

Data Visualization

Nathan Warner



**Northern Illinois
University**

Computer Science
Northern Illinois University
United States

Contents

1	Web programming: Html, css, and JS	2
1.1	HTML and CSS	2
1.2	JS	7

Web programming: Html, css, and JS

1.1 HTML and CSS

- **SVGS with html:** An SVG (Scalable Vector Graphics) file is an XML-based image format used to display vector graphics. Unlike PNG or JPG images, SVGs scale infinitely without losing quality.

Key properties:

- Resolution-independent
 - Small file size for simple graphics
 - Fully stylable with CSS
 - Scriptable with JavaScript
 - Ideal for icons, diagrams, charts, and UI graphics
- **Embedding SVG directly into HTML (inline SVG):** This is the most powerful and flexible method.

```
1 <svg width="200" height="100" viewBox="0 0 200 100">
2   <rect x="10" y="10" width="180" height="80"
   ↪ fill="steelblue" />
3   <circle cx="100" cy="50" r="30" fill="orange" />
4 </svg>
```

- <svg> defines the canvas
 - width / height define display size
 - viewBox defines the internal coordinate system
 - Shapes (rect, circle, line, path) are drawn inside
 - Fully stylable with CSS
 - Can be animated
 - JavaScript access to elements
 - Best for interactive graphics
- **SVG elements:**

Element	Purpose
<rect>	Rectangle
<circle>	Circle
<ellipse>	Ellipse
<line>	Line
<polyline>	Connected lines
<polygon>	Closed shape
<path>	Complex shapes
<text>	Text

- **SVG viewBox:**

```
1 <svg viewBox="0 0 200 100">
```

Means:

- Coordinate system starts at (0, 0)
- Width = 200 units
- Height = 100 units

This allows scaling without distortion.

- **CSS for <svg>:**

- fill
- fill-opacity
- fill-rule
- stroke
- stroke-width
- stroke-opacity
- stroke-linecap
- stroke-linejoin
- stroke-dasharray
- stroke-dashoffset
- stroke-miterlimit
- color
- opacity
- x
- y
- cx
- cy
- r
- rx
- ry
- width
- height
- transform
- transform-origin
- transform-box
- font-family
- font-size
- font-style
- font-weight
- letter-spacing

- word-spacing
 - text-anchor
 - dominant-baseline
 - alignment-baseline
 - direction
 - writing-mode
 - display
 - visibility
 - overflow
 - clip-path
 - mask
 - filter
 - cursor
 - pointer-events
 - animation
 - animation-name
 - animation-duration
 - animation-delay
 - animation-iteration-count
 - animation-timing-function
 - transition
 - transition-property
 - transition-duration
- **Paths:** The `<path>` element is the most powerful and flexible shape in SVG. Unlike `<rect>` or `<circle>`, a path can describe:
 - Straight lines
 - Curves
 - Arcs
 - Complex shapes
 - Icons, symbols, letters
 - Entire illustrations

It works by following a series of drawing commands stored in the *d* attribute. The *d* string is a mini drawing language, it reads left to right.

- **Move to (M):** Moves the “pen” without drawing.

M x y

Moves to (x, y)

- **Line to (L):** Draws a straight line.

L x y

Draws a line from the current point to (x, y)

- **Close path (Z):** Closes the shape by connecting back to the start.

Z

- **Quadratic Curve (Q):** (cx, cy) = control point (x, y) = end point

Q cx cy x y

The control point:

- * pulls the curve
- * bends its direction
- * determines how steep or shallow the curve is

The curve only passes through:

- * The start point
- * The end point

- **Cubic Bézier (C):**

C x1 y1, x2 y2, x y

Two control points, more control.

- **Arc Command (Rounded Shapes) (A):**

A rx ry x-axis-rotation large-arc sweep x y

Note the difference between lowercase and uppercase control characters

- **Uppercase:** Absolute position
- **Lowercase:** Relative movement

- **Triangle with path:**

```
1 <path d="M 50 10 L 30 80 L 70 80 Z" />
```

- **Fill and stroke:**

```
1 <path d="M 50 10 L 30 80 L 70 80 Z"
2 stroke="black"
3 stroke-width="6"
4 fill="red"
5 />
```

- **Grouping:** The <g> element is used to group multiple SVG elements together so that transformations, styles, or attributes can be applied to them collectively.

```
1 <svg width="200" height="200">
2   <g>
3     <!-- SVG elements go here -->
4   </g>
5 </svg>
```

Instead of transforming each element individually, you apply the transformation once to the group.

```
1 <g transform="translate(50, 50)">
2   <circle cx="0" cy="0" r="20" />
3   <rect x="30" y="-10" width="40" height="20" />
4 </g>
```

Both shapes move together. You can apply styles such as fill, stroke, opacity, etc., to all child elements.

```
1 <g fill="blue" stroke="black" stroke-width="2">
2   <circle cx="50" cy="50" r="20" />
3   <rect x="90" y="30" width="40" height="40" />
4 </g>
```

Events applied to a <g> affect all child elements.

```
1 <g onclick="alert('Clicked group!')">
2   <circle cx="50" cy="50" r="20" />
3   <rect x="80" y="40" width="30" height="30" />
4 </g>
```

1.2 JS

- **Var, let, and const:**

- **Var:** Function-scoped, not block-scoped. Ignores `{}` blocks such as `if`, `for`, and `while`. Hoisted to the top of the function. Initialized as **undefined**. Can be reassigned, can be redeclared
- **Let:** Block-scoped, exists only inside `{}` where it is defined. Hoisted, but not initialized. Exists in the Temporal Dead Zone (TDZ) until declared. Can be reassigned, cannot be redeclared in the same scope

```
0 let x = 3;
1 x = 4;    // OK
2 let x = 5; // Error
```

- **Const:** Block-scoped, same as `let`. Hoisted but in the TDZ. Must be initialized at declaration

```
0 const z = 10; // Good
1 const y;     // Error
```

Cannot be reassigned. Const prevents reassignment, not mutation.

- **Immutable types:** These cannot be changed after creation. Any “modification” creates a new value. Primitive Types are all immutable

- * number
- * string
- * boolean
- * null
- * undefined
- * symbol
- * bigint

```
0 let s = "hello";
1 s[0] = "H";    // No effect
2 console.log(s); // "hello"
```

- **Mutable types:** These are objects and collections, whose contents can change without changing the reference.

- * Object
- * Array
- * Function
- * Date
- * Map / Set

- **Pass by value and pass by reference:** JavaScript does not technically have pass-by-reference.

- * Primitive values are passed by value
- * Objects are passed by value of their reference

All primitives are passed by value.

Objects are somewhat passed by reference, technically by value of reference. The reference (memory address) is copied, not the object itself.

```
0  function modify(obj) {  
1      obj.x = 10;  
2  }  
3  
4  const data = { x: 1 };  
5  modify(data);  
6  
7  console.log(data.x); // 10
```

- * data holds a reference to the object
- * That reference is copied into obj
- * Both point to the same object
- * Mutating the object affects both

– **Objects:** Key value pairs

```
0  var obj = {x: 2, y: 4}; obj.x = 3; obj.y = 5;
```

Prototypes for instance functions. We can access properties via dot notation or [] notation. Objects may also contain functions

```
0  var student = {firstName: "John",  
1      lastName: "Smith",  
2      fullName: function() { return this.firstName + " "  
        ↪      + this.lastName; }},  
3  student.fullName()
```

Note: Dot-notation only works with certain identifiers, bracket notation works with more identifiers, like if the key was a string.

- **JSON:** Data interchange format, subset of JS. Uses nested objects and arrays. Data only, no functions.
- **Functional programming in JS**