CS 571 - Data Visualization & Exploration

SVG and Javascript

Instructor: Hamza Elhamdadi

UMassAmherst

Upcoming Dates

Feb 28 (Friday): Project Proposal Due

Homework 2 will be released Feb 28 (due Mar 14)

What We've Covered So Far

Theory of Data Visualization

- What is Data Visualization?
- Perception & Cognition
- Color
- Data Abstraction
- ...

Programming Tools for Interactive Data Visualization

- HTML and CSS
- ...

Today's Class

Theory of Data Visualization

- What is Data Visualization?
- Perception & Cognition
- Color
- Data Abstraction
- ...

Programming Tools for Interactive Data Visualization

- HTML and CSS
- SVG
- Javascript
- ...

SVG (Scalable Vector Graphics)

SVG (Scalable Vector Graphics)

So far, we've focused on textual elements in HTML

But, to create data visualizations, we need graphics

SVG (Scalable Vector Graphics)

SVG is a subset of HTML5 that allows us to create graphics that scale with our browser window

D3js creates and manipulates SVG elements

Alternatives we won’t cover: Canvas and WebGL

SVG (Scalable Vector Graphics)

To create an SVG drawing, we can use an SVG element

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Page Title</title>
</head>
<body>
    <svg width="400" height="300"></svg>
</body>
</html>
```

Make sure to specify the width and height attributes (in pixels)

Circles

To draw a circle, use the <circle> tag

```
 7  <body>
 8      <svg width="400" height="300">
 9        <circle cx="200" cy="100" r="50"
                  style="fill: gray;
                         stroke: black;
                         stroke-width: 5px;" />
10      </svg>
11  </body>
```

Attributes: cx (x coordinate), cy (y coordinate), and r (radius)

CSS Styles: fill (color of the circle), stroke (color of the outline), stroke-width (size of the outline)

Coordinate System

SVG coordinates (x, y) originate from the top-left

```
7  <body>
8      <svg width="400" height="300">
9        <circle cx="50" cy="50" r="20" style="fill: red;" />
10       <circle cx="100" cy="100" r="20" style="fill: blue;" />
11       <circle cx="150" cy="150" r="5" style="fill: green;" />
12     </svg>
13 </body>
```

Inconsistencies between HTML and SVG

Some HTML attributes like title don’t work with SVG

```
 7 <body>
 8     <svg width="400" height="300">
 9       <circle cx="10" cy="10" r="5" />
10       <title>Information on Hover!<title>
11     </svg>
12 </body>
```

Instead, <title> is its own element in SVG

Ellipses

To draw an ellipse, use the <ellipse> tag

```
 7  <body>
 8      <svg width="400" height="300">
 9        <ellipse cx="200" cy="100" rx="100" ry="50"
                  style="fill: gray;
                         stroke: black;
                         stroke-width: 5px;" />
10      </svg>
11  </body>
```

Attributes: cx (x coordinate), cy (y coordinate), rx (x radius), and ry (y radius)

Rectangles

To draw a rectangle, use the <rect> tag

```
 7  <body>
 8      <svg width="400" height="300">
 9        <rect x="50" y="50" width="200" height="100"
                  style="fill: gray;
                         stroke: black;
                         stroke-width: 5px;" />
10      </svg>
11  </body>
```

Attributes: x (x coordinate of top left corner),
y (y coordinate of top left corner),
width, and height

Lines

To draw a straight line, use the <line> tag

```
 7 <body>
 8     <svg width="400" height="300">
 9       <line x1="50" y1="50" x2="200" y2="100"
                 style="stroke: black; stroke-width: 5px; />
10     </svg>
11 </body>
```

Attributes: first point (x1, y1) and second point (x2, y2)

Paths

You can use the <path> element to draw complex shapes

The <path> element uses its own micro-language.
Some example commands:

- M 10 10 (Moves the position without drawing a line)
- L 50 10 (draws a Line from previous position to the position specified)
- Z (closes a path using a straight line to the first point)
- C (draws Curves)

We typically won't write path code manually. We'll use D3 for this.
But its important to understand how D3 works under the hood.

Paths

Here’s an example <path>

```
 7  <body>
 8      <svg width="400" height="300">
 9        <path d=" M 20 20
                   L 60 20
                   L 60 60
                   L 110 20
                   L 110 20
                   C 110 60
                     160 60
                     160 20 " style="fill: none; stroke: black"/>
12      </svg>
13  </body>
```

Ordering

The order in which elements are drawn is the order in which they appear

```
 7 <body>
 8     <svg width="400" height="300">
 9       <rect x="50" y="50" width="200" height="100"
               style="fill: lightgray;" />
10       <circle cx="250" cy="100" r="50"
                 style="fill: steelblue;" />
11     </svg>
12 </body>
```

Here, the circle is drawn on top of the rectangle

Ordering

The order in which elements are drawn is the order in which they appear

```
 7 <body>
 8     <svg width="400" height="300">
 9       <circle cx="250" cy="100" r="50"
                 style="fill: steelblue;" />
10       <rect x="50" y="50" width="200" height="100"
                style="fill: lightgrey;" />
11     </svg>
12 </body>
```

Here, the rectangle is drawn on top of the circle

Grouping

We can group elements using the <g> element to apply atttributes to several elements at once.

```
 7 <body>
 8     <svg width="400" height="300">
 9       <g fill="steelblue">
10         <rect x="0" y="0" width="10" height="10" />
11         <circle cx="50"  cy="50"  r="10" />
12         <circle cx="50"  cy="100" r="10" />
13         <circle cx="100" cy="100" r="10" />
14       </g>
15     </svg>
16 </body>
```

Transforming

We can transform elements using the “transform” attribute

```
 7 <body>
 8     <svg width="400" height="300">
 9       <g transform="translate(150,150) scale(1,-1)"
           fill="steelblue">
10         <rect x="0" y="0" width="10" height="10" />
11         <circle cx="50"  cy="50"  r="10" />
12         <circle cx="50"  cy="100" r="10" />
13         <circle cx="100" cy="100" r="10" />
14       </g>
15     </svg>
16 </body>
```

The transform attribute is read from right to left

Transforming (CAUTION)

Pay attention to what you are transforming

```
 7 <body>
 8     <svg width="400" height="300">
 9       <g transform="translate(100,100) scale(1,-1)"
           fill="steelblue">
10         <text x="10" y="50">This text is upside down</text>
11       </g>
12     </svg>
13 </body>
```

Practice

Create four rectangles:

- Give them the same **x** and **height** (x = 0, height = 20)
- Space the rectangles 10 pixels apart in the y direction
  - if rectangle 1 is at **y = 0**, rectangle 2 should be at
    - **y = rect_1_y + height + 10** = 0 + 20 + 10 **= 30**
- Give each rectangle a **width**
- Give rectangle 2 an **id**
- Give rectangles 3 and 4 the same **class**
- Add styles to the id and class to change the color of the rectangles

Sample Output:

Javascript

Javascript

With HTML, CSS, and SVG, we can make mostly static websites

But, in this class, we're interested in interactive visualizations.

We can use Javascript to make our web visualizations interactive.

Javascript

Javascript is a dynamically typed language (like Python)

- No int, float, string, etc.
- Don’t need to declare variables ahead of time

Javascript is object-oriented

- but inheritance is implemented using prototypes

The Very Basics of Javascript

Writing to the console using console.log

```
1 console.log('Message');
2 console.log(22 / 7);
3 console.log(Math.PI);
```

The Very Basics of Javascript

Declaring variables

```
 1 a = 0;   // a number
 2 b = "1"; // a string
 3 c = ["Paul", "John", "Ringo", "George"]; // an array
 4 d = [1981, 1984, 1954, 1949]; // another array
 5 e = [1, 2, "3", [4]]; // also a valid array (but don't)
 6 f = false; // a boolean
 7
 8 //checking the type of a variable
 9 console.log("Type of f:", typeof(f));
10 console.log("Type of a:", typeof(a));
```

Note: These are all global variables

The Very Basics of Javascript

Declaring local variables

```
var a = 3; // function-scoped, don't use var
let name = "Hamza"; // block-scoped
const birthplace = "Tampa, FL"; // constant, block-scoped

birthplace = "Amherst, MA"; // throws an error

const TEST = [2, 3, 4]; // Complex data types remain mutable
console.log("Original Array:", TEST);
TEST[2] = 9;
console.log("Modified array:", TEST);
```

The Very Basics of Javascript

Variable Scope:

- let and const create block-scoped variables
  - the variable is only accessible inside the block it was declared (e.g., inside an if statement)
  - Note: blocks are defined with curly braces {}
- var creates a function-scoped variable
  - meaning the only way to isolate it is to wrap it in a function
  - not recommended
- No identifier creates a global variable
  - meaning the variable is accessible anywhere in the code

The Very Basics of Javascript

Operations with variables

```
let sum = 1 + 3;
sum += 1;
sum++;
sum--;
let divResult = sum / 13;
let mod = sum % 2;

typeof(divResult);

let compoundString = "two " + "strings";

let templateString = `Result of divide = ${divResult}`;
```

Arrays

Array Basics

```
let numArray = [15, 21, 21, 4];
let empty = []; // empty array declaration
console.log(numArray[0]);

// you can do this, but you shouldn't
let multiTypeArray = [0, "This", "is", true, "unfortunately"];
console.log(multiTypeArray.length);

let nested = [[1,2],[3,4],[5,6]];
console.log(nested[1][0]);
```

Arrays

Array Methods

```
let numArray = [15, 21, 21, 4];

numArray.push(3);
let newLength = numArray.push(4);

let lastElement = numArray.pop();

let pos = numArray.indexOf(21);
console.log("Index:", pos);
```

Arrays

Array Sorting

```
 1 numArray.sort();
 2 console.log("Sorted as strings: ", numArray);
 3
 4 function compareNumbers(a,b){
 5   return a - b;
 6 }
 7
 8 numArray.sort(compareNumbers);
 9 console.log("Sorted as numbers: ", numArray);
10
11 numArray.sort(function(a,b) { return b - a });
12 console.log("Reverse sorted: ", numArray);
```

Objects

Object Basics

```
let obj = {
  key1: 3,
  key2: 4
};
console.log("Value of key1:", obj.key1);
console.log("Value of key2:", obj["key2"];

let obj2 = {
  "key3": 3,
  "key4": 4
};
```

Objects

More Object Basics

```
let obj = {
  name: "John Doe",
  birthYear: 1954,
  nationality: "USA",
  countries: ["USA", "Germany", "Honduras"]
};

obj.favColor = "blue";
console.log("favorite color: ", obj.favColor);
```

Control Structures

Control Structure Basics

```
 1 if(1 === parseInt("1")) {
 2   console.log("First if");
 3 } else if (2 === 3) {
 4   console.log("Else if");
 5 } else {
 6   console.log("Else");
 7 }
 8 if (1 == "1") console.log("double equals ignores type");
 9
10 if (1 === "1") console.log("won't print");
11 else console.log("safer to use triple equals");
```

Control Structures

Ternary Operators

```
1 // CONDITION ? HAPPENS_IF_TRUE : HAPPENS_IF_FALSE
2
3 true === true ? console.log("true") : console.log("false");
4
5 let a = false;
6 let b = a ? 20 : 30;
7
8 console.log(b / 10);
```

Control Structures

Switch statements

```
let i = "some case";
switch(i) {
  case "string literals are okay":
    console.log("this case doesn’t happen");
    break;
  case "some case":
    console.log("string matches!");
    break;
  default:
    console.log("don't forget to break after each case");
}
```

Loops

For Loops

```
let output = "";
for (let i = 0; i < 10; ++i){
  output += i + ", ";
}
console.log("for loop result: " + output);
```

Loops

While Loops

```
let i = 3;
output = "";
while(i < 100){
  output += `${i}, `;
  i = i * 2;
}
console.log(`while loop result: ${output}`);
```

Loops

forEach function

```
let years = [1954, 1949, 1981, 1982];

years.forEach(function (d) {
  console.log(d);
});
```

Loops

For Of Loops

```
let years = [1954, 1949, 1981, 1982];

for (let year of years) {
  console.log(years);
}
```

Loops

For In Loops

```
let years = [1954, 1949, 1981, 1982];

for(let year in years){ // don't use for in loops
  console.log(year); // you don't get the year
  console.log(years[year]); // you get the object keys
}

let obj = { a: 1, b: 2, c: 3 };
for(let property in obj) console.log(property, obj[property]);
```

Functions

Function Basics

```
 1 function numberFunction(myNumber) {
 2   return (myNumber < 10) ? myNumber : myNumber * myNumber;
 3 }
 4
 5 console.log(numberFunction(30));
 6 console.log(numberFunction(-5));
 7 console.log(numberFunction("50"));
 8 console.log(numberFunction("what"));
 9 console.log(numberFunction(30, "huh?"));
10 console.log(numberFunction());
```

Functions

Anonymous Functions

```
let variableFunction = function(v) {
  return (v > 10) ? "big" : "small";
}
console.log(variableFunction(30));

variableFunction = function(x) { return x - 5; }
console.log(variableFunction(30));
```

Functions

Functions as Parameters

```
1 let variableFunction = function(v) {
2   return (v > 10) ? "big" : "small";
3 }
4
5 let myNumbers = [1, 2, 3, 4];
6 console.log( myNumbers.map(variableFunction) );
7 console.log( myNumbers.map(function(d){ return d * 2 }) );
```

Functions

Arrow Functions

```
let myNumbers = [13, 16, 19, 22];
console.log( myNumbers.map( d => d * 2) );

let f1 = () => 12;
console.log( "f1: " + f1() );

let f2 = x => x * 2;
console.log( "f2(4) = " + f2(4) );
```

Functions

Arrow Functions

```
let f3 = (x, y) => {
  let z = x * 2 + y;
  y++;
  x *= 3;
  return (x + y + z) / 2;
}

console.log( `f3(3,4) = ${f3(3,4)}` );
```

Practice

1. Create the following variable:
    - let **arr** = [{ val: “10” }, { val: “25” }, { val: “twenty” }];
2. Write a function **oddNegatives** that:
    - takes as input an object with key **val** which contains a string
    - parses the value of **val** to a number **num**
    - if **num** is even, returns **num**
    - if **num** is odd, returns **-num**
    - if **num** is NaN, returns 0 (hint: use isNaN)
3. **Map** the array **arr** using oddNegatives
4. Create a **count** variable and initialize it to zero.
5. Use a **for of** loop to add each number in the mapped array to count
6. Log **count** to the console

Object-Oriented Javascript

Prototypical Inheritance

```
let base = { v1: 1, v2: 2 };
let derived = { v1: 5, v3: 3 };
console.log("base.v1:", base.v1);
console.log("derived.v1:", derived.v1);
console.log("derived.v2:", derived.v2);

Object.setPrototypeOf(derived, base);
console.log("derived.v1", derived.v1);
console.log("derived.v2", derived.v2);
```

Object-Oriented Javascript

More Prototypical Inheritance

```
let empty = Object.create(null); // same as {}
let base = Object.create(empty); // base inherits from empty
base.color = "blue";

let derived = Object.create(base); // derived inherits base
console.log(derived.color);
```

Object-Oriented Javascript

Classes

```
 1 class Base {
 2   constructor(first, second){
 3     this.first = first;
 4     this.second = second;
 5   }
 6   multiply() { return this.first * this.second }
 7 };
 8
 9 let base = new Base(2, 4);
10 console.log("Base multiply: " + base.multiply());
```

Object-Oriented Javascript

Class Inheritance

```
 1 class Derived extends Base {
 2   constructor(first, second, third){
 3     super(first, second);
 4     this.third = third;
 5   }
 6   multiply() { return this.first * this.second * this.third }
 7 };
 8
 9 let derived = new Derived(2, 4, 6);
10 console.log("Derived multiply: " + derived.multiply());
```

The special this variable

this Basics

```
 1 function createObj(value) {
 2   return {
 3     x: 3,
 4     get: function () { return this.x },
 5     set: function (val) { this.x = val }
 6   };
 7 }
 8 let o = createObj();
 9 console.log( "x:", o.get() );
10 o.set(5);
11 console.log( "x after setting:", o.get() );
```

The special this variable

Be careful with this

```
 1 function createObj(value) {
 2   return {
 3     x: 3,
 4     get: function () { return this.x },
 5     set: function (val) { this.x = val }
 6   };
 7 }
 8 let o = createObj();
 9 console.log( "x:", o.get() );
10 let f = obj.get;
11 console.log( "x?", f() );
```

Default Values

```
 1 function sum(x = 11, y = 31) {
 2   console.log(x + y);
 3 }
 4 sum();
 5 sum(5,6);
 6 sum(0,42);
 7 sum(5);
 8 sum(5, undefined);
 9 sum(5, null);
10 sum(undefined, 6);
11 sum(null, 6);
```

Lazy Expressions

```
1 function randomNum(){
2   return Math.random();
3 }
4
5 function fun(id=randomNum()){
6   console.log(id);
7 }
8 fun(3);
9 fun();
```

Spread and Gather Operators

Spread

```
1 function spread(a, b, c) {
2   console.log("values:", a, b, c);
3 }
4 spread(...[1, 2, 3]);
```

Spread and Gather Operators

Gather

```
1 function gather(x, y, ...z) {
2   console.log(x, y, z);
3 }
4 gather(1, 2, 3, 4, 5, 6, 7, 8);
```

Destructuring

Array Destructuring

```
1 function abc() {
2   return [1,2,3];
3 }
4 let [x, y, z] = abc();
5 console.log(x, y, z);
```

Destructuring

Object Destructuring

```
 1 function xyz(){
 2   return {a: 1, b: 2, c: 3};
 3 }
 4
 5 let {a, b, c} = xyz();
 6 console.log(a, b, c);
 7
 8 const person = { name: "John Doe", address: "LGRT" };
 9 const { name, address } = person;
10 console.log(name, address);
```