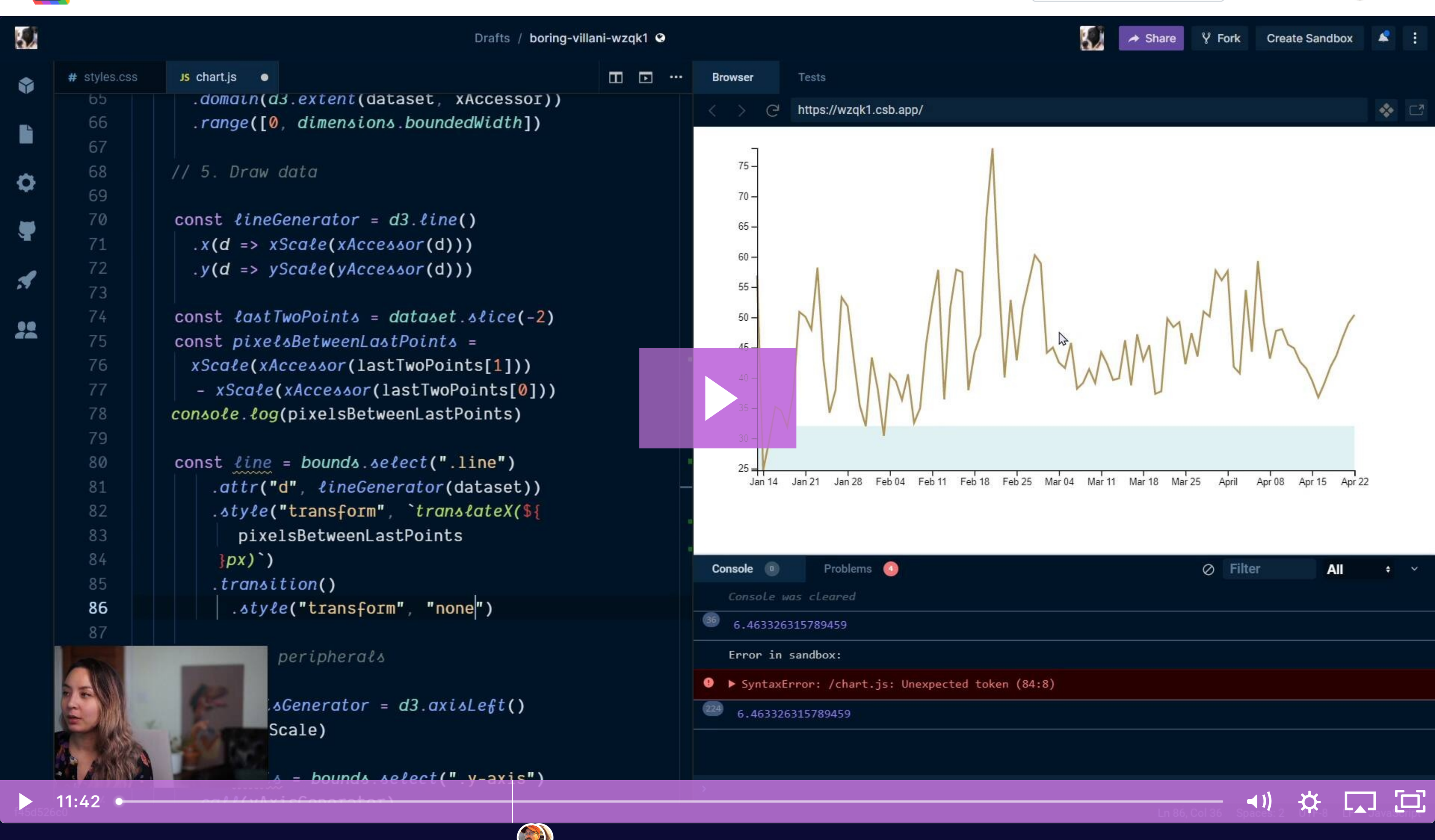


Fullstack D3 Masterclass

WELCOME:
INTRODUCTION
INTRODUCTION: Start Here

MODULE 0:
MAKING YOUR FIRST CHART
LESSON 1: (1:43) Making Your First Chart
LESSON 2: (5:51) Loading the weather dataset
LESSON 3: (8:35) Setting up our line chart
LESSON 4: (5:15) Drawing our chart
LESSON 5: (3:24) Creating our workspace
LESSON 6: (4:28) Adding an SVG element
LESSON 7: (4:42) Creating our bounding box
LESSON 8: (10:06) Creating our scales
LESSON 9: (6:27) Drawing the line
LESSON 10: (5:08) Drawing the axes
LESSON 11: Week 1: Exercise
APPENDIX 1: Datasets

MODULE 1:
MAKING A SCATTERPLOT
LESSON 1: (1:28) Making a Scatterplot
LESSON 2: (2:03) Steps in drawing any chart
LESSON 3: (1:57) Step 1: Access data
LESSON 4: (7:24) Step 2: Create chart dimensions
LESSON 5: (2:59) Step 3: Draw canvas
LESSON 6: (4:20) Step 4: Create scales
LESSON 7: (15:27) Step 5: Draw data
LESSON 8: (11:29) Step 6: Draw peripherals
LESSON 9: (1:55) Looking at our chart
LESSON 10: (5:47) Extra credit: adding a color scale
LESSON 11: Week 2: Exercise

MODULE 2:
MAKING A BAR CHART
LESSON 1: (3:31) Making a Bar Chart
LESSON 2: (2:03) Access data
LESSON 3: (2:49) Create dimensions
LESSON 4: (2:25) Draw canvas
LESSON 5: (10:25) Create scales
LESSON 6: (8:35) Draw data
LESSON 7: (7:11) Adding Labels
LESSON 8: (14:28) Draw peripherals
LESSON 9: (8:58) Extra credit
LESSON 10: (13:13) Accessibility
LESSON 11: Week 3: Exercise

MODULE 3:
ANIMATIONS AND TRANSITIONS
LESSON 1: (3:00) Animations and Transitions
LESSON 2: (6:47) CSS transitions
LESSON 3: (8:14) CSS transitions with a chart
LESSON 4: (22:12) d3.transition
LESSON 5: (11:41) Lines
LESSON 6: Week 4: Exercise

MODULE 4:
INTERACTIONS
LESSON 1: (0:55) Interactions
LESSON 2: (7:09) d3 events
LESSON 3: (3:09) Destroying d3 event listeners
LESSON 4: (18:51) Bar chart
LESSON 5: (25:04) Scatter plot
LESSON 6: (24:19) Line chart
LESSON 7: Week 5: Exercise

MODULE 5:
DATA VISUALIZATION BASICS
LESSON 1: (4:17) Data Visualization Basics
LESSON 2: (4:52) Types of data
LESSON 3: (4:49) Ways to visualize a metric
LESSON 4: (2:58) Chart design
LESSON 5: (7:43) Example redesign
LESSON 6: (5:26) Color scales
LESSON 7: (4:48) Custom color scales
LESSON 8: (9:33) Color spaces
LESSON 9: (4:36) Color tips
LESSON 10: Week 6: Exercise

MODULE 6:
ADVANCED EXAMPLE
LESSON 1: (2:23) Radar Weather Chart
LESSON 2: (7:16) Getting set up
LESSON 3: (34:54) Adding gridlines
LESSON 4: (2:17) Adding freezing
LESSON 5: (12:55) Adding the temperature area
LESSON 6: (8:26) Adding the UV index marks
LESSON 7: (7:38) Adding the cloud cover bubbles
LESSON 8: (7:34) Adding the precipitation bubbles
LESSON 9: (20:09) Adding annotations
LESSON 10: (30:51) Adding the tooltip
LESSON 11: (0:52) Wrapping up
LESSON 12: Week 7: Exercise

MODULE 7:
D3 + JAVASCRIPT FRAMEWORKS
LESSON 1: (2:48) D3 + Javascript Frameworks
LESSON 2: (3:11) React.js
LESSON 3: (4:48) Digging in
LESSON 4: (4:03) Creating dimensions in React
LESSON 5: (6:12) Drawing our canvas in React
LESSON 6: (3:39) Creating our scales in React
LESSON 7: (5:26) Drawing our data in React
LESSON 8: (7:34) Drawing our peripherals in React
LESSON 9: (15:11) Drawing our peripherals in React, take two
LESSON 10: (4:27) Setting up interactions in React, and wrapping up
LESSON 11: (7:41) Using d3 with Angular.js
LESSON 12: (7:30) Using d3 with Svelte.js

MODULE 8:
INTERVIEWS
LESSON 1: Interviews
LESSON 2: (16:01) Shirley Wu
LESSON 3: (19:07) Ian Johnson
LESSON 4: (14:45) Russell Goldenberg
LESSON 5: (28:42) Will Chase

Ask a question

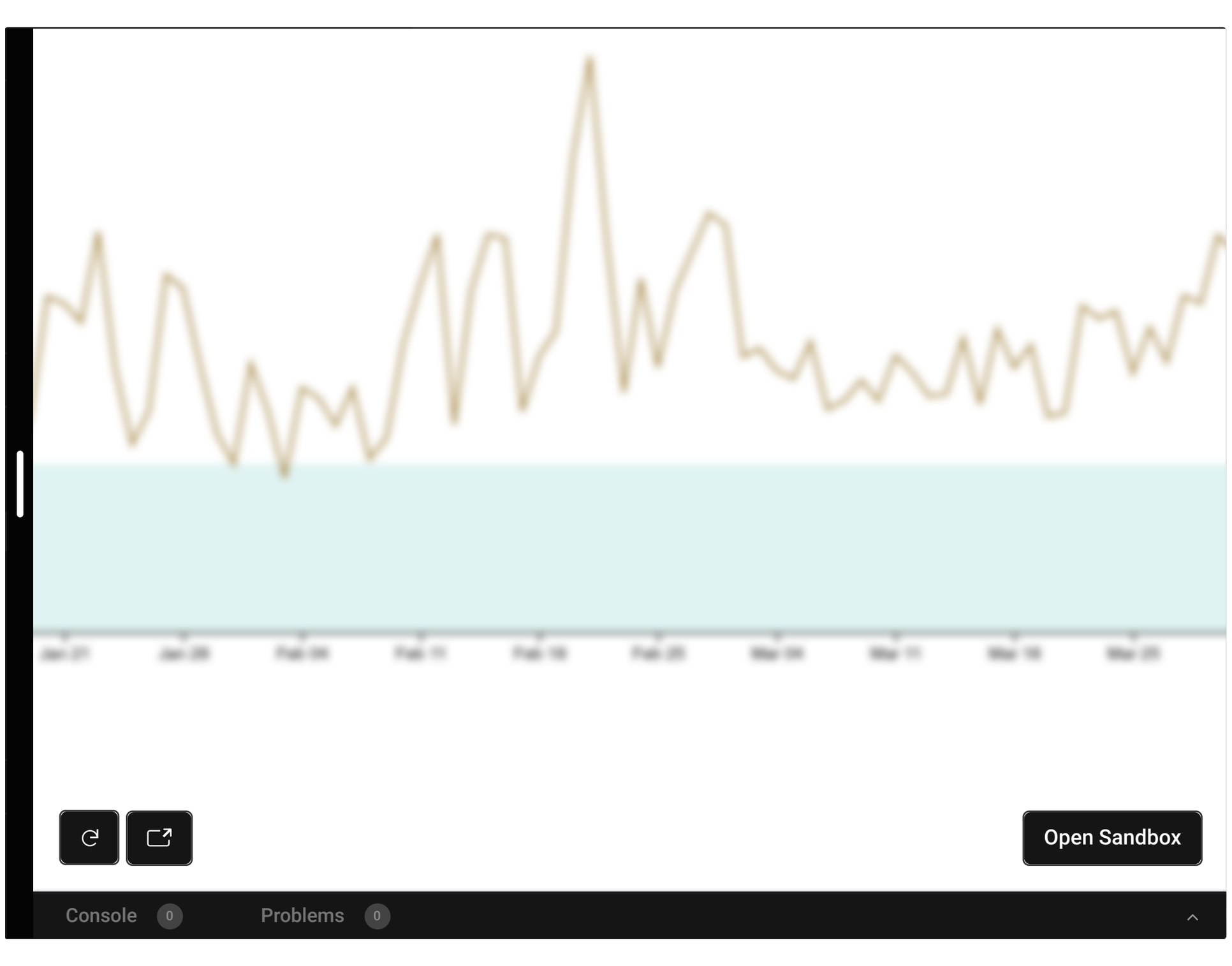
Lines

We talk through a more complex example: animating our line chart when it gets new data. This is trickier than it might seem at first!

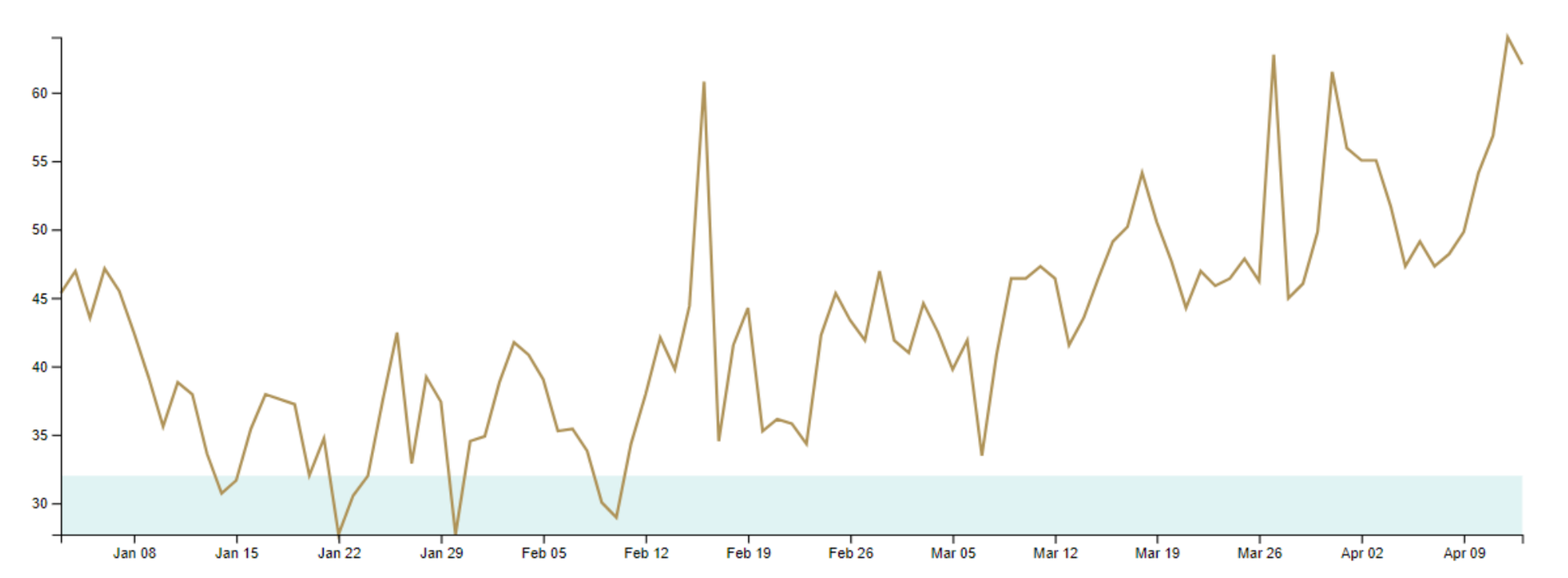
LESSON DISCUSSION 1

Contents
Lines
Final code for this lesson

After animating bars, animating line transitions should be easy, right? Let's find out!



Does this example look familiar? This is our timeline drawing code from **Module 1** with some small updates.



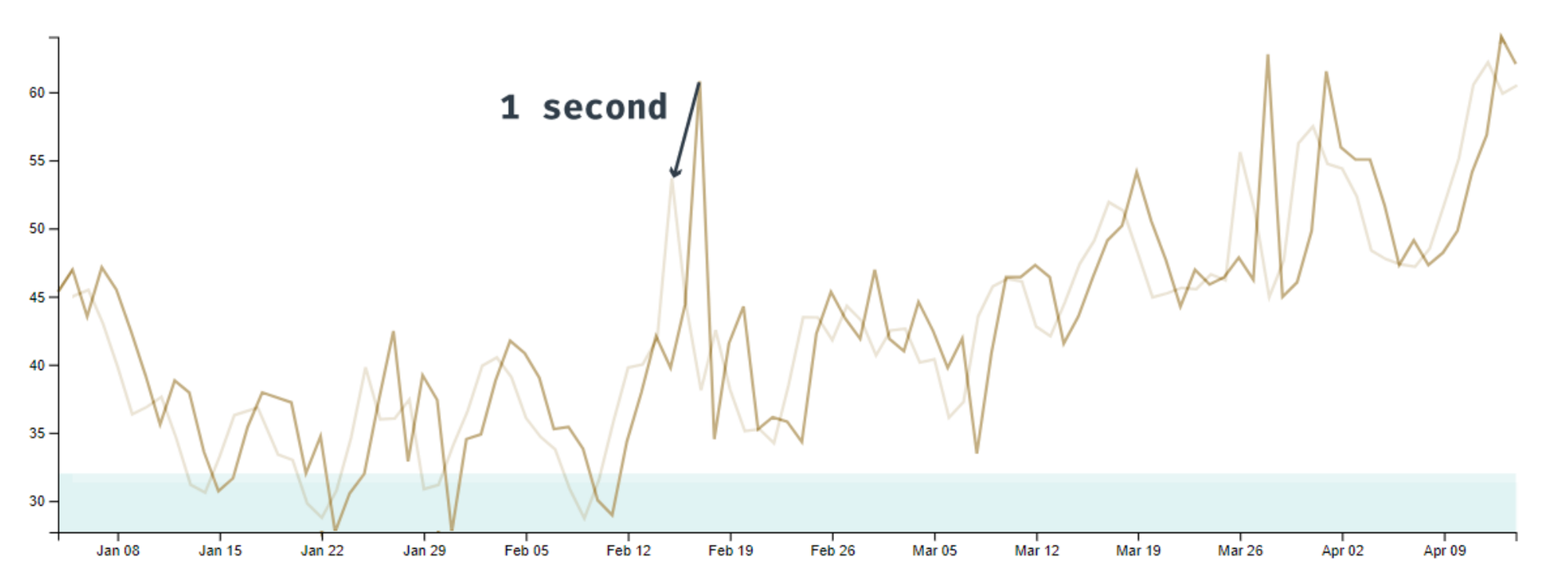
One of the main changes is an `addNewDay()` function at the bottom of the script. This exact code isn't important — what is good to know is that `addNewDay()` shifts our dataset one day in the future. To simulate a live timeline, `addNewDay()` runs every 1.5 seconds.

> If you read through the `addNewDay()` code and were confused by the `...dataset.slice(1),` syntax, the `...` is using ES6 spread syntax to expand the dataset (minus the first point) in place. Read more about it in the MDN docs.

We can see our timeline updating when we load our webpage, but it looks jerky. We know how to smooth the axis transitions, let's make them nice and slow.

Great! Now let's transition the line.

What's going on here? Why is our line wriggling around instead of adding a new point at the end?



Remember when we talked about how path **d** attributes are a string of draw-to values, like a learn-coding turtle? d3 is transitioning each point to **the next point at the same index**. Our transition's `.attr()` function has no idea that we've just shifted our points down one index. It's guessing how to transition to the new **d** value, animating each point to the next day's y value.

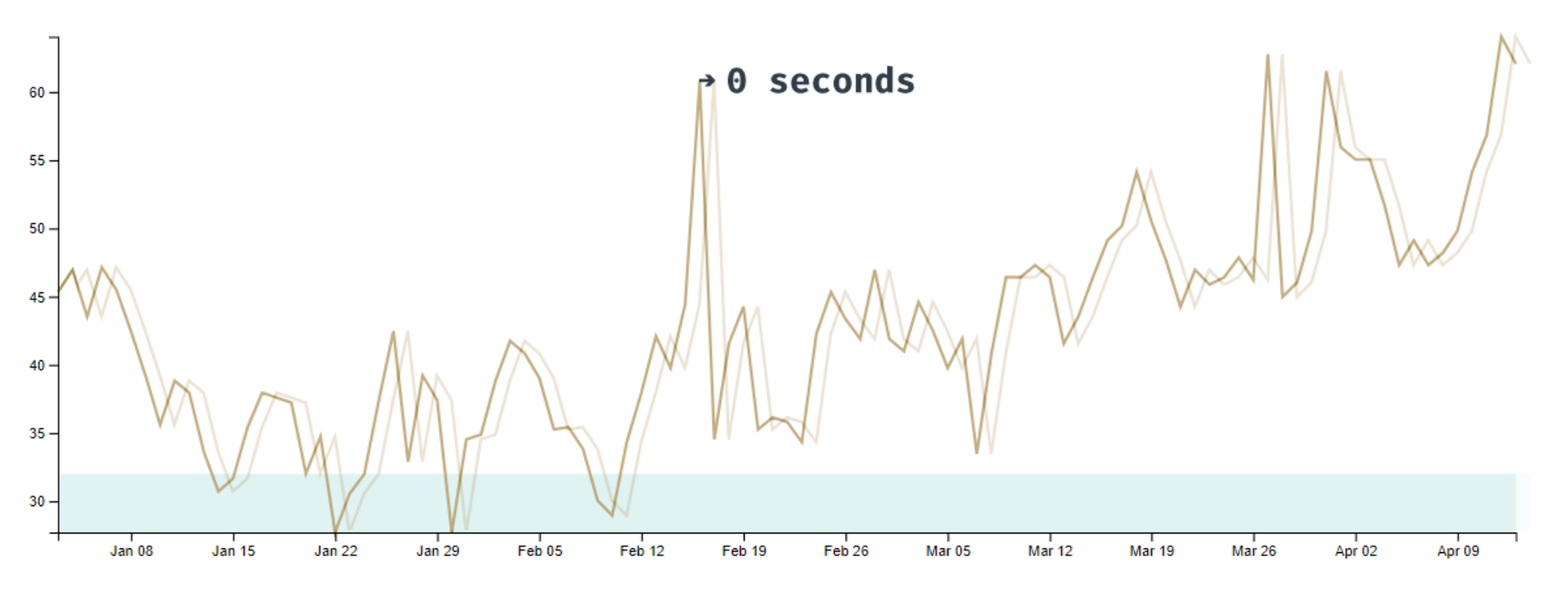
Pretend you're the `.attr()` function - how would you transition between these two **d** values?

It would make the most sense to transition each point individually, interpolating from `0 50` to `0 60` instead of moving each point to the left.

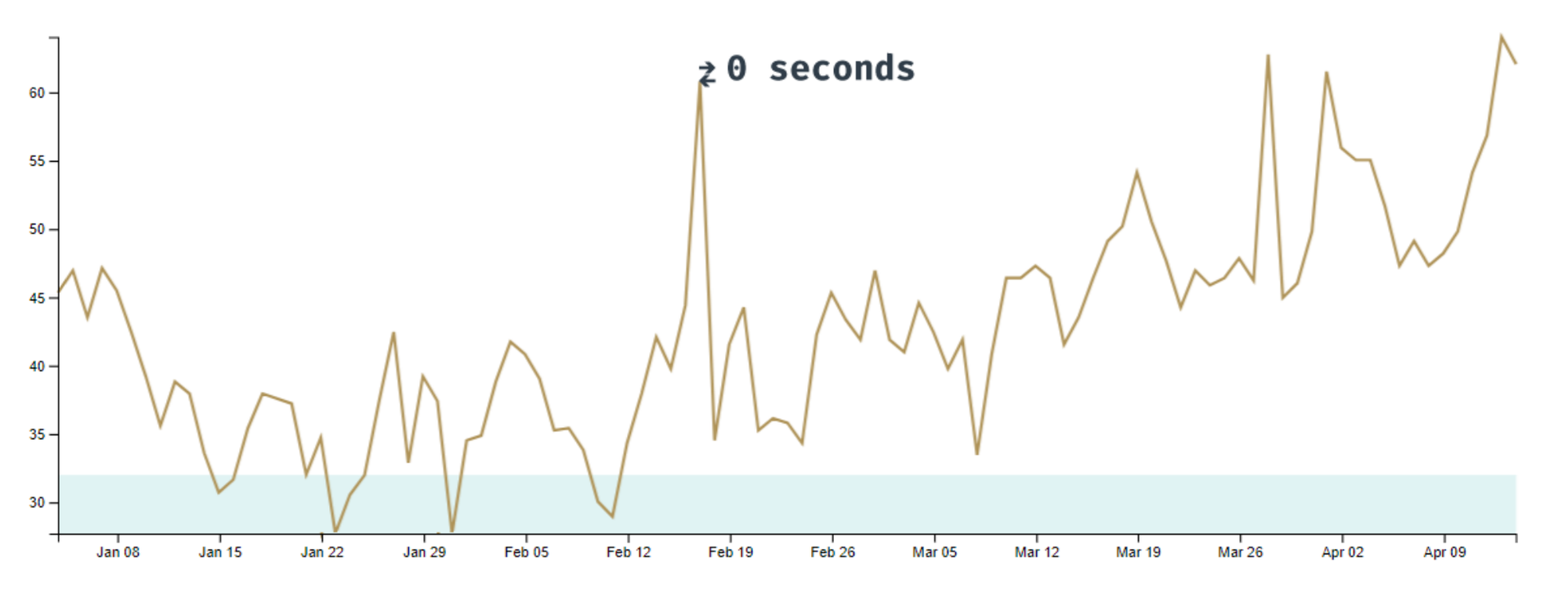
Great, we understand *why* our line is wriggling, but how do we shift it to the left instead?

Let's start by figuring out **how far we need to shift our line** to the left. Before we update our line, let's grab the last two points in our dataset and find the difference between their x values.

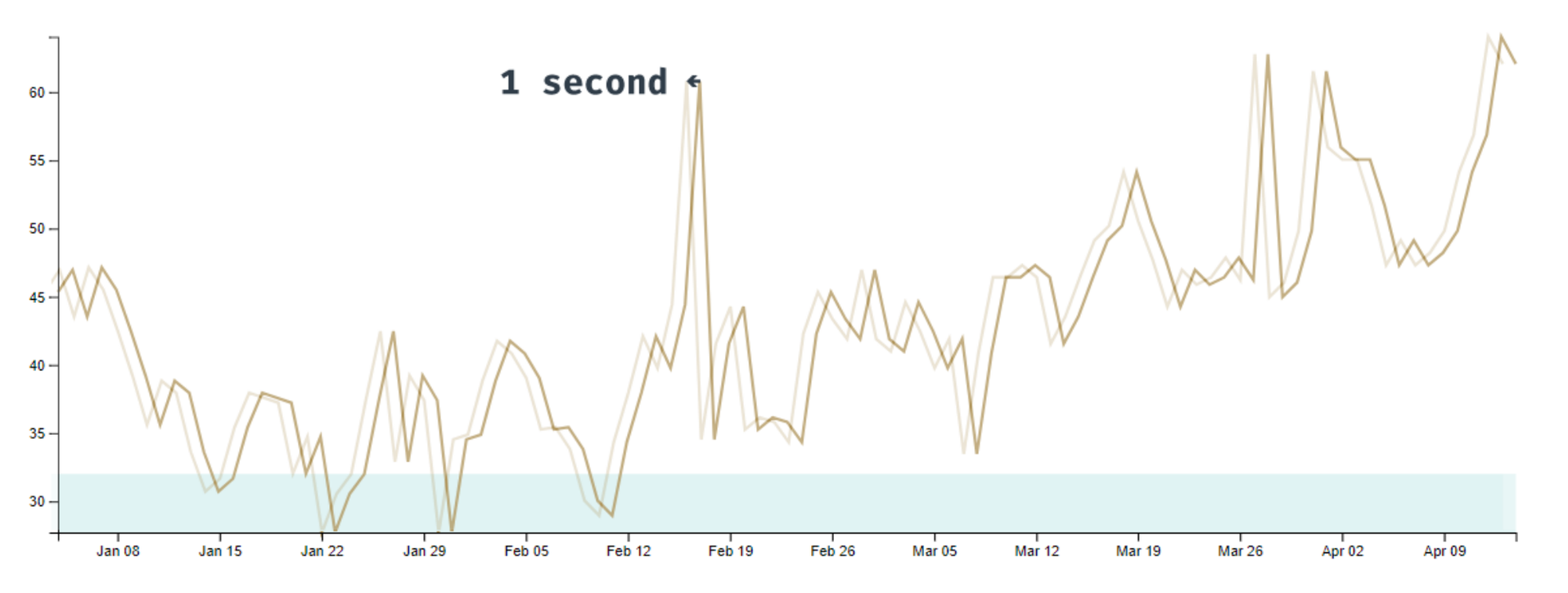
Now when we update our line, we can instantly shift it to the right to match the old line.



This shift should be invisible because **we're shifting our x scale to the left by the same amount at the same time**.



Then we can animate un-shifting the line to the left, to its normal position on the x axis.



Okay great! We can see the line updating before it animates to the left, but we don't want to see the new point until it's within our bounds. **The easiest way to hide out-of-bounds data is to add a `<clipPath>`**.

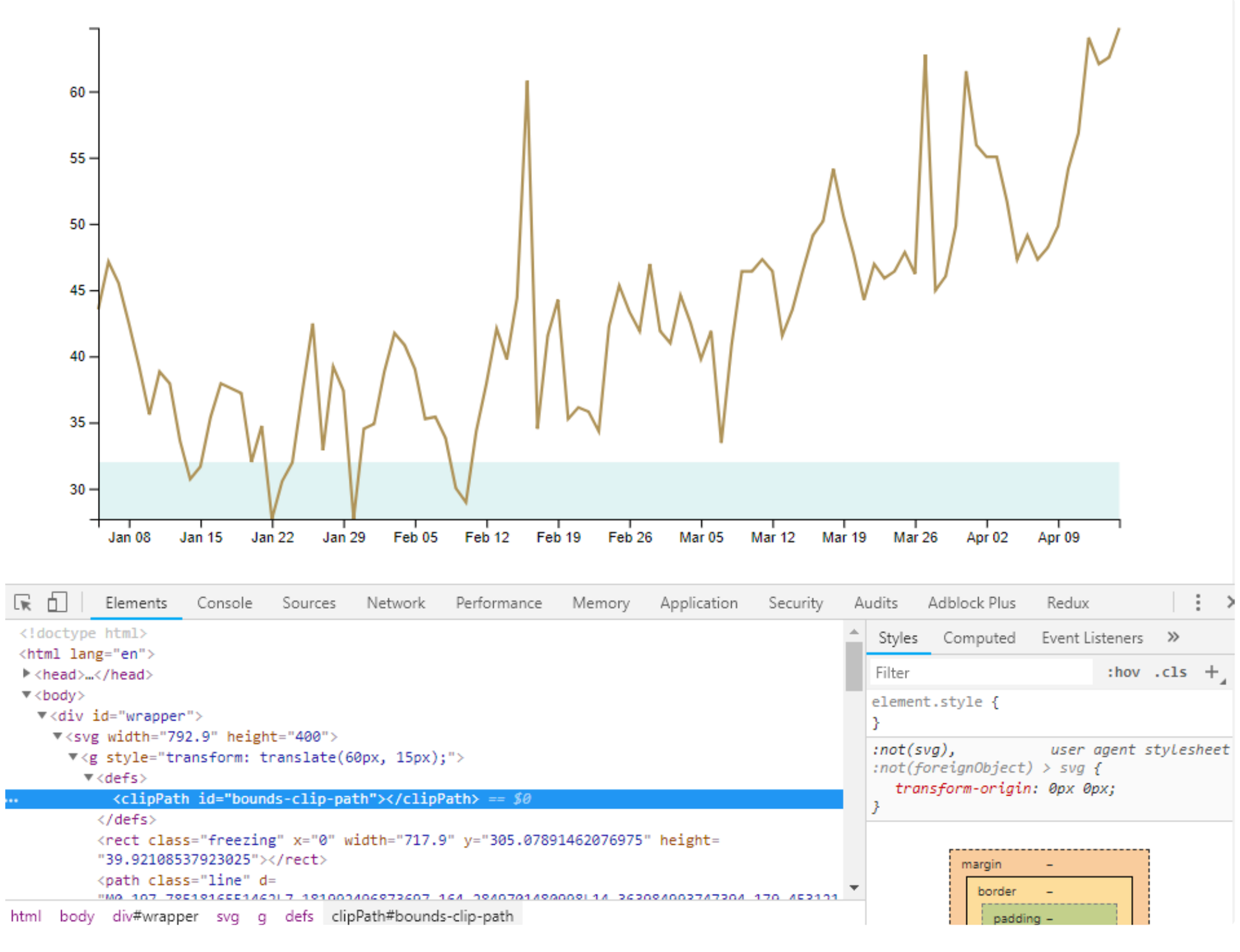
A `<clipPath>` is an SVG element that:

- is sized by its children. If a `<clipPath>` contains a **circle**, it will only paint content within that circle's bounds.
- can be referenced by another SVG element, using the `<clipPath>`'s `id`.

Before we test it out, we need to learn one important SVG convention: using `<defs>`. The SVG `<defs>` element is used to store any re-usable definitions that are used later in the `<svg>`. By placing any `<clipPath>`s or gradients in our `<defs>` element, we'll make our code more accessible. We'll also know where to look when we're debugging, similar to defining constants in one place before we use them.

Now that we know this convention, let's create our `<defs>` element and add our `<clipPath>` inside. We'll want to put this definition right after we define our **bounds**. Let's also give it an `id` that we can reference later.

If we inspect our `<clipPath>` in the Elements panel, we can see that it's not rendering at all.

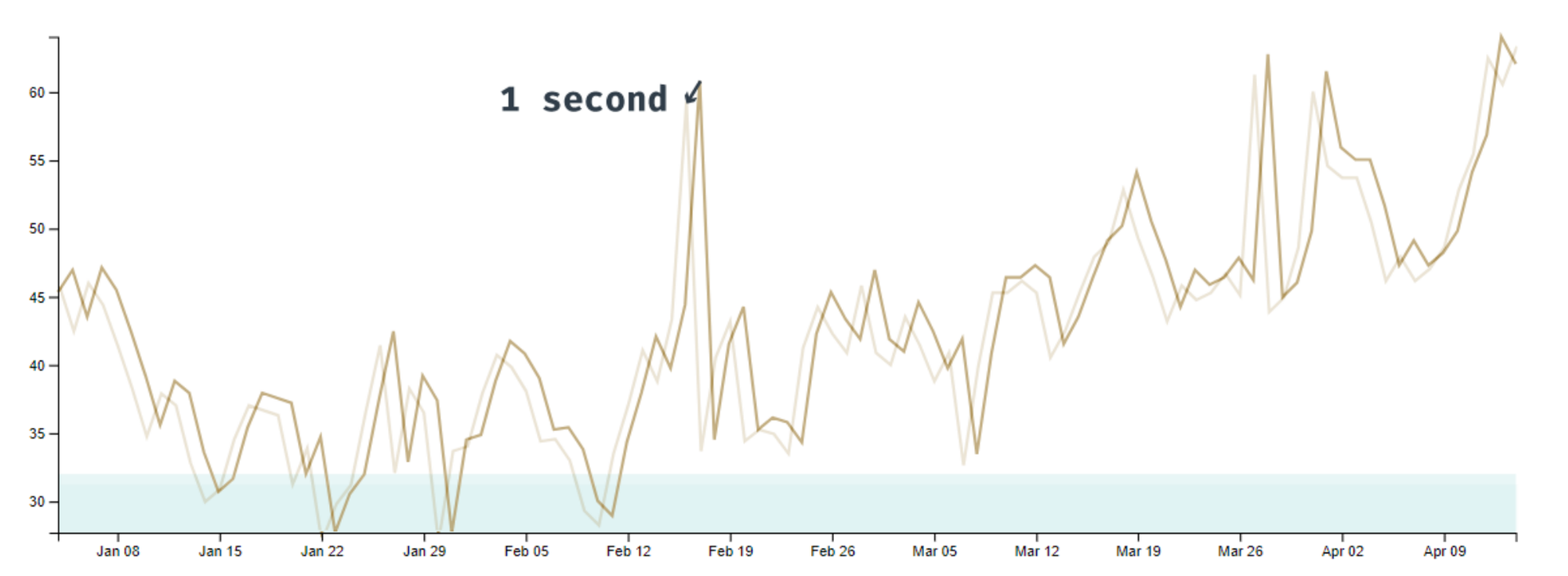


Remember, the `<clipPath>` element's shape depends on its children, and it has no children yet. Let's add a `<rect>` that covers our bounds.

To use our `<clipPath>` we'll create a group with the attribute `clip-path` pointing to our `<clipPath>`'s `id`. The order in which we draw SVG elements determines their "z-index". Keeping that in mind, let's add our new group *after* we draw the **freezing** `<rect>`.

Now we can update our `path` to sit inside of our new group, instead of the `bounds`.

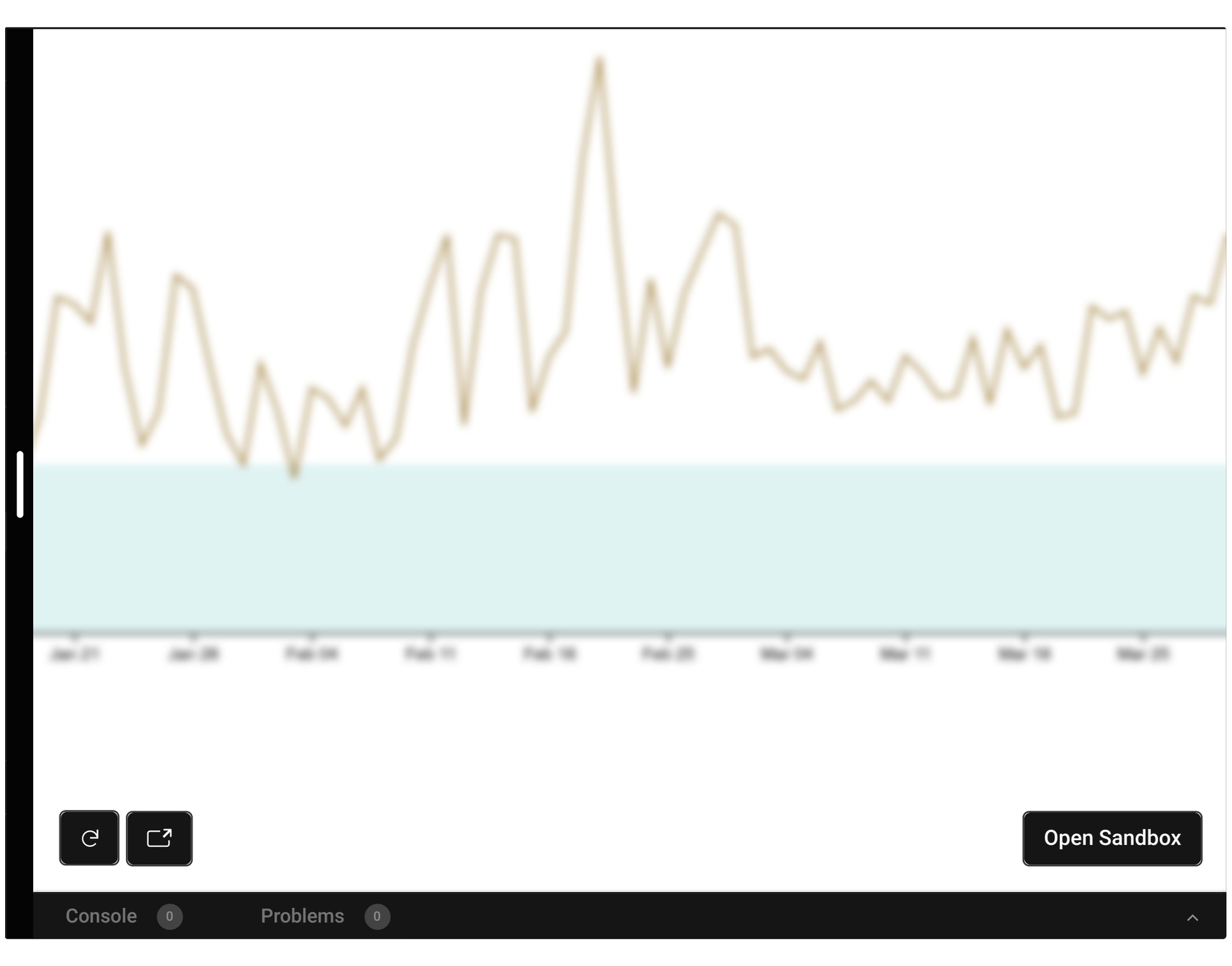
Voila! We can see that our line's new point isn't fully visible until it has finished un-shifting.



> We can see that the first point of our dataset is being removed before our line un-shifts. I bet you could think of a few ways around this — feel free to implement one or two! We could save the old dataset and preserve that extra point until our line is unshifted, or we could slice off the first data point when we define our x scale. In a production graph, the solution would depend on how our data is updating and what's appropriate to show.

Now that we have the tools needed to make our chart transitions lively, we'll learn how to let our users interact with our charts!

Final code for this lesson



Previous Lesson: d3.transition

Next Lesson: Week 4: Exercise