CREATE WITH DATA

FUNDAMENTALS OF HTML, SVG, CSS & JAVASCRIPT FOR DATA VISUALISATION

Peter Cook

Fundamentals of HTML, SVG, CSS and JavaScript for Data Visualisation

Peter Cook

Fundamentals of HTML, SVG, CSS and JavaScript for Data Visualisation

This version was published on 2nd May 2025.

This is a preview version.

Table of Contents

Chapter 1. Introduction

Welcome! This book covers the fundamentals of HTML, SVG, CSS and JavaScript for data visualisation.

If you work with data you might spend most of your time using Excel, R or Python. You might be tasked with creating charts in a Python Jupyter notebook or creating a dashboard using R's Shiny or Python's Dash. Often these charts are built with HTML, SVG, CSS and JavaScript and a good understanding of these languages is useful when you need to dive deeper.

You might be interested in learning D3 or other JavaScript charting libraries, in which case this book can help you get up to speed with HTML, SVG, CSS and JavaScript.

This book doesn't assume prior knowledge of these languages, but it's helpful if you've some coding experience. Being familiar with a text editor (or IDE) is also helpful.

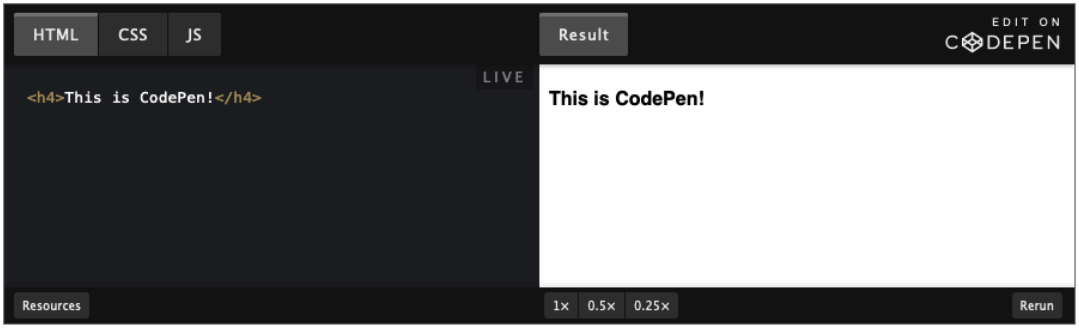
It's by no means a comprehensive tutorial on HTML, SVG, CSS and JavaScript. You'd need a much bigger book for that. Think of this book as presenting the minimum of what you ought to know if you're wanting to work with web based data visualisations. If you'd like to understand my other data visualisation books and courses (such as D3 Start to Finish and Visualising Data with JavaScript) you definitely need to understand the content in this book.

1.1. Setting up

In order to get started quickly I recommend using CodePen to look at and experiment with code examples (see next section). At the end of this book, in the **Tools and Set-up** chapter I show you how to set up a project so you can write code locally (where the files are stored on your computer instead of on CodePen). This takes a bit of time to set up but I recommend this approach if you're serious about web development.

1.2. CodePen

CodePen is a web site that lets you experiment with HTML, SVG, CSS and JavaScript without having to set up any tools.



At key points in the book you'll find a link to a CodePen example. If you visit the link you can view and experiment with the code. I encourage you to experiment with the CodePen examples - there's very little you can break and learning through experimentation is very effective.

You can freely edit the CodePen examples from this book and your changes don't affect the original example. If you want to keep your changes, click Fork (at the bottom of the page) and create a CodePen account (or log in). Save your work by clicking the Save button at the top of the page.

1.3. Stay in touch

I love to stay in touch with my readers. One of the best ways to do this is via my mailing list. I send occasional messages containing useful information related to implementing data visualisations (e.g. using JavaScript or other tools). There'll also be discount codes for my other books. You can sign up here.

Chapter 2. Web languages: HTML, SVG, CSS & JavaScript

Most websites and web applications (including data visualisations) are built from 4 languages: **HTML**, **SVG**, **CSS** and **JavaScript**.

HTML stands for Hyper Text Markup Language and describes the **content** of a webpage.

SVG stands for Scalable Vector Graphics and describes **shapes** such as circles, lines and rectangles. It's often used when visualising data (to describe bars, lines, circles, axes, legends etc.).

CSS stands for Cascading Style Sheets and describes the **style** (such as font weight, colour or size) and **position** of HTML and SVG elements.

JavaScript is a programming language that can **manipulate** HTML and SVG, do general **computation** (such as data manipulation) and handle **interaction**.

We look at each of these in upcoming chapters.

Chapter 3. JavaScript

JavaScript is the programming language of the web. It's built into the majority of web browsers and can:

- **add**, **modify** and **remove** HTML and SVG elements
- handle **interactivity** (for example, mouse clicks)
- **fetch** data and other resources
- perform general purpose **computation** (such as data processing)

Web-based data visualisations, particularly if they're interactive, probably use JavaScript.

JavaScript started out as a little used language for websites (it was typically used to handle a few user interactions) but is now a widespread language with hundreds of tools and libraries in its ecosystem.

Libraries such as D3, Leaflet and Chart.js are written in JavaScript and provide useful features for building data visualisations.

The next two sections will cover the basics of JavaScript namely **variables** and **types**. Subsequent sections will look at **operators**, **conditionals**, **iteration** and **functions**.

In order to experiment and practise JavaScript I recommend using the jsconsole website. This allows you to enter JavaScript code and it evaluates it immediately. (If you're more experienced you can use your browser's debug console or Node.)

When you see example code in this section you'll occasionally see `// ` followed by a value:

```
10 + 5;  // 15
```

The value indicates the value that the code returns. (If you type `10 + 5` into jsconsole and press return it'll show the result `15` on the next line.)

3.1. JavaScript variables

If you have a value (such as some text or a number) you can use a **variable** to point to that data. In effect you're giving the data a label so that you can refer to it elsewhere in your code. In JavaScript, a variable must be **declared** before it's used. Typically this is done using the `let` keyword:

```
let myMessage;
```

> `let` is the modern way of declaring a variable. The older way is to use `var`. The differences are subtle and are beyond the scope of this book. You can read more here.

> JavaScript statements usually end with a semicolon `;`. This is optional but we'll use them in this book.

You can assign a value (such as a number or string) to a variable. Here's how to assign a string (a piece of text enclosed between quotation marks) to the variable `myMessage`:

```
myMessage = "Hello world!";
```

You can also assign numbers to variables:

```
let a = 10;
let b = 20;
```

(The above code uses a shorthand where the variable is declared and assigned in a single statement.)

Any expression can be assigned to a variable.

> An expression is a piece of code that evaluates to a single value such as `"hello"` or `2 * 10`. These evalute to `"hello"` and `20`, respectively. An expression can also consist of variables. For example `a + b` evaluates to `30`.

Let's assign the expression `a + b` to a new variable `c`:

```
let c = a + b;
c;  // 30
```

The `a + b` expression is evaluated and the result `30` is assigned to `c`.

3.2. Exercises

Open jsconsole and type the following:

```
let a = 10;
let b = 20;
```

Now type `a` followed by the return key. What do you see?

jsconsole evaluates `a` and outputs the result `10`. Now type:

```
let c = a + b;
```

Now type `c` followed by the return key. The `c` variable will be evaluated and its value `30` is ouput.

3.3. JavaScript data types

The most common data types in JavaScript are:

- strings (e.g. `"Hello world"`)
- numbers (e.g. `1234`, `10.567`, `1.23e2`)
- booleans (`true` and `false`)
- arrays (lists of values)
- objects (lists of key-value pairs)

The above are sufficient to build data visualisations. We'll look at strings, numbers and booleans in this section. There are separate chapters for arrays and objects.

3.3.1. Strings

Strings represent text. The text is enclosed by single quotes (') or double quotes ("):

```
'A JavaScript string'
"A string enclosed in double quotes"
```

If your text contains a single quote you use double quotes to enclose it (and vice versa):

```
"Bob's learning to code in JavaScript"
```

You can add strings together using the `+` operator:

```
let firstName = "Jolene";
let lastName = "Smith";
let fullName = firstName + " " + lastName;
fullName;  // "Jolene Smith"
```

In the following examples we've a variable `a` with value `"This is a string"`:

```
let a = "This is a string";
```

You can get the length of a string using `.length`:

```
a.length;  // 16
```

You can get the nth character in the string using square brackets:

```
a[0];  // "T"
a[2];  // "i"
```

String manipulation

A string can be split according to where a shorter string `s` occurs using `split(s)`. For example let's split the string `a` by a single space character:

```
a.split(' ');  // ["This", "is", "a", "string"]
```

(The square brackets represent an array which we'll cover in a subsequent section.)

You can compute the upper case version of a string using `toUpperCase()`:

```
a.toUpperCase();  // "THIS IS A STRING"
```

Note that this doesn't change (or mutate) the string. It just returns a new string and `a`'s value is still `"This is a string"`. To make variable `a` upper case you can assign the output of `a.toUpperCase()` to `a`:

```
a = a.toUpperCase();
a;  // "THIS IS A STRING"
```

There's a lower case counterpart called `toLowerCase()`.

String conversion

You can convert a string containing a number into a number using `parseFloat`. For example:

```
parseFloat('123.456');  // 123.456
parseFloat('5432');     // 5432
parseFloat('abc');      // NaN (not a number)
parseFloat('1234abc');  // 1234
parseFloat('0');        // 0
parseFloat('');         // NaN
```

Note that passing an empty string into parseFloat returns `NaN` which means 'not a number'.

> It's quite common to see the `+` operator used to convert strings into numbers. For example `+'123'` evaluates to `123`. You need to use with caution because converting an empty string (e.g. `+''`) evaluates to 0. Sometimes when working with data, missing data is represented by an empty string and if this is converted using `+` it'll evaluate as 0 which isn't always desirable.

Exercises

Open jsconsole and type the following:

```
a = "This is a string";
a.length;
a[2];
a.split(' ');
a.toUpperCase();
```

Check that your output is similar to:

```
16
"i"
["This", "is", "a", "string"]
"THIS IS A STRING"
```

3.3.2. Numbers

JavaScript has a number type for representing numbers. Integers (whole numbers) and decimals may be represented:

```
let a = 123;
let b = 123.456;
```

The mathematical operators `+` (addition), `-` (subtraction), `*` (multiplication) and `/` division may be applied to numbers:

```
let c = 123 + 456;
let d = 3 * 123;
```

Brackets can be used to enforce precedence:

```
(10 + 20) * 2;  // 60
10 + (20 * 2);  // 50
```

The first expression evaluates as `60` and the second one `50`. (An expression is a piece of code that evaluates to a single value.)

The following examples use a variable `n` with value `123.456`:

```
let n = 123.456;
```

You can round numbers using `round`, `floor` and `ceil`.

`round` returns the closest integer, `floor` rounds the number down and `ceil` rounds the number up:

```
Math.round(n);  // 123
Math.floor(n);  // 123
Math.ceil(n);   // 124
```

You can get the string representation of a number using `toString()`:

```
n.toString();  // "123.456"
```

You can get the square root of a number using `Math.sqrt()`:

```
Math.sqrt(100);  // 10
```

Exercises

Open jsconsole and type the following:

```
let n = 123.456;
Math.round(n);
n.toString();
Math.sqrt(100);
```

Your output values should be:

```
123
"123.456"
10
```

3.3.3. Booleans

Boolean types can have two values `true` and `false`:

```
var learningCSS = true;
var learningJavaScript = true;
var learningPython = false;
```

You'll see later on (in the operators section) that booleans are returned by operators such as `===` (is equal to) and `>` (is greater than).

3.4. JavaScript arrays

Arrays represent **lists of values**. Each value can be of any type, even arrays! Arrays are one of the fundamental data structures used when working with data. (You'll see in the section on objects that it's common to represent a table of data using an array.) An array is represented by **square brackets** and each value is separated by a **comma**. Here are 4 different arrays:

```
[ 10, 20, 30 ]                    // an array of numbers
[ "An", "array", "of", "strings" ]  // an array of strings
[ [ 10, 20 ], [ 30, 40 ] ]        // an array of arrays
[ true, true, false ]             // an array of booleans
```

3.4.1. Accessing array elements

You can access the `i`th item in an array by writing `[i]` after the array:

```
let a = [ "An", "array", "of", "strings" ];
a[0];  // "An"
a[2];  // "of"
```

Arrays are indexed from `0`. So the first element in an array is referenced by `[0]`. For nested arrays such as:

```
let a = [ [ 10, 20 ], [ 30, 40 ] ];
```

you access the values by stringing together the square brackets:

```
a[0];     // [10, 20]
a[0][1];  // 20
```

`a[0]` is the first element of `a` and evaluates to `[10, 20]`.

`a[0][1]` means index `1` of `a[0]` and evaluates to `20`.

3.4.2. Array operations

In the following table, these variables are used:

```
let a = [ 10, 20, 30 ];
let b = [ 40, 50, 60 ];
let c = [ "An", "array", "with", "text" ];
```

Useful array operations include the following:

Operation	Examples	Result
Get array length	`a.length`	`3`
Sort an array	`c.sort()`	`c` is now `["An", "array", "text", "with"]`
Reverse an array	`a.reverse()`	`a` is now `[30, 20, 10]`
Get a portion (or 'slice') of an array. The 1st number indicates the start index of the slice. The 2nd number indicates the end index (inclusive) of the slice.	`c.slice(1, 2)`	`["array", "with"]`
Join each element of an array with a given string	`a.join(' and ')`	`"10 and 20 and 30"`
Join two arrays	`a.concat(b)`	`[10, 20, 30, 40, 50, 60]`
Add an element to the end of an array	`b.push(70)`	`b` is now `[40, 50, 60, 70]`
Remove (and return) the last element of an array	`a.pop()`	Evaluates to `30`. `a` is now `[10, 20]`

Take care when sorting

Be careful when sorting arrays of numbers. The following example:

```
let d = [11, 10, 9];
d.sort();
```

results in `d` being `[10, 11, 9]`. Which is rather surprising.

It's because JavaScript sorts as strings, even if it's an array of numbers! It's really strange. Read https://stackoverflow.com/questions/48201502/strange-javascript-sorting-bug for more insight.

Another quirk of JavaScript is that some of the above operations change the original array while others don't. `sort` and `reverse` overwrite the array with the new order. Similarly, `push` and `pop` change the length of the original array.

The remaining operations just evaluate to a value. They don't have any effect on the array.

3.4.3. Exercises

Open jsconsole and type the following:

```
let a = [10, 20, 30];
a.length;
a.reverse();
a.push(0);
a;
```

The final value of `a` should be `[30, 20, 10, 0]`.

The first expression just outputs the length of `a`. The second reverses `a` so `a` is now `[30, 20, 10]`. The third statement adds `0` on to the end of `a`. The final expression `a` evaluates to `[30, 20, 10, 0]`.