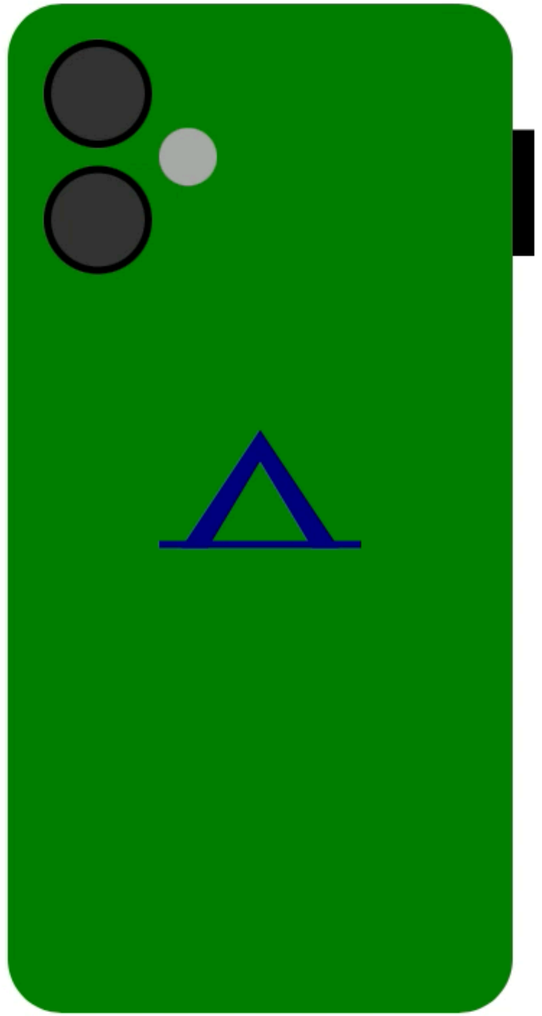
Assignment 1

- Write HTML, CSS, and SVG
- Text markup and styling (information)
- Drawing markup and styling (camera)
- Draw Bar chart using Plot library
- Due Wednesday, Jan. 28



Assignment 1

- Write HTML, CSS, and SVG
- Text markup and styling (information)
- Drawing markup and styling (camera)
- Draw Bar chart using Plot library
- Due Wednesday, Jan. 28



SVG Grouping

- Very powerful, useful for animations and transformations
- `<g> <circle …/> <circle … /> <circle … /></g>`
- Can add transforms to the group:

```
<svg width="200" height="200">
    <g transform="translate(0, 200) scale(1, -1)">
        <circle cx="50" cy="50" r="10"/>
        <circle cx="80" cy="80" r="10"/>
        <circle cx="110" cy="50" r="10"/>
        <circle cx="140" cy="90" r="10"/>
    </g>
</svg>
```

[SVG Example, Scheidegger, 2016]

JavaScript in one slide

- Interpreted and Dynamically-typed Programming Language
- Statements end with semi-colons, normal blocking with brackets
- Variables: `var a = 0; let b = 2; const d = 4;`
- Operators: `+, -, *, /, [ ]`
- Control Statements: `if (<expr>){…} else {…}, switch`
- Loops: `for, while, do-while`
- Arrays: `var a = [1,2,3]; a[99] = 100; console.log(a.length);`
- Functions: `function myFunction(a,b) { return a + b; }`
- Objects: `var obj = {x: 2, y: 4}; obj.x = 3; obj.y = 5;`
  - Prototypes for instance functions
- Comments are `/* Comment */` or `// Single-line Comment`

JavaScript References

- Learn Just Enough JavaScript: Introduction, P. Boffa
- Eloquent JavaScript, M. Haverbeke
- MDN Tutorials
- Interactive Data Visualization for the Web, Murray

JavaScript Objects

- `var student = {name: "John Smith", id: "000012345", class: "Senior", hometown: "Peoria, IL, USA"};`
- Objects contain multiple values: key-value pairs called **properties**
- Accessing properties via dot-notation: `student.name`
- May also contain functions:

```
- var student = {firstName: "John",
                 lastName: "Smith",
                 fullName: function() {
             return this.firstName + " " + this.lastName; }};
  student.fullName()
```

- JavaScript Object Notation (JSON): data interchange format
  - nested objects and arrays (data only, no functions!), **subset** of JavaScript

Objects as Associative Arrays/Dictionaries

- Objects have key-value pairs and can be addressed via those keys, either via dot-notation or via bracket notation: `[<key>]`
- Example:

```
states = {"AZ": "Arizona", "IL": "Illinois", …};
// Get a state's name given it's abbreviation
console.log("IL is " + states["IL"]);
```

- Similar to dictionaries or associative arrays in other languages (e.g. Python)
- Dot-notation only works with certain identifiers, bracket notation works with more identifiers
- Notebook

Functional Programming

Functional Programming in JavaScript

- Functions are first-class objects in JavaScript
- You can pass a function to a method just like you can pass an integer, string, or object
- Instead of writing loops to process data, we can instead use a `map/filter/reduce/forEach` function on the data that runs our logic for each data item
- `map`: transform each element of an array
- `filter`: check each element of an array and keep only ones that pass
- `forEach`: run the function for each element of the array
- `reduce`: collapse an array to a single object

Quiz

- Using `map`, `filter`, `reduce`, and `forEach`, and given this data:
  - `var a = [6, 2, 6, 10, 7, 18, 0, 17, 20, 6];`
- Questions:
  - How would I return a new array with values one less than in a?
  - How would I find only the values >= 10?
  - How would I sum the array?
  - How would I create a reversed version of the array?

Quiz Answers: Notebook

- Data: `var a = [6, 2, 6, 10, 7, 18, 0, 17, 20, 6];`
- How would I subtract one from each item?
  - `a.map(function(d) { return d-1; })`
- How would I find only the values >= 10?
  - `a.filter(function(d) { return d >= 10; })`
- How would I sum the array?
  - `a.reduce(function(s,d) { return s + d; })`
- How would I create a reversed version of the array?
  - `b = []; a.forEach(function(d) { b.unshift(d); });`
  - ...or `a.reverse() // modifies in place`
- Arrow functions shorten such calls: `a.map(d => d-1);`
  `a.filter(d => d >= 10); a.reduce((s,d) => s+d);`

Function Chaining in JavaScript

- When programming functionally, it is useful to chain functions
- No intermediate variables!
- Often more readable code
- jQuery Example:
  - `$("#myElt").css("color", "blue").height(200).width(320)`
- Used a lot in Web programming, especially D3
- Can return the same object or a new object
- Lazy chaining keeps track of functions to be applied but will apply them later (e.g. when the page loads)

Closures in JavaScript

- Functions can return functions with some values set
- Allows assignment of some of the values
- Closures are functions that "remember their environments" [MDN]

```
function makeAdder(x) {
  return function(y) {
    return x + y;
  };
}
var add5 = makeAdder(5);
var add10 = makeAdder(10);

console.log(add5(2));  // 7
console.log(add10(2)); // 12
```

- Notebook