

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)

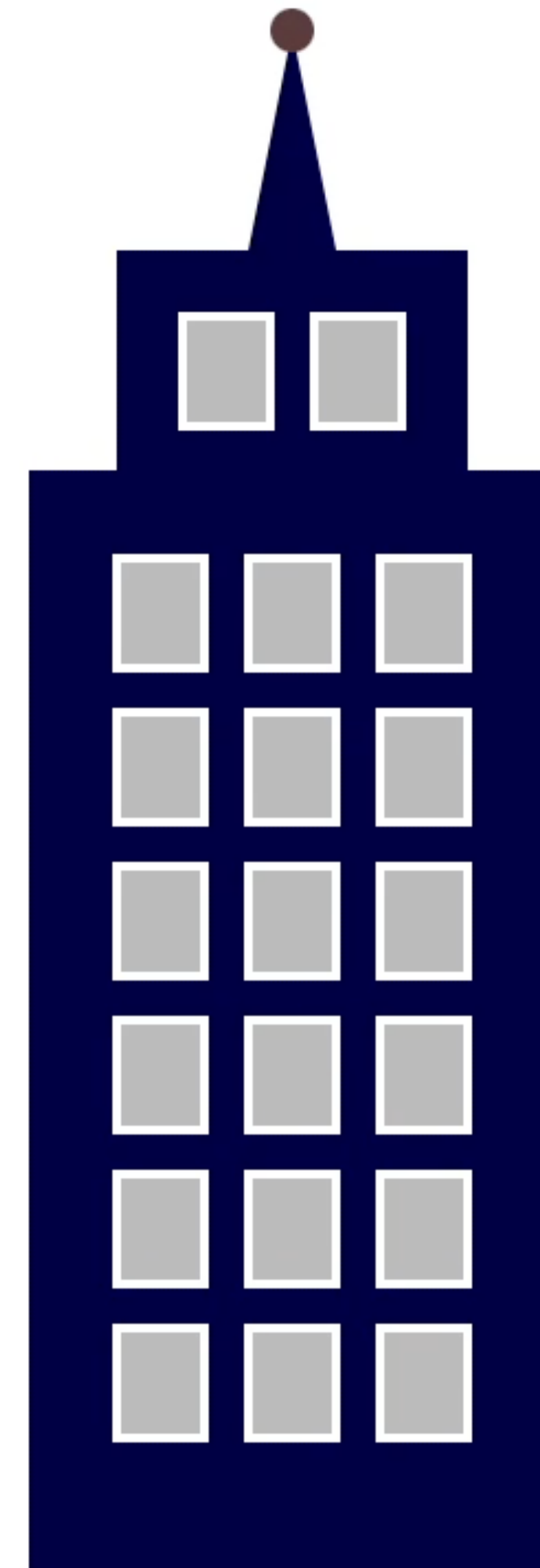
- Markup languages allow users to encode the **semantics** of text
- Elements structure a document
 - Elements delineated by tags: `<h1>An element</h1>`
 - Document Object Model (DOM)
 - We can **navigate** this tree
- Identifying and Classifying elements: `id` and `class` attributes
 - `id` identifies 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; }`

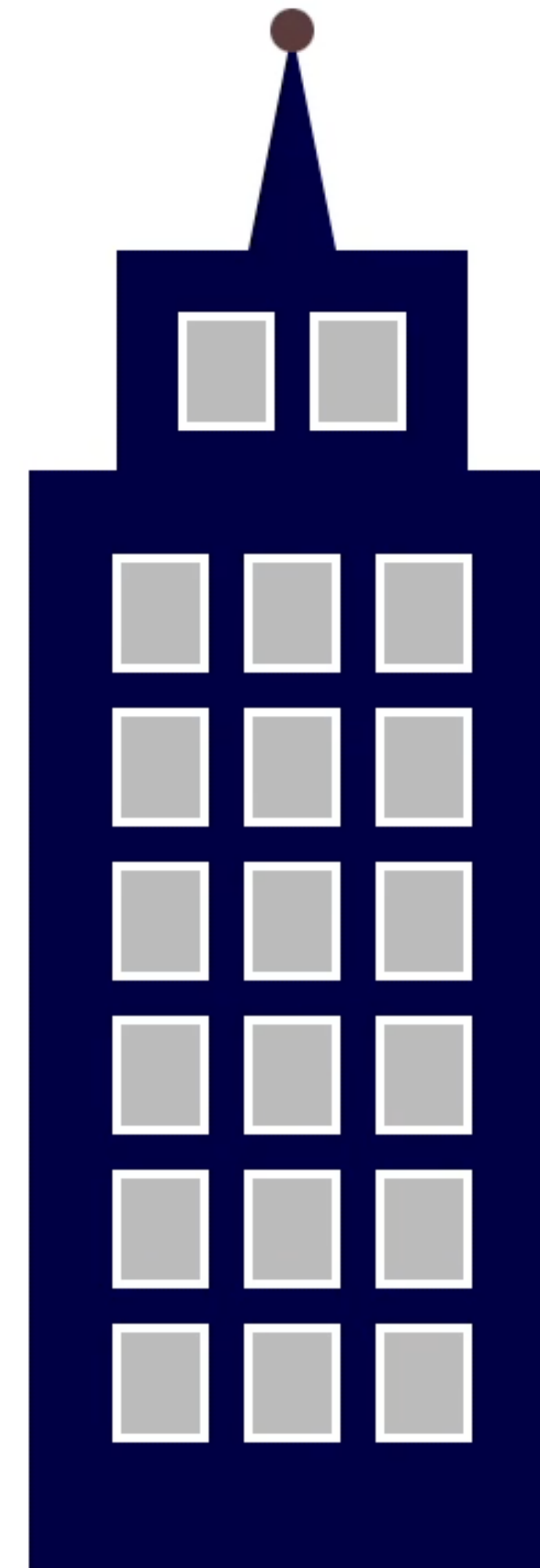
Assignment 1

- Write HTML, CSS, and SVG
- Use Plot library
- Text markup and styling (information)
- Drawing markup and styling (tower)
- Draw Bar chart



Assignment 1

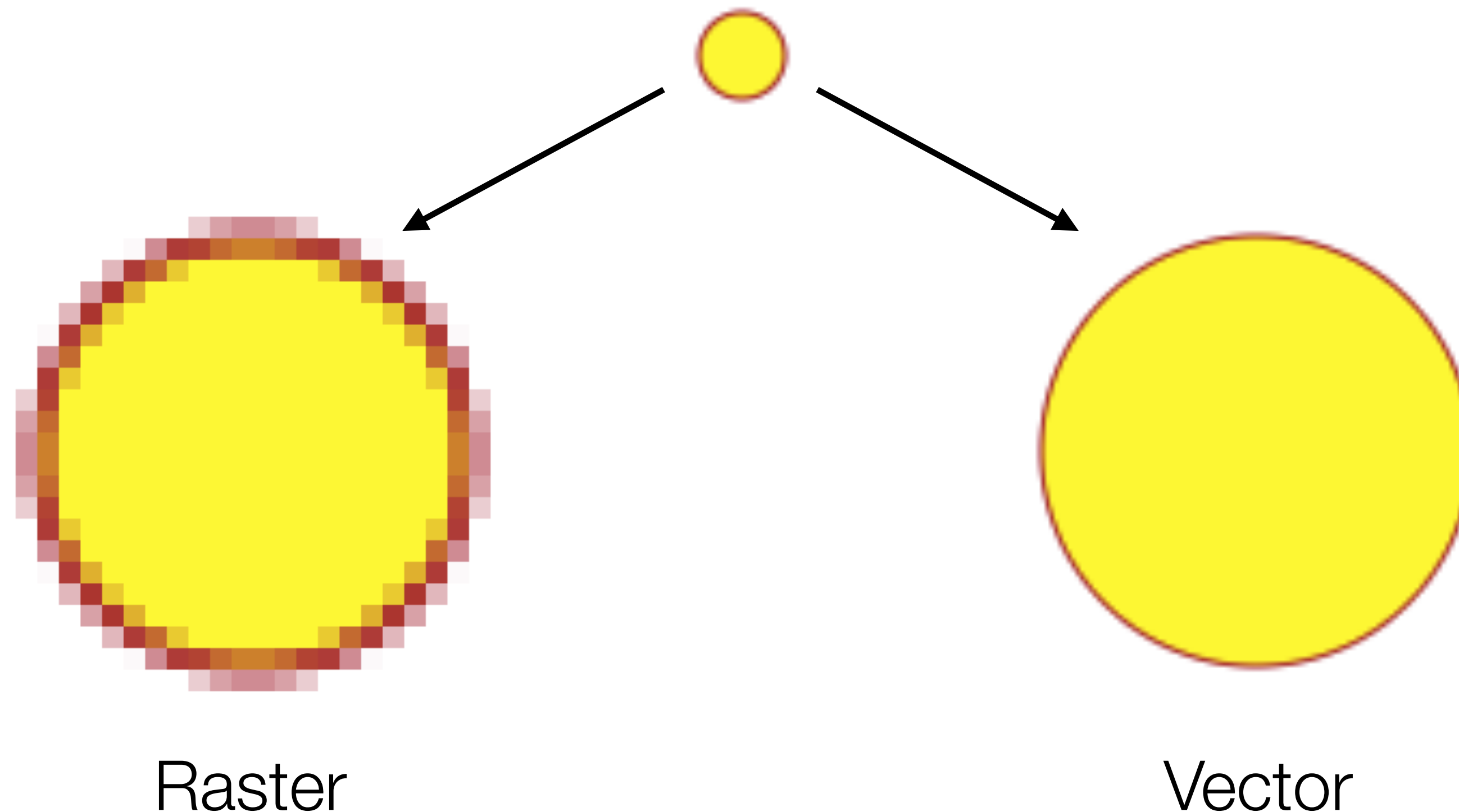
- Write HTML, CSS, and SVG
- Use Plot library
- Text markup and styling (information)
- Drawing markup and styling (tower)
- Draw Bar chart



What is the difference between
vector and raster graphics?

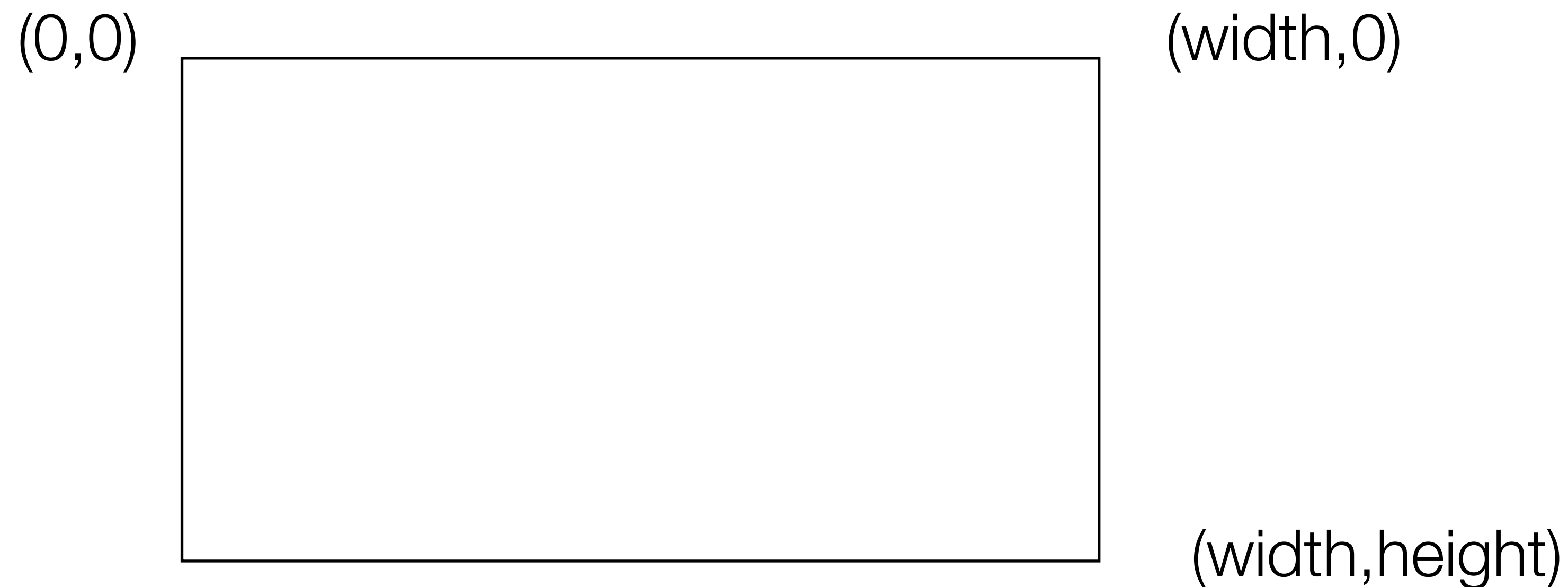
Scalable Vector Graphics (SVG)

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



SVG Background

- Another markup language:
 - Describe the shapes and paths by their endpoints, characteristics
- SVG can be embedded into HTML5 documents!
- Pixel Coordinates: **Top-left** origin



SVG Elements

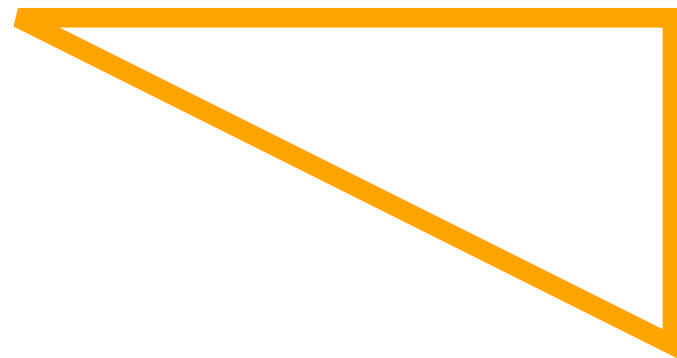
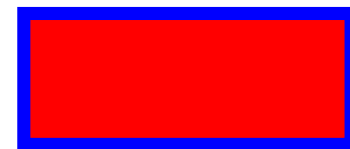
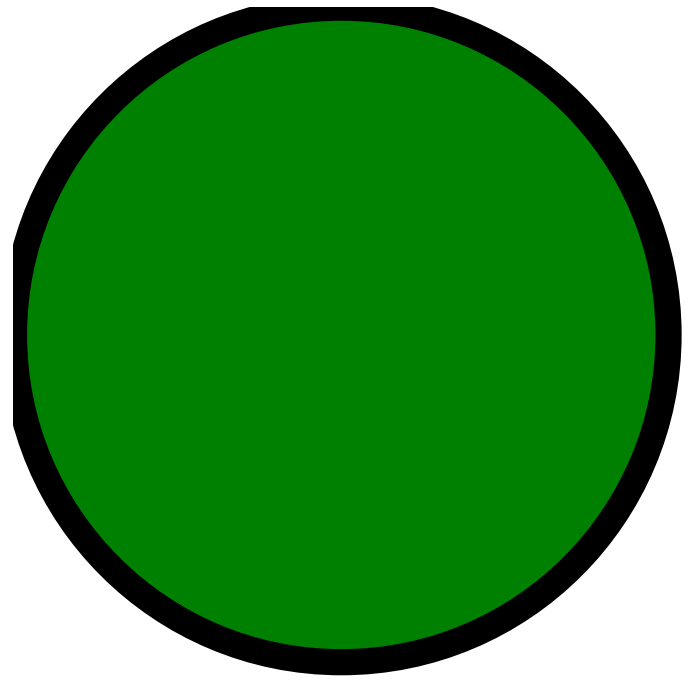
- Drawing primitives:
 - Lines, Circles, Rects, Ellipses, Text, Polylines, Paths
 - Work by specifying information about **how** to draw the shape
 - Lots more: see [MDN Documentation](#)
- Ordering/Stacking:
 - SVG Elements are drawn in the order they are specified
- Paths: directions for drawing
 - <https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths>

SVG Styles

- We can specify styles of SVG elements in CSS!
- Example:

```
circle { fill: green; stroke: black;
        stroke-width: 4px; }
.normal { fill: red; stroke: blue;
          stroke-width: 2px; }
#p1 { fill: none; stroke: red; stroke-width: 3px; }
```

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; }
```

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/Dictionaryes

- 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?

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

Manipulating the DOM with JavaScript

- Key global variables:
 - `window`: Global namespace
 - `document`: Current document
 - `document.getElementById(...)`: Get **one** element via its id
 - `document.querySelector(...)`: Get **one** element via selector
 - `document.querySelectorAll(...)`: Get **all** matching elements via selector
- HTML is parsed into an in-memory document (DOM)
- Can access and **modify** information stored in the DOM
- Can add information to the DOM

Example: JavaScript and the DOM

- Start with no real content, just divs:

```
<div id="firstSection"></div>  
<div id="secondSection"></div>  
<div id="finalSection"></div>
```

- Get existing elements:

- `document.querySelector/querySelectorAll`
- `document.getElementById`

- Programmatically add elements:

- `document.createElement`
- `document.createTextNode`
- `Element.appendChild`
- `Element.setAttribute`

Bears

Chicago, IL

2018-2019 NFC North Champions



What will happen this year?

Example (continued): Using Data to Build Content

Bears

Chicago, IL

2018-2019 NFC North Champions

September 9: Loss (23-24)

September 17: Win (24-17)

September 23: Win (16-14)

September 30: Win (48-10)

October 14: Loss (28-31)

October 21: Loss (31-38)

October 28: Win (24-10)

November 4: Win (41-9)

November 11: Win (34-22)

November 18: Win (25-20)

November 22: Win (23-16)

December 2: Loss (27-30)

December 9: Win (15-6)

December 16: Win (24-17)

December 23: Win (14-9)

December 30: Win (24-10)

- We can loop through data to add content to a web page (schedule and results)
- Data:

```
[{"date": "September 9", "opponent": "Green Bay Packers", "home": false, "win": false, "score": "23-24"}, ... ]
```
- Can use `forEach` to iterate through each game and build content
- Or, for...of loop:

```
for (const game of data)
```
- Notebook

Observable's HTML Templating

- Allows JavaScript expressions to be **inlined** in HTML (or SVG content)
- Use `${...}`
- Example:
 - [JavaScript] `name = "Prof. Koop"`
 - [HTML] `<p>Hello, my name is ${name}</p>`

Using Observable's HTML Templating

```
<div id="firstSection">
  <h1>Bears</h1><p>Chicago, IL</p>
</div>
<div id="secondSection">
  <h2>2018-2019 NFC North Champions</h2>
</div>
<div id="finalSection">
  ${scores.map( (game) => html`<p>${game.date} :  

    ${game.win ? "Win" : "Loss"} (${game.score})</p>` ) }
  </img>
  <p>What will happen this year?</p>
</div>
```

Notebook

Creating SVG figures via JavaScript

- SVG elements can be accessed and modified just like HTML elements
- Create a new SVG programmatically and add it into a page:
 - ```
var divElt = document.getElementById("chart");
var svg = document.createElementNS(
 "http://www.w3.org/2000/svg", "svg");
divElt.appendChild(svg);
```
- You can assign attributes:
  - ```
svg.setAttribute("height", 400);  
svg.setAttribute("width", 600);  
svgCircle.setAttribute("r", 50);
```

Manipulating SVG via JavaScript

- SVG can be navigated just like the DOM
- Example:

```
function addEltToSVG(svg, name, attrs) {  
    var element = document.createElementNS(  
        "http://www.w3.org/2000/svg", name);  
    if (attrs === undefined) attrs = {};  
    for (var key in attrs) {  
        element.setAttribute(key, attrs[key]);  
    }  
    svg.appendChild(element);  
}  
mysvg = document.getElementById("mysvg");  
addEltToSVG(mysvg, "rect", {"x": 50, "y": 50,  
    "width": 40, "height": 40,  
    "fill": "blue"});
```

- Notebook