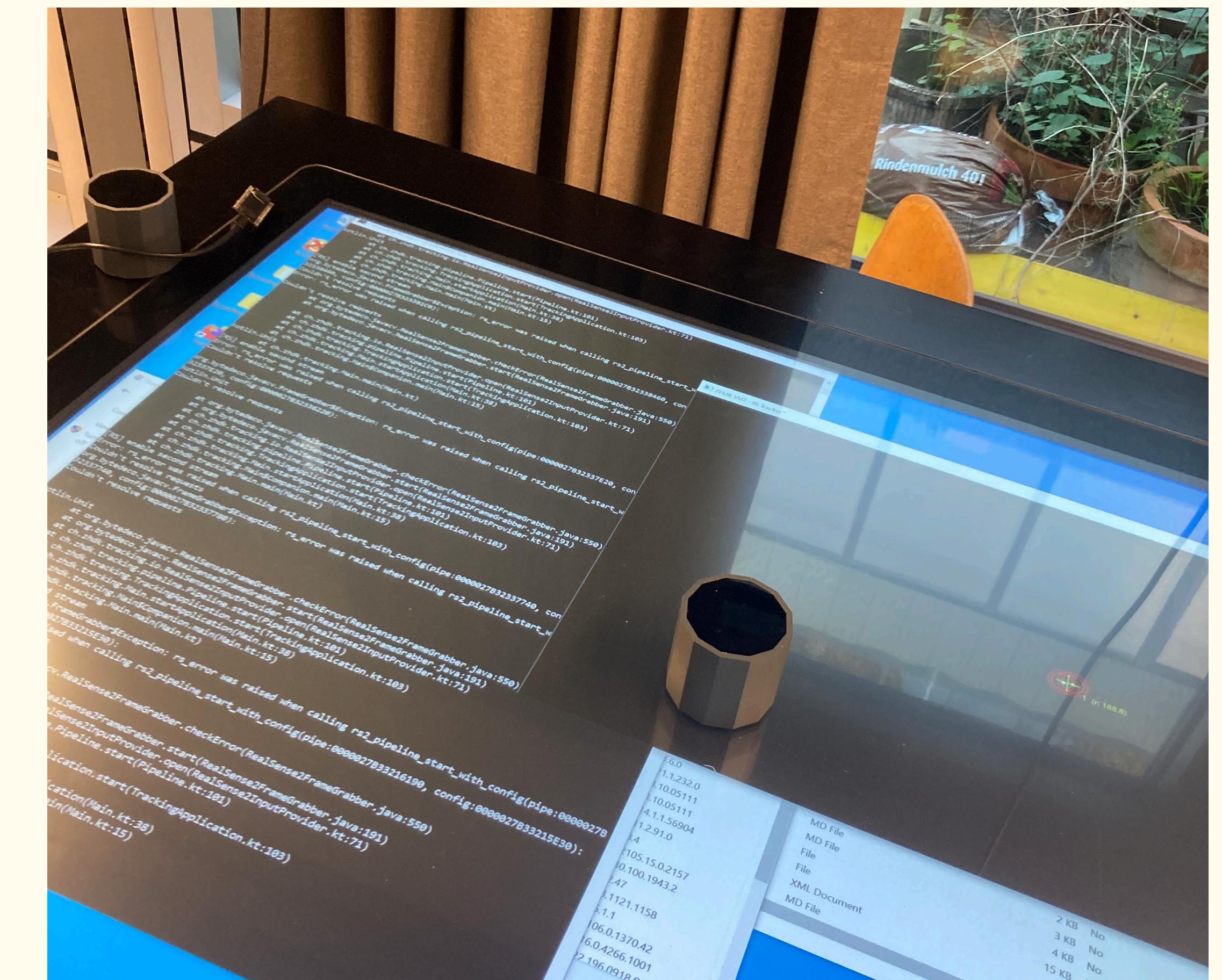


# Touch Table Emulator

- Touch table and infrared tracking devices
- Emulator on GitHub to work with 3D-tokens



# Touch Table Emulator

**Languages**

Language	Percentage
Kotlin	62.6%
Java	31.0%
C++	4.8%
Prolog	0.8%
CMake	0.5%
C	0.1%
Other	0.2%

**Touch Table IR Tracking System** build passing

This project aims to create an infrared tracking system for tactile tables.

**Tracking application detecting a single tactile object.**

**Quick Install**

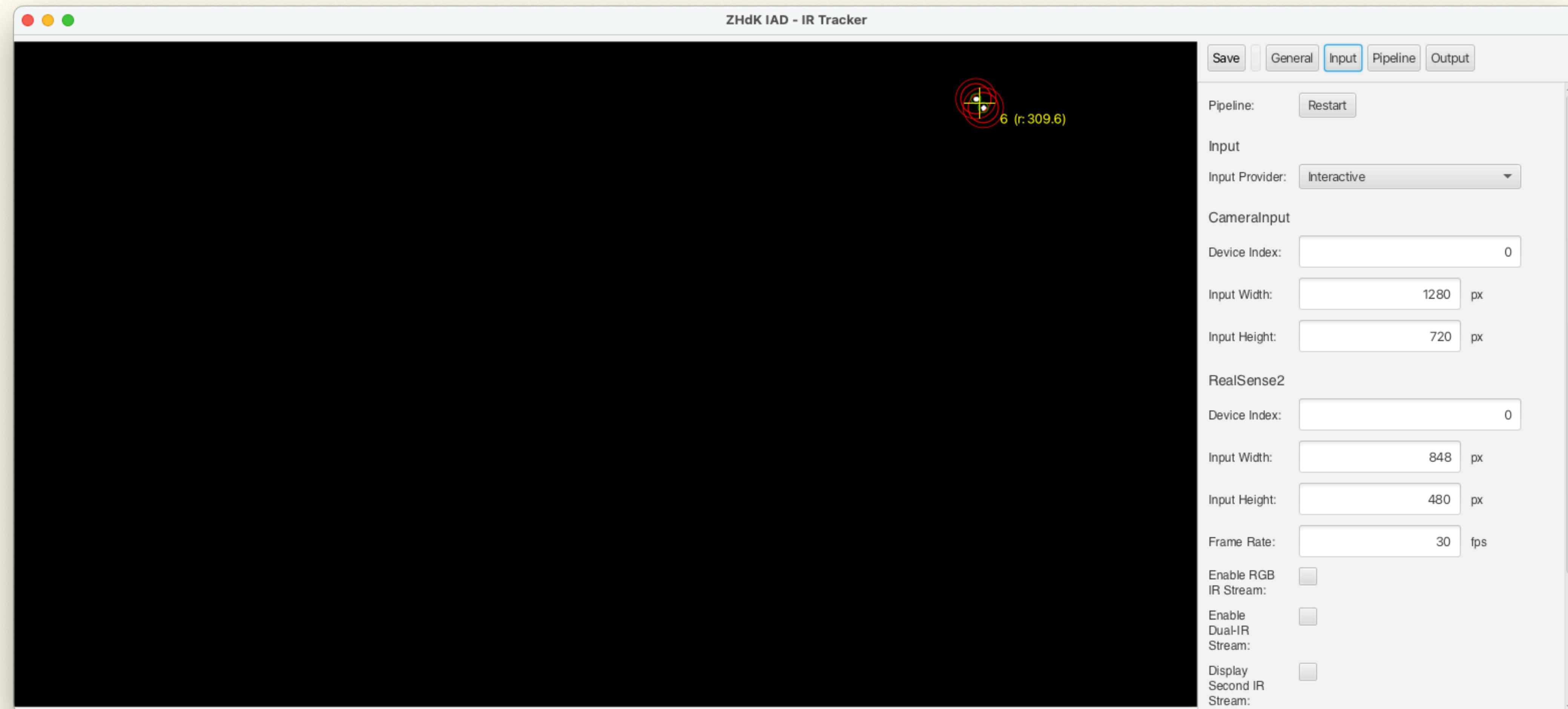
**MacOS**

Just copy and past this snippet in the terminal to quickly install and start the system (only MacOS).

```
echo IR Tracking Quickstart
if [[ $? != 0 ]] ; then
    echo installing Homebrew
    ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/i
else
    echo Homebrew already installed
fi
```

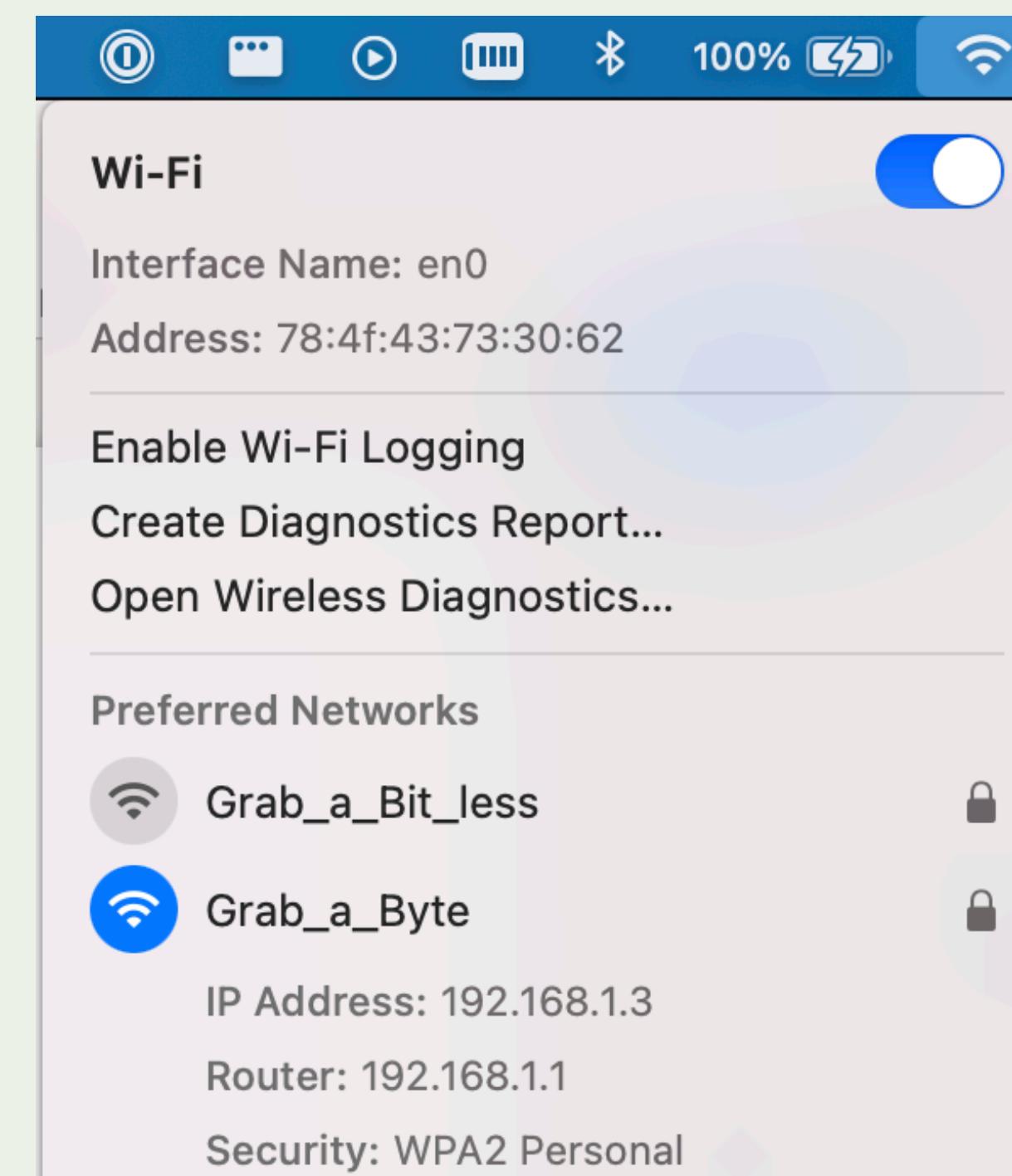
- <https://github.com/IAD-ZHDK/touchtable-infrared-tracker>

# Touch Table Emulator



# How to debug on touch devices?

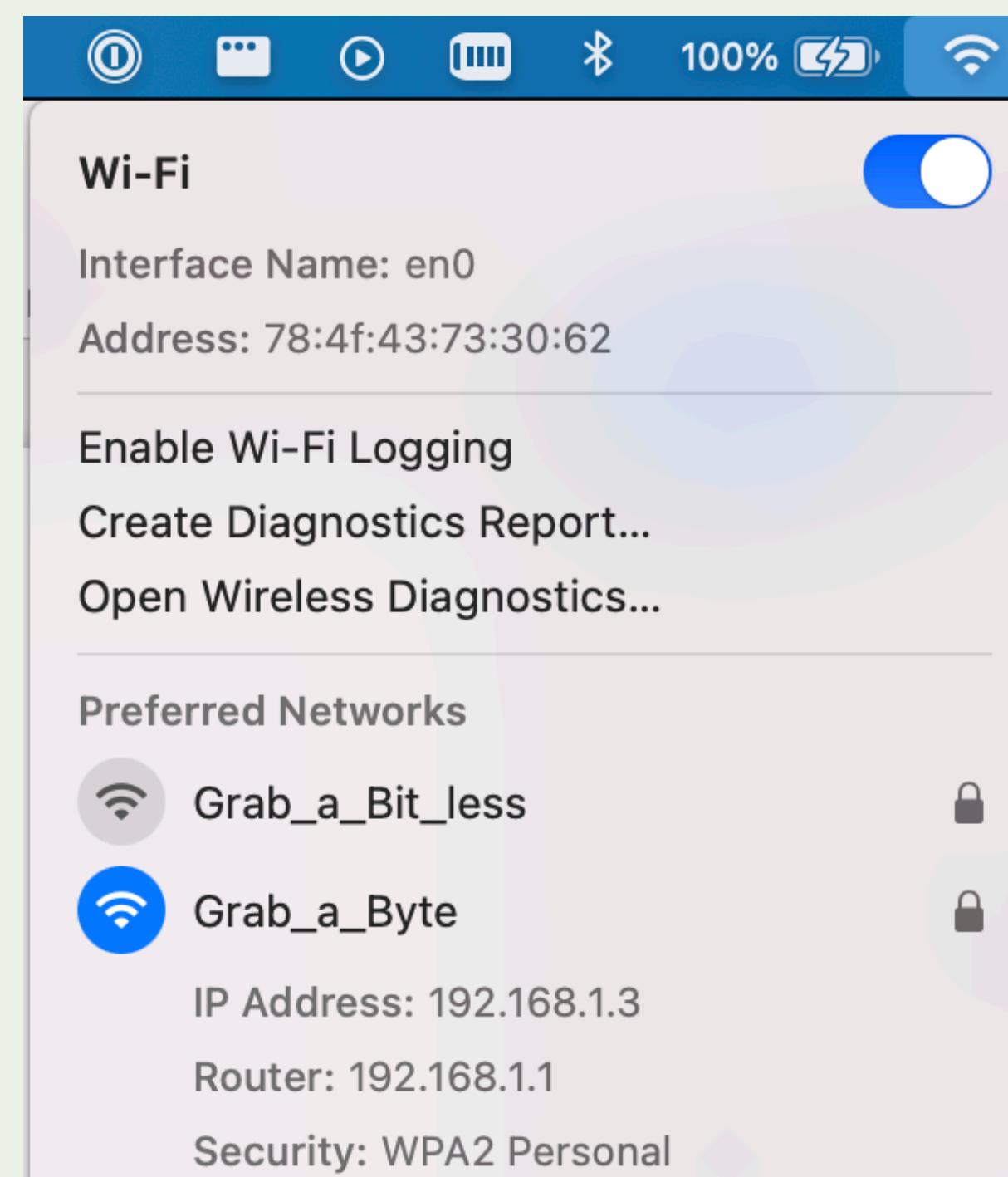
- Online web page... but not easy to debug
- Local development (e.g. Live Server)
  - Make sure your touch device and your computer are using the same Wi-Fi
  - Copy computer's IP to your touch device
  - Make sure you set the port & path correctly



Two screenshots of a Firefox Developer Edition window on a Mac, illustrating the setup for local development on a touch device (iPad).  
The top screenshot, labeled 'Computer', shows the Firefox toolbar with tabs for 'DETM' (playing), 'An intr...', 'd3/d3...', 'd3/API...', and 'Introduction'. The address bar shows the URL '127.0.0.1:5500/public/examples/hammerjs/'. The main content area displays the message 'tap gesture detected.' and the status bar shows the time as 12:26 and the date as Thu 4 Nov.  
The bottom screenshot, labeled 'iPad', shows the same Firefox window but with a red oval highlighting the address bar. The address bar now shows the URL '192.168.1.3:5500/public/examples/hammerjs/'. The main content area also displays 'tap gesture detected.' and the status bar shows the time as 12:26 and the date as Thu 4 Nov. This demonstrates how the IP address of the local machine (192.168.1.3) has been copied from the computer's browser and pasted into the iPad's browser to facilitate local development.

# ... and @ ZHdK?

- ngrok for the win
- Allows to set a “tunnel” to your localhost, serving it on publicly available url.
- requires shell skills (Terminal.app)



Computer

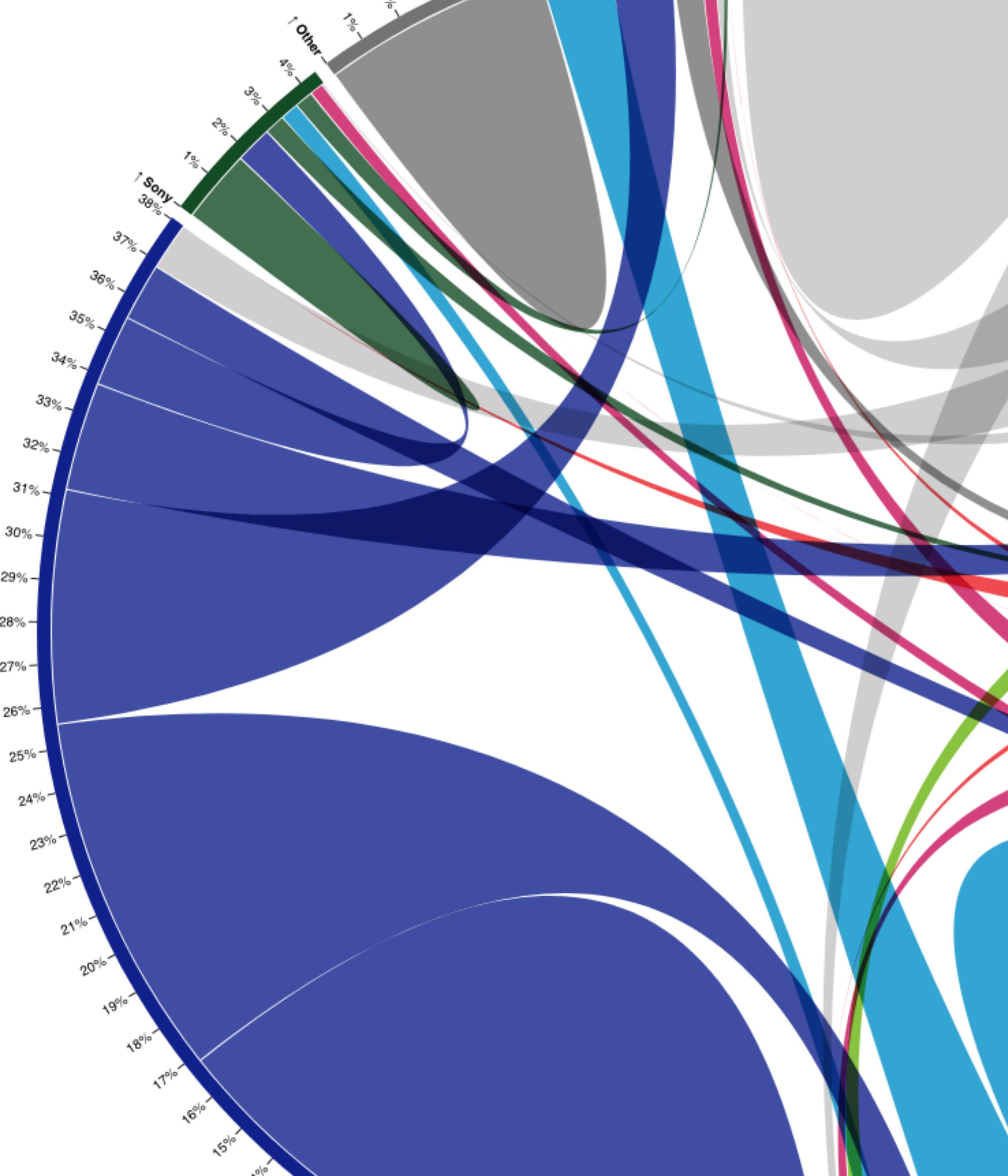
A screenshot of the Firefox Developer Edition browser on a Mac. The address bar shows the URL 127.0.0.1:5500/public/examples/hammerjs/. The page content area displays the message "tap gesture detected." Below the browser window, a status bar shows the IP address 192.168.1.3:5500/public/examples/hammerjs/. A red circle highlights the URL in the address bar. The iPad screen below shows the same content.

iPad

A screenshot of the Firefox Developer Edition browser on an iPad. The address bar shows the URL 192.168.1.3:5500/public/examples/hammerjs/. The page content area displays the message "tap gesture detected." Below the browser window, a status bar shows the IP address 192.168.1.3:5500/public/examples/hammerjs/. A red circle highlights the URL in the address bar.

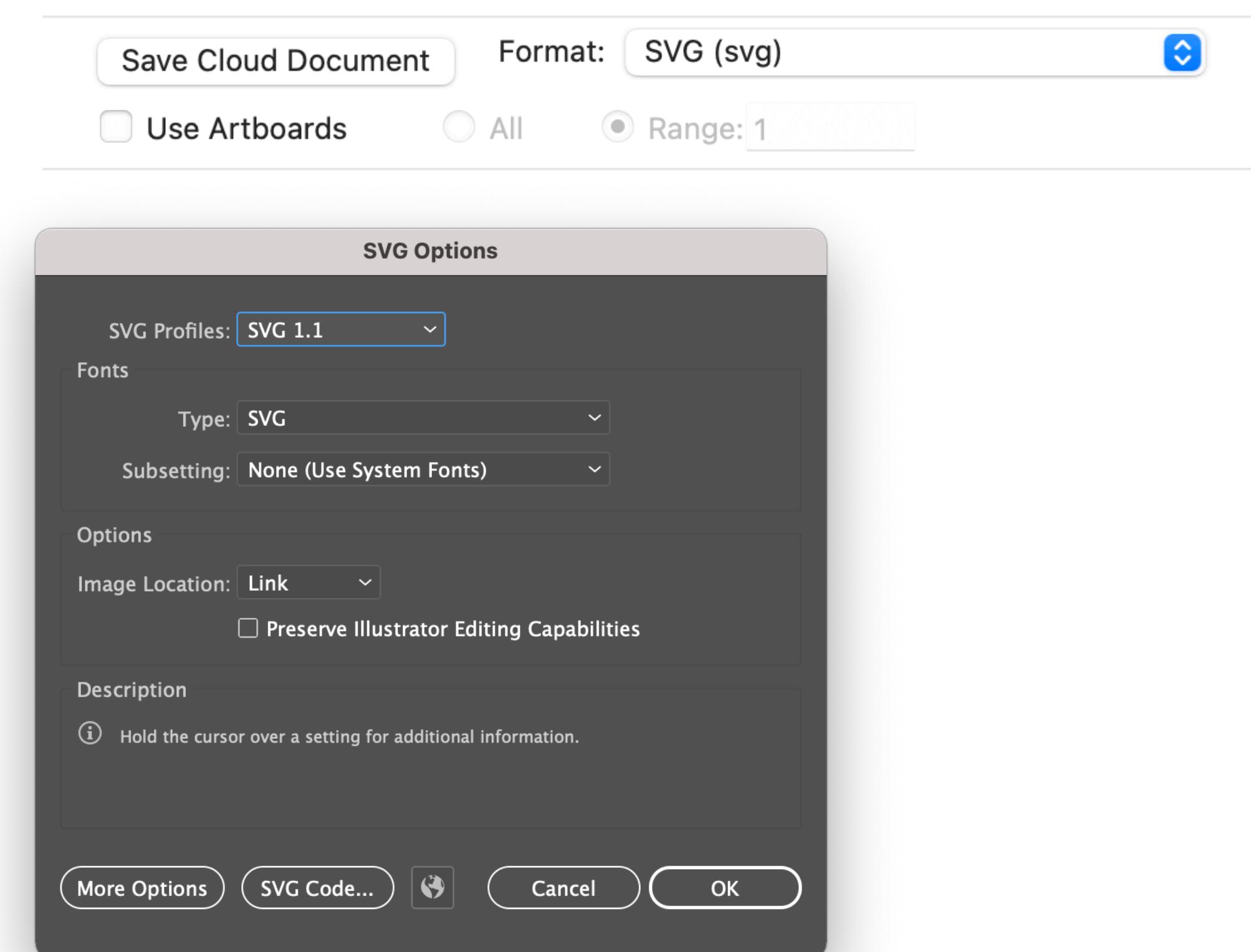
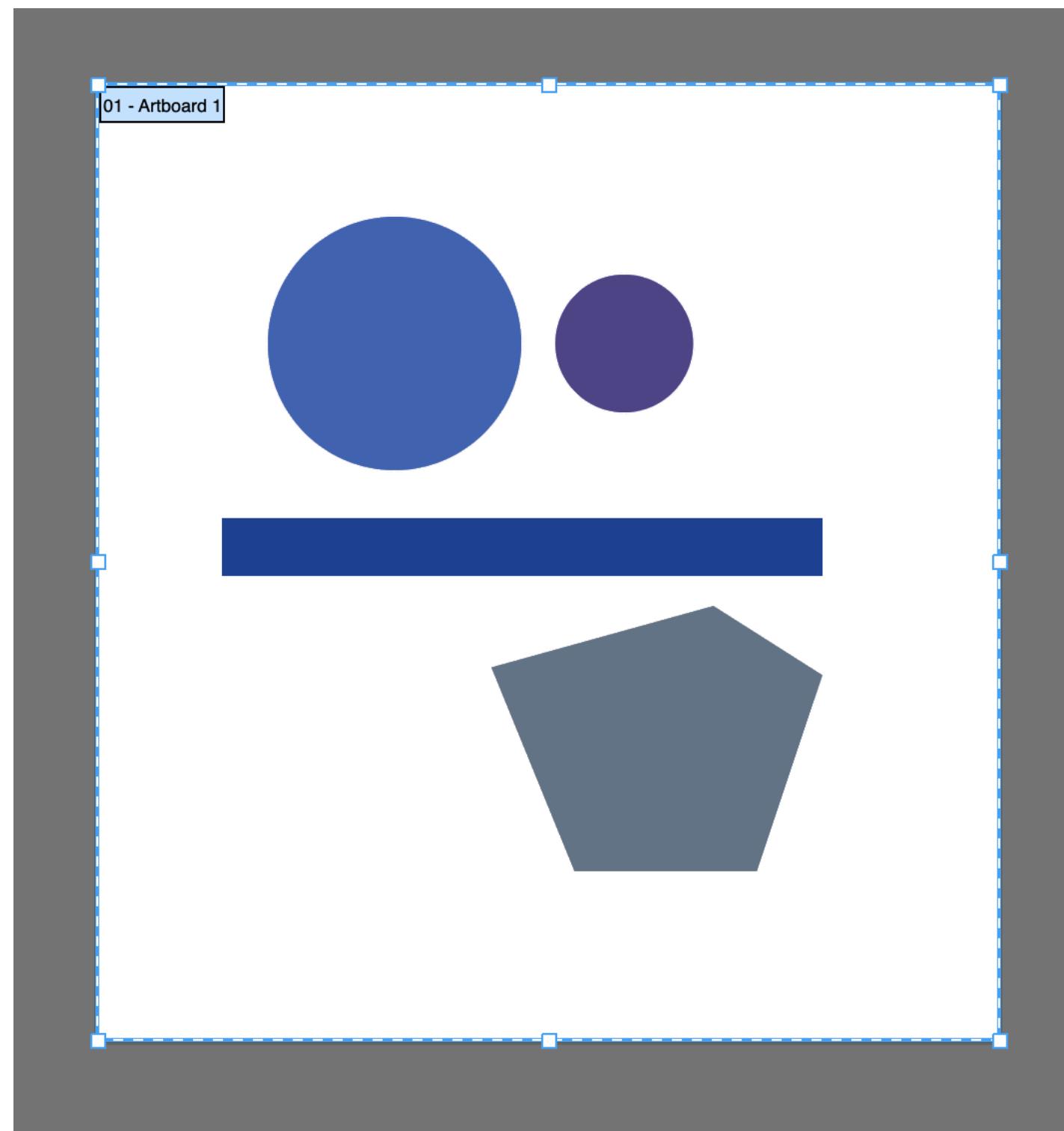
# d3

- «D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS... »
- <https://d3js.org/>



# $d3 \rightarrow SVG$

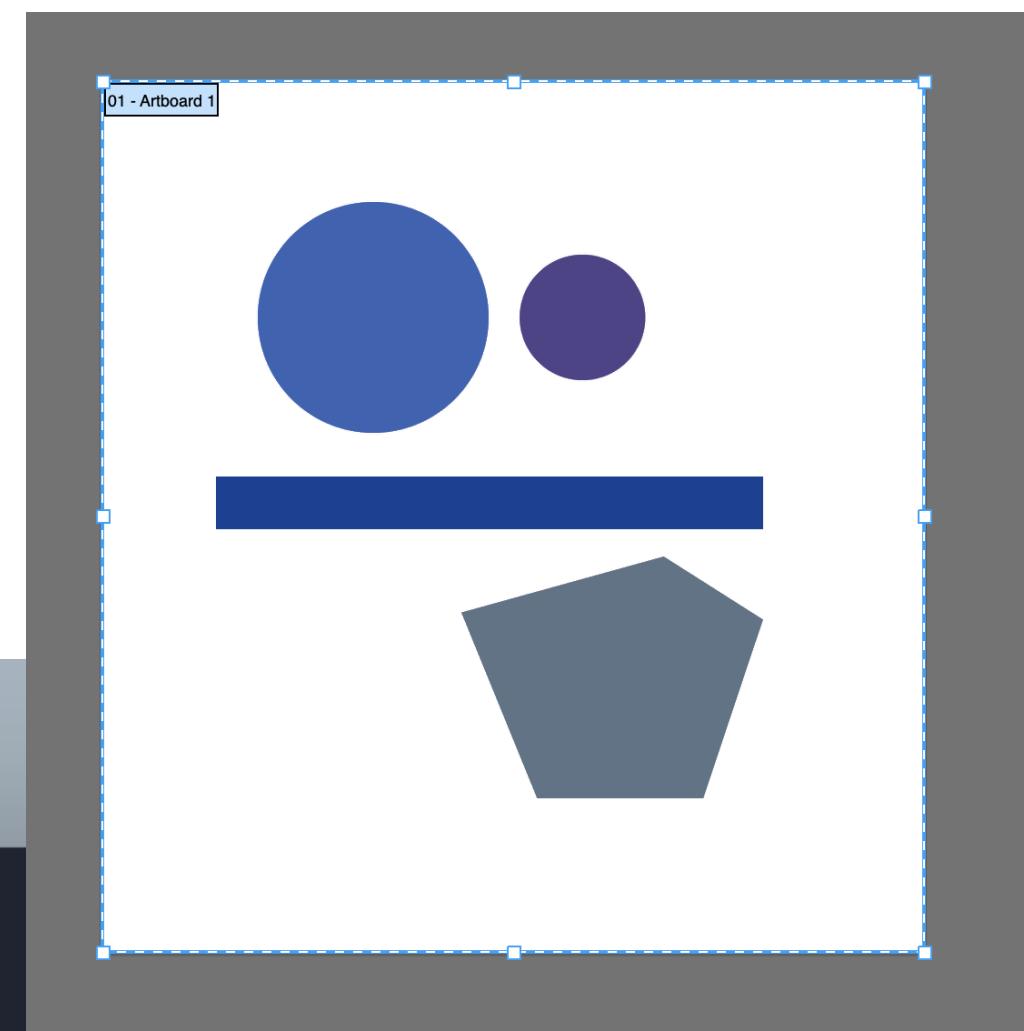
- SVG: Scalable Vector Graphics
- <https://developer.mozilla.org/en-US/docs/Web/SVG>



# SVG

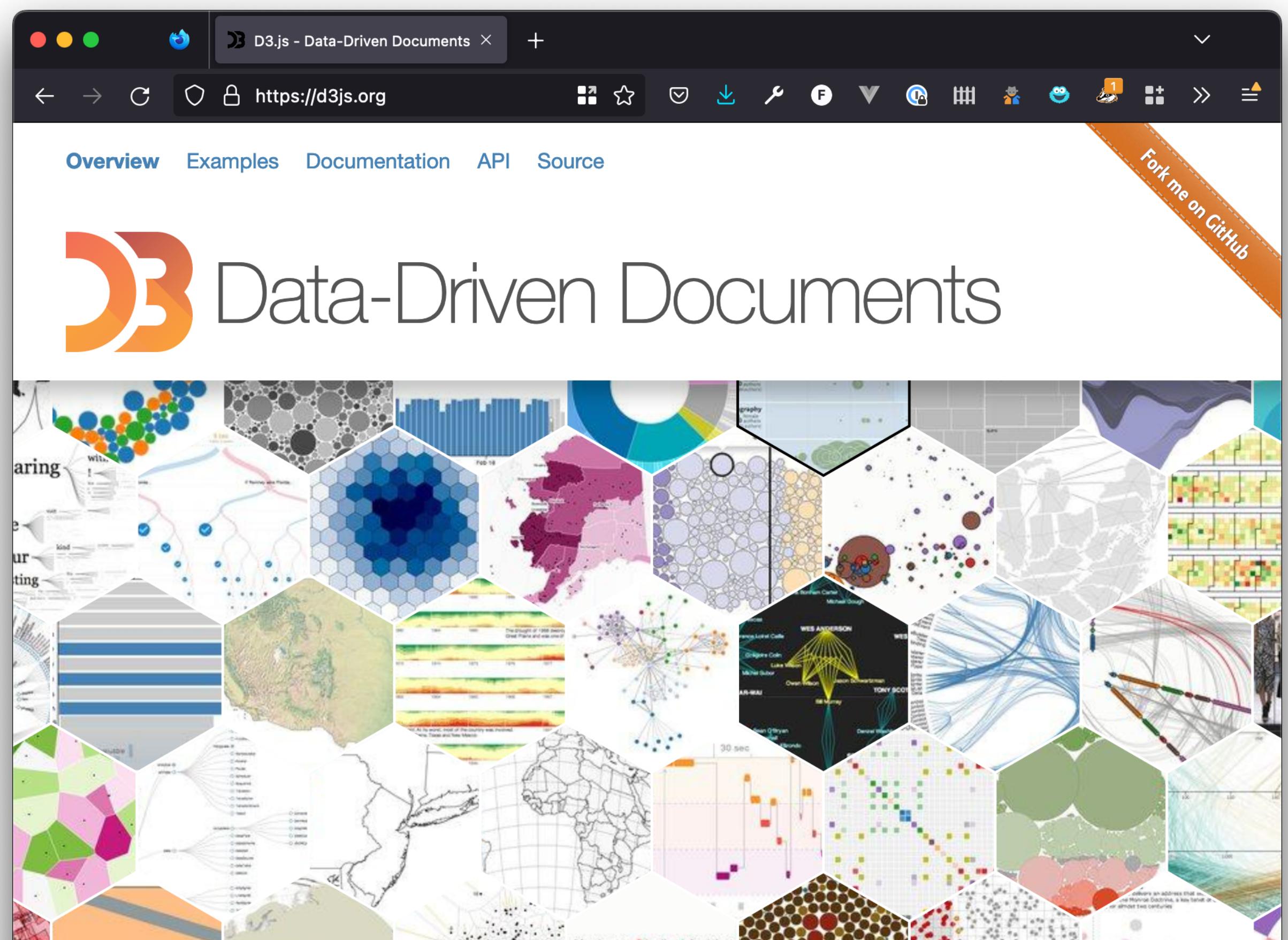


```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!-- Generator: Adobe Illustrator 25.0.1, SVG Export Plug-In . SVG Version: 6.00 Build 0) -->
3  <svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
4    viewBox="0 0 753.3 798.3" style="enable-background:new 0 0 753.3 798.3;" xml:space="preserve">
5  <style type="text/css">
6    .st0{fill:#4668B1;}
7    .st1{fill:#524589;}
8    .st2{fill:#214193;}
9    .st3{fill:#677284;}
10 </style>
11 <circle class="st0" cx="248.3" cy="216.7" r="106.7"/>
12 <circle class="st1" cx="440.2" cy="216.7" r="58.3"/>
13 <rect x="103.3" y="361.7" class="st2" width="503.3" height="50"/>
14 <polygon class="st3" points="328.1,486.7 514.8,435 606.7,493.3 551.5,658.3 398.1,658.3 "/>
15 </svg>
16
```



# D3: Intro

- <https://d3js.org/>
- Read introduction (10–15min)



The screenshot shows the homepage of the D3.js website. At the top, there's a navigation bar with links for Overview, Examples, Documentation, API, and Source. Below the navigation is the D3.js logo and the text "Data-Driven Documents". A large, colorful collage of various data visualizations is centered on the page, including maps, charts, and network graphs. In the top right corner of the main content area, there's a "Fork me on GitHub" button. At the bottom of the page, there's a yellow banner with the text "Like visualization and creative coding? Try interactive JavaScript notebooks in [Observable!](#)". Below this banner, there's a paragraph about what D3.js is and how it works, followed by links to various resources and social media.

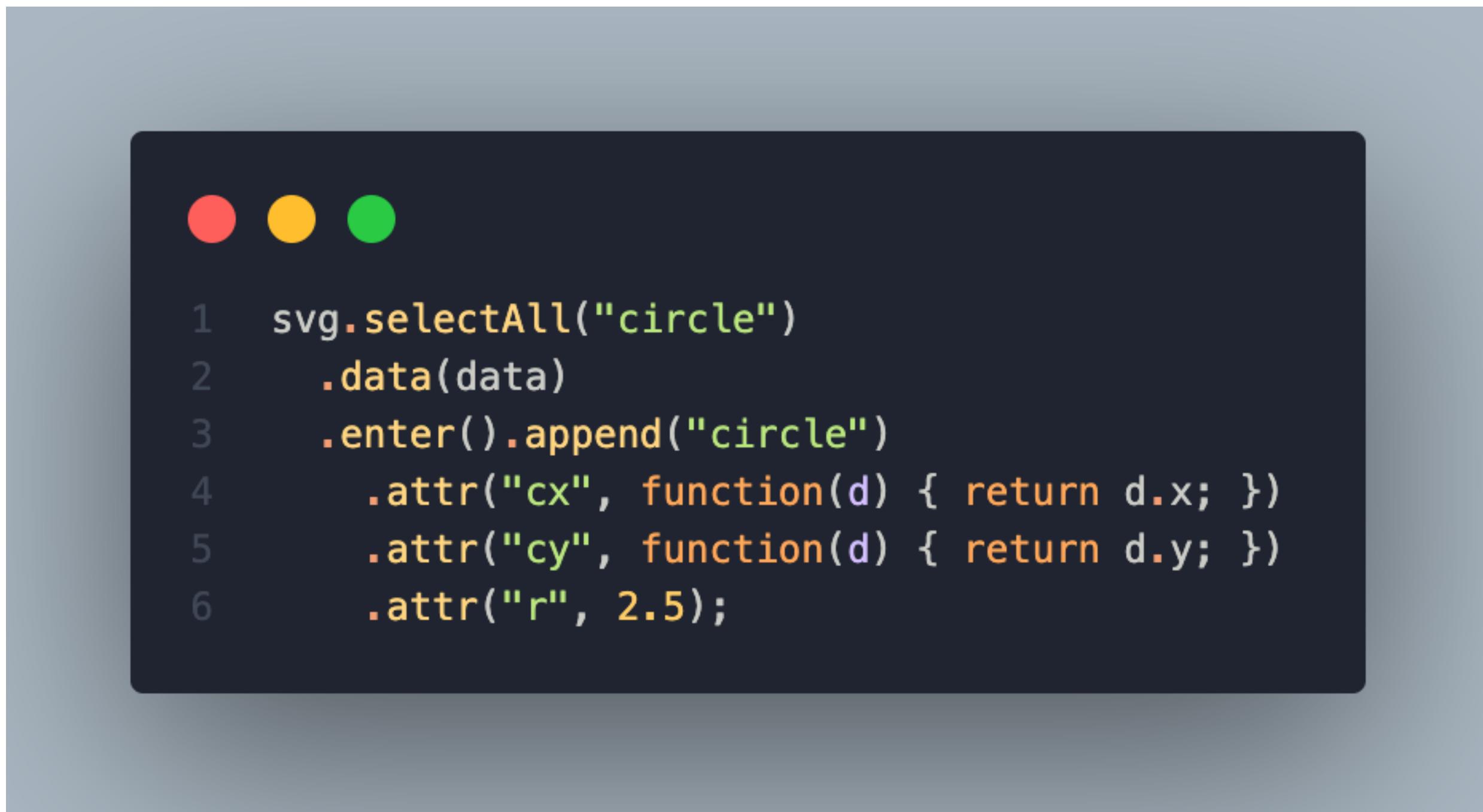
**D3.js** is a JavaScript library for manipulating documents based on data. **D3** helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

Download the latest version (7.6.1) here:

- [d3-7.6.1.tgz](#)

[See more examples](#)  
[Chat with the community](#)  
[Follow announcements](#)  
[Report a bug](#)  
[Ask for help](#)

# *D3: Basic principles*

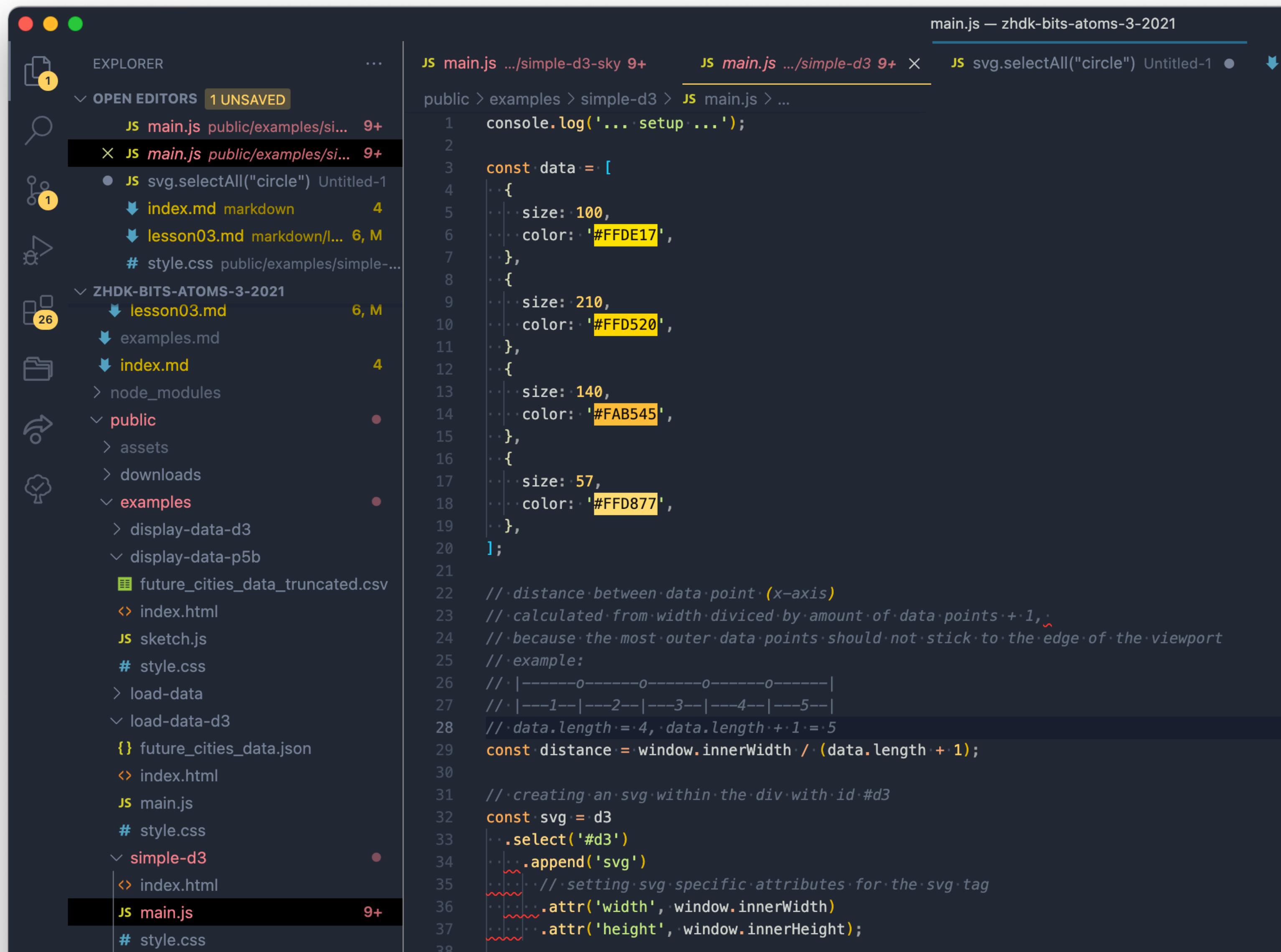


[https://www.d3-graph-gallery.com/intro\\_d3js.html#data](https://www.d3-graph-gallery.com/intro_d3js.html#data)  
D3 Docs/API: <https://github.com/d3/d3/blob/main/API.md>

Examples  
[https://observablehq.com/](https://observablehq.com/@d3/) respectively: <https://observablehq.com/@d3/streamgraph-transitions>

# D3: Example

- Examples from Email:
  - simple-d3
  - simple-d3-sky



The screenshot shows a dark-themed code editor interface. On the left is the Explorer sidebar, which lists several files and folders. In the main area, there are two tabs open in the editor:

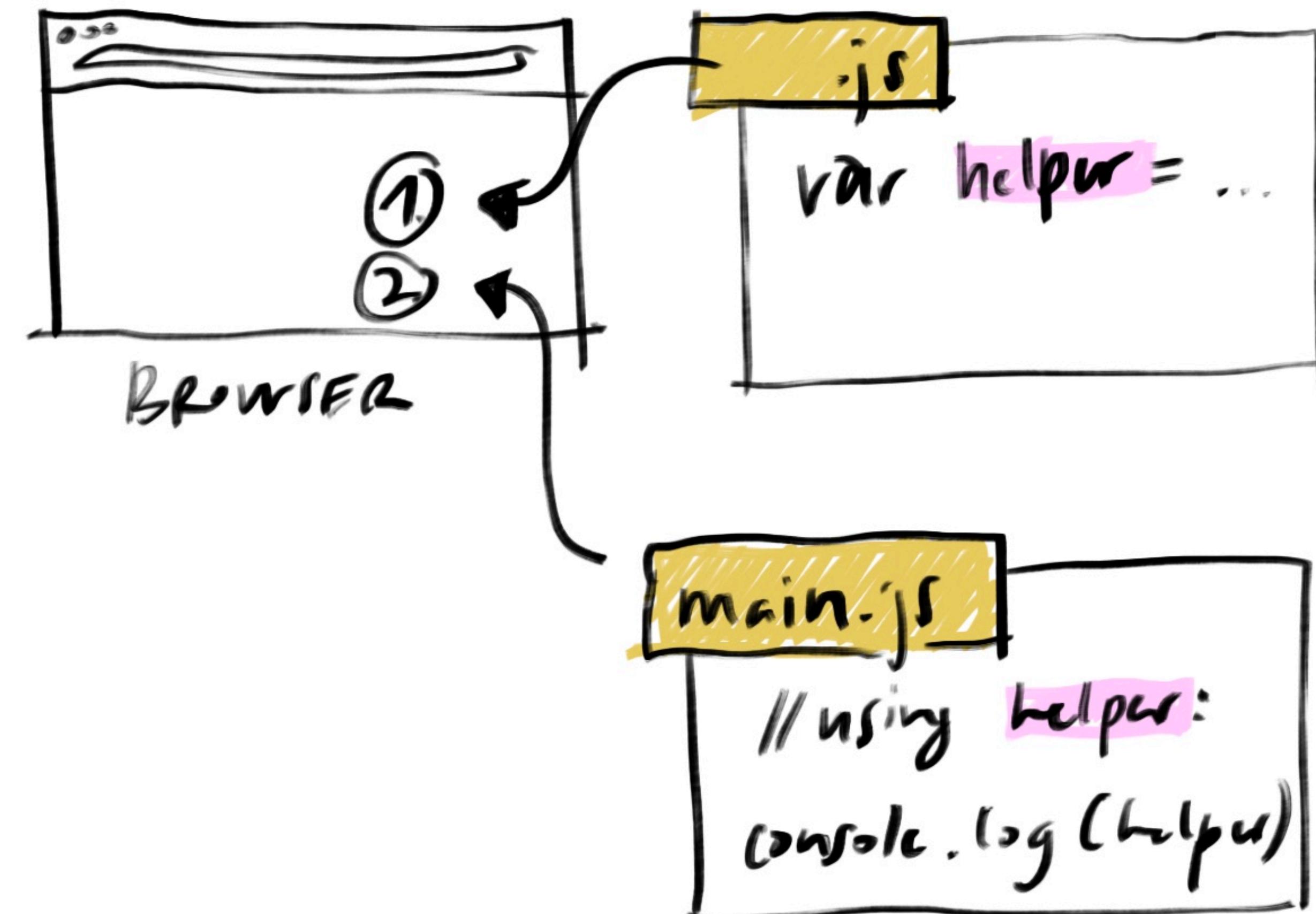
- JS main.js .../simple-d3-sky 9+**: This tab contains a snippet of JavaScript code for a sunburst chart. It defines a variable `data` with three nested arrays representing concentric rings. The innermost ring has a size of 100 and a color of #FFDE17. The middle ring has a size of 210 and a color of #FFD520. The outermost ring has a size of 140 and a color of #FAB545. The code also includes a call to `svg.selectAll("circle")`.
- JS main.js .../simple-d3 9+ ×**: This tab contains a snippet of JavaScript code for a sunburst chart. It uses the same data structure and styling as the first file, but includes additional comments explaining the calculation of distance between data points and the creation of an SVG element.

```

1 console.log('... setup ...');
2
3 const data = [
4   {
5     size: 100,
6     color: '#FFDE17',
7   },
8   {
9     size: 210,
10    color: '#FFD520',
11  },
12  {
13    size: 140,
14    color: '#FAB545',
15  },
16  {
17    size: 57,
18    color: '#FFD877',
19  },
20 ];
21
22 // distance between data point (x-axis)
23 // calculated from width divided by amount of data points + 1, -
24 // because the most outer data points should not stick to the edge of the viewport
25 // example:
26 // |-----o-----o-----o-----o-----|
27 // |---1--|---2--|---3--|---4--|---5--|
28 // data.length == 4, data.length + 1 == 5
29 const distance = window.innerWidth / (data.length + 1);
30
31 // creating an svg within the div with id #d3
32 const svg = d3
33   .select('#d3')
34   .append('svg')
35   // setting svg specific attributes for the svg tag
36   .attr('width', window.innerWidth)
37   .attr('height', window.innerHeight);
38

```

# d3: Under the hood



# d3: Under the hood



```
1 var d3 = {  
2   // ...  
3 };
```



```
1 // main.js  
2 // has access to d3 object  
3  
4 var d3Method = d3.select;  
5 console.log('d3Method: ', d3Method);
```

d3Method: ▶ function Un(t) ↵

main.js:9:9

# d3: Under the hood



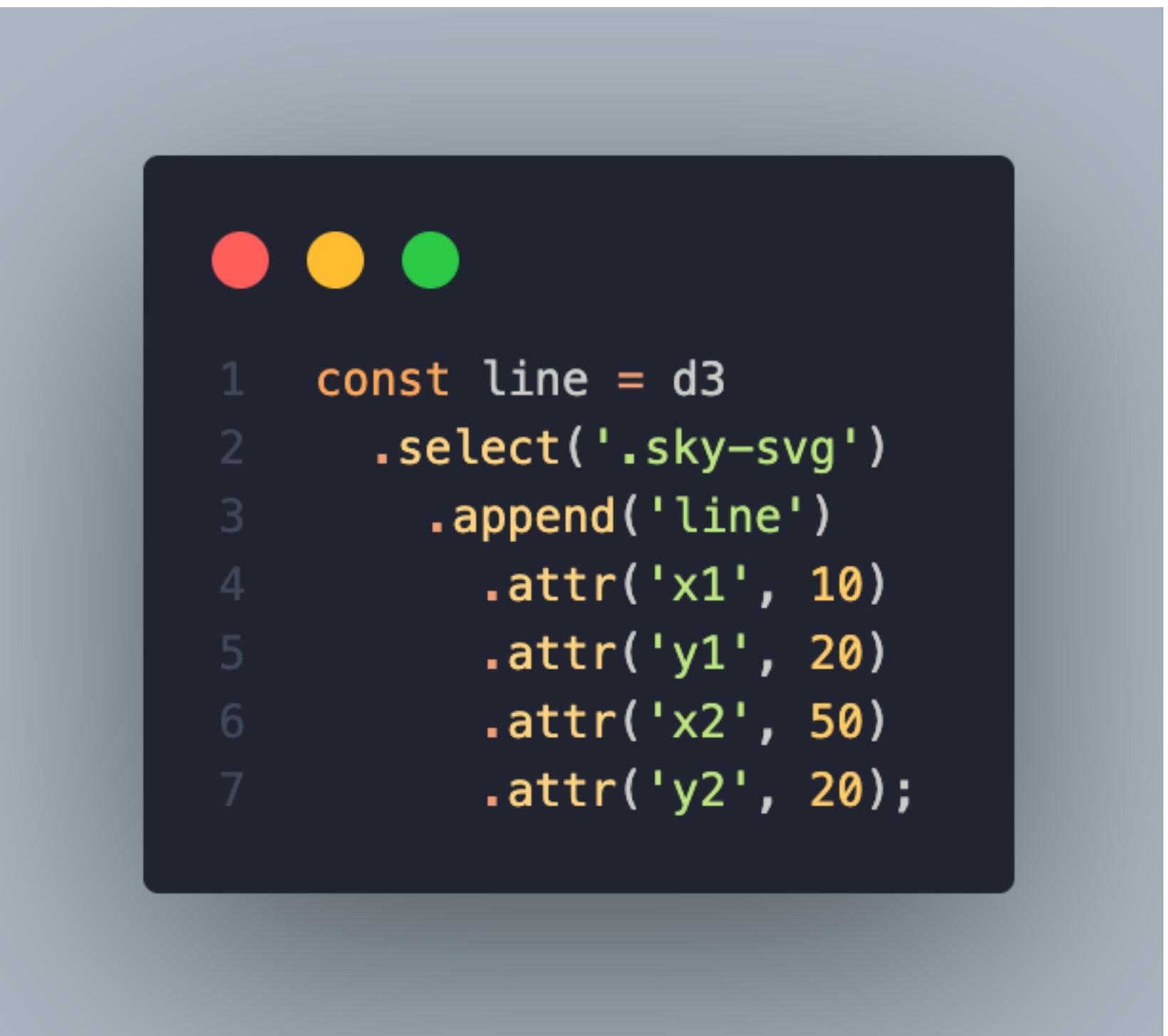
```
1 var d3 = {  
2   select: function(selector) {  
3     // do something (with selector)  
4     // return something  
5   },  
6   size: 100,  
7   author: 'mbostock'  
8 }
```



```
1 // main.js  
2 // can now make use of d3 method (function)  
3  
4 var mySelection = d3.select('#div');
```

# d3: Under the hood

- But what happens here?... →  
Methods chaining/  
function chaining



```
1 const line = d3
2   .select('.sky-svg')
3     .append('line')
4       .attr('x1', 10)
5         .attr('y1', 20)
6           .attr('x2', 50)
7             .attr('y2', 20);
```

# d3: Under the hood



```
1 // jQuery
2 var selectedDiv = $("#myDiv");
3
4 selectedDiv.css('color: red');
5 selectedDiv.height('100px');
6 selectedDiv.width('100px');
```

Is the same as:



```
1 // jQuery
2 var selectedDiv = $("#myDiv");
3
4 selectedDiv.css('color: red').height('100px').width('100px');
```

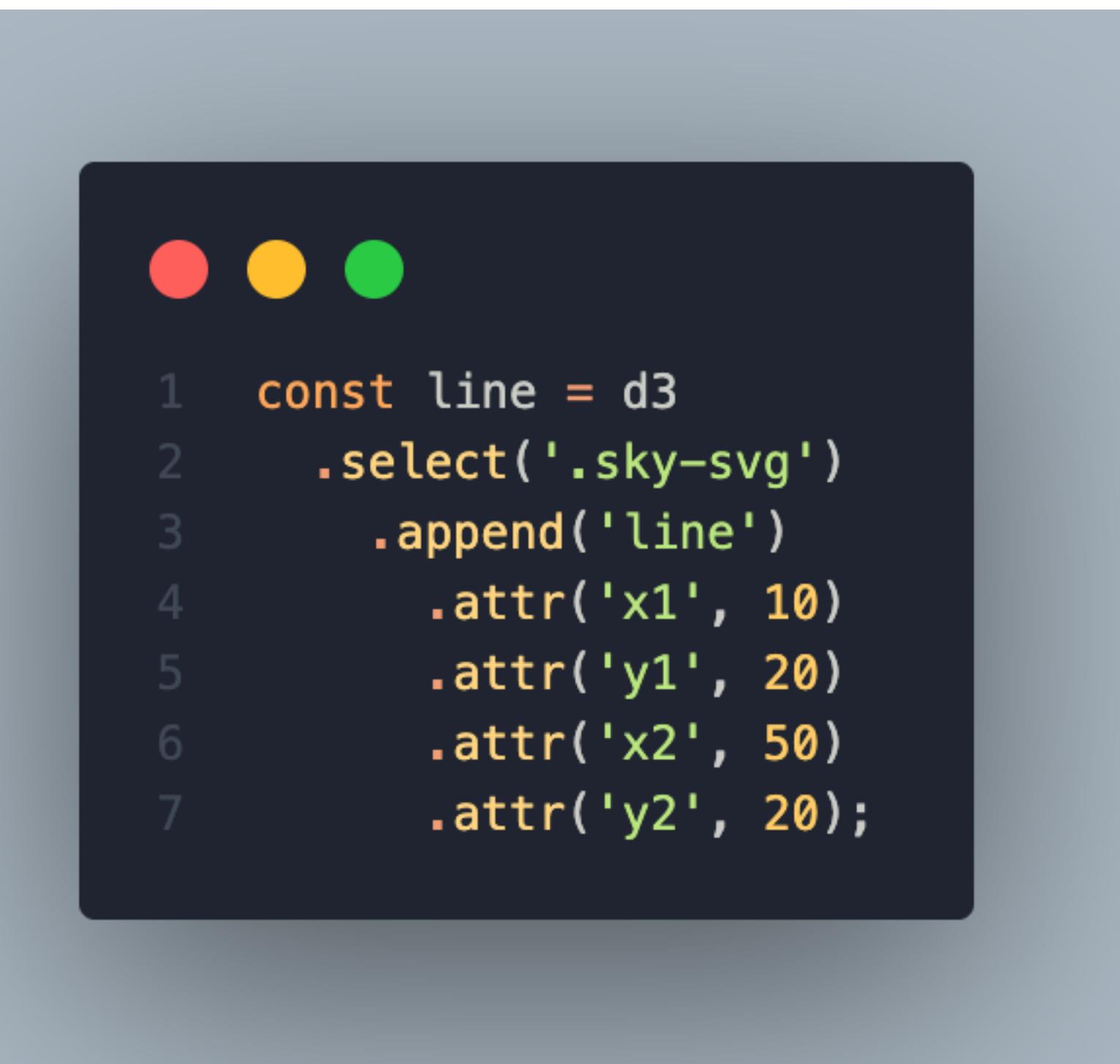
and also the same as:



```
1 // jQuery
2 var selectedDiv = $("#myDiv");
3
4 selectedDiv.css('color: red')
5   .height('100px')
6   .width('100px');
```

# *d3: Under the hood*

- ... we use a lot of chained d3 methods. Therefore we only use a semicolon (;) at the end.



```
1 const line = d3
2   .select('.sky-svg')
3     .append('line')
4       .attr('x1', 10)
5         .attr('y1', 20)
6           .attr('x2', 50)
7             .attr('y2', 20);
```

# *Exercise*

- Work on your homework.

# d3: Under the hood

- What about these functions  
in there?



```
1 const mySVG = d3.select('#d3-svg')
2   .selectAll('whatever')
3   .data(data)
4   .enter()
5   .append('circle')
6     .attr('cx', function(d, currentIndex) { return currentIndex * distance + distance })
7     .attr('cy', 300)
8     .attr('r', function(d) { return d.size / 2 })
9     .attr('fill', function(d) { return d.color });
```

# d3: Under the hood

First we call the method attr() from d3

```
.attr('cx', function(d, currentIndex) { return currentIndex * distance + distance })
```

What does this function look like?

→ [https://github.com/d3/d3-selection/blob/v3.0.0/README.md#selection\\_attr](https://github.com/d3/d3-selection/blob/v3.0.0/README.md#selection_attr)

→ <https://github.com/d3/d3-selection/blob/main/src/selection/attr.js>

## Modifying Elements

- `selection.attr` - get or set an attribute.

# `selection.attr(name[, value])` · Source

If a `value` is specified, sets the attribute with the specified `name` to the specified `value` on the selected elements and returns this selection. If the `value` is a constant, all elements are given the same attribute value; otherwise, if the `value` is a function, it is evaluated for each selected element, in order, being passed the current datum (`d`), the current index (`i`), and the current group (`nodes`), with `this` as the current DOM element (`nodes[i]`). The function's return value is then used to set each element's attribute. A null value will remove the specified attribute.

If a `value` is not specified, returns the current value of the specified attribute for the first (non-null) element in the selection. This is generally useful only if you know that the selection contains exactly one element.

The specified `name` may have a namespace prefix, such as `xlink:href` to specify the `href` attribute in the XLink namespace. See [namespaces](#) for the map of supported namespaces; additional namespaces can be registered by adding to the map.

```

43  export default function(name, value) {
44    var fullname = namespace(name);
45
46    if (arguments.length < 2) {
47      var node = this.node();
48      return fullname.local
49        ? node.getAttributeNS(fullname.space, fullname.local)
50        : node.getAttribute(fullname);
51    }
52
53    return this.each((value == null
54      ? (fullname.local ? attrRemoveNS : attrRemove) :
55      (fullname.local ? attrFunctionNS : attrFunction)
56      : (fullname.local ? attrConstantNS : attrConstant)))(fullname, value));
57  }
```

The diagram illustrates the flow of the code. An arrow points from the line '# selection.attr(name[, value]) · Source' to the start of the source code block. Another arrow points from the line '• selection.attr - get or set an attribute.' to the explanatory text about setting attributes. Three question marks are placed over the source code block, indicating areas of interest or confusion: one above the 'if (arguments.length < 2)' block, one above the 'return this.each' block, and one above the final closing brace of the function.

# d3: Under the hood

Functions as a parameter?

```
.attr('cx', function(d, currentIndex) { return currentIndex * distance + distance })
```



```

43 export default function(name, value) {
44   var fullname = namespace(name);
45
46   if (arguments.length < 2) {
47     var node = this.node();
48     return fullname.local
49       ? node.getAttributeNS(fullname.space, fullname.local)
50       : node.getAttribute(fullname);
51   }
52
53   return this.each((value == null
54     ? (fullname.local ? attrRemoveNS : attrRemove) : (typeof value === "function"
55     ? (fullname.local ? attrFunctionNS : attrFunction)
56     : (fullname.local ? attrConstantNS : attrConstant)))(fullname, value));
57 }
```

Very complicated and short code to determine,  
what is passed to the "each" function...  
Don't worry!

# d3: Under the hood

```
27  function attrFunction(name, value) {  
28    return function() {  
29      var v = value.apply(this, arguments);  
30      if (v == null) this.removeAttribute(name);  
31      else this.setAttribute(name, v);  
32    };  
33  }
```

```
.attr('cy', 300)
```

```
.attr('cx', function(d, currentIndex) { return currentIndex * distance + distance })
```

```
.attrFunction(name, value)
```

```
.attrFunction('cy', 300)
```

```
.attrFunction('cx', function(d, currentIndex) { return currentIndex * distance + distance }) )
```

# d3: Under the hood

```

27  function attrFunction(name, value) {
28    return function() {
29      var v = value.apply(this, arguments);
30      if (v == null) this.removeAttribute(name);
31      else this.setAttribute(name, v);
32    };
33  }

```

With `apply()` the function, which is passed in as the variable 'value', is executed.

```

const array = ['a', 'b'];
const elements = [0, 1, 2];
array.push.apply(array, elements);
console.info(array); // ["a", "b", 0, 1, 2]

```

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Function/apply](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply)

```
var v = value.apply(this, function(d, currentIndex) { return currentIndex * distance + distance }) )
```



This function is executed and we take the return value 'v' to set the attribute:



```
1 this.setAttribute(name, v);
```

<https://developer.mozilla.org/en-US/docs/Web/API/Element/setAttribute>

# d3: Under the hood

Digging even deeper... → this.each

```
return this.each((value == null
  ? (fullname.local ? attrRemoveNS : attrRemove) : (typeof value === "function"
  ? (fullname.local ? attrFunctionNS : attrFunction)
  : (fullname.local ? attrConstantNS : attrConstant)))(fullname, value));
```

main ▾

[d3-selection](#) / [src](#) / [selection](#) / [each.js](#) / <> Jump to ▾



**mbostock** Fix #57 - rename \_nodes to \_groups.

1 contributor

<https://github.com/d3/d3-selection/blob/main/src/selection/each.js>

10 lines (8 sloc) | 289 Bytes

```
1  export default function(callback) {
2
3    for (var groups = this._groups, j = 0, m = groups.length; j < m; ++j) {
4      for (var group = groups[j], i = 0, n = group.length, node; i < n; ++i) {
5        if (node = group[i]) callback.call(node, node._data_, i, group);
6      }
7    }
8
9    return this;
10 }
```

# d3: Under the hood

[d3-selection / src / selection / each.js](#) / [Jump to ▾](#)

stock Fix #57 - rename \_nodes to \_groups.

ributor

(8 sloc) | 289 Bytes

```
import default function(callback) {
  for (var groups = this._groups, j = 0, m = groups.length; j < m; ++j) {
    for (var group = groups[j], i = 0, n = group.length, node; i < n; ++i) {
      if (node = group[i]) callback.call(node, node.__data__, i, group);
    }
  }
  return this;
}
```

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Function/call](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call)

callback.call() executes our anonymous function we passed into attr()

```
function(d, currentIndex) { return currentIndex * distance + distance }
```

with

```
1   callback.call(node, node.__data__, i, group);
```

This means it executes our function and passes in:

1. node.\_\_data\_\_
2. i (index)
3. group

# d3: Under the hood

```
1 callback.call(node, node.__data__, i, group);
```

This means it executes our function and passes in:

1. node.\_\_data\_\_
2. i (index)
3. group



```
1 .attr('cx', function(nodeData, index, group) {  
2   console.log('nodeData: ', nodeData);  
3   console.log('index: ', index);  
4   console.log('group: ', group);  
5   return index * distance + distance;  
6 })
```

```
nodeData: ▶ Object { size: 100, color: "#FFDE17" }  
index: 0  
group:  
▶ Array(4) [ circle, circle, circle, circle ]  
nodeData: ▶ Object { size: 210, color: "#FFD520" }  
index: 1  
group:  
▶ Array(4) [ circle, circle, circle, circle ]  
nodeData: ▶ Object { size: 140, color: "#FAB545" }  
index: 2
```

# d3: Under the hood

- So this is why we can use relatively plain javascript, calling d3 methods and passing in anonymous (=without a name) functions as parameters...



```
1 const mySVG = d3.select('#d3-svg')
2   .selectAll('whatever')
3   .data(data)
4   .enter()
5   .append('circle')
6     .attr('cx', function(d, currentIndex) { return currentIndex * distance + distance })
7     .attr('cy', 300)
8     .attr('r', function(d) { return d.size / 2 })
9     .attr('fill', function(d) { return d.color });
```