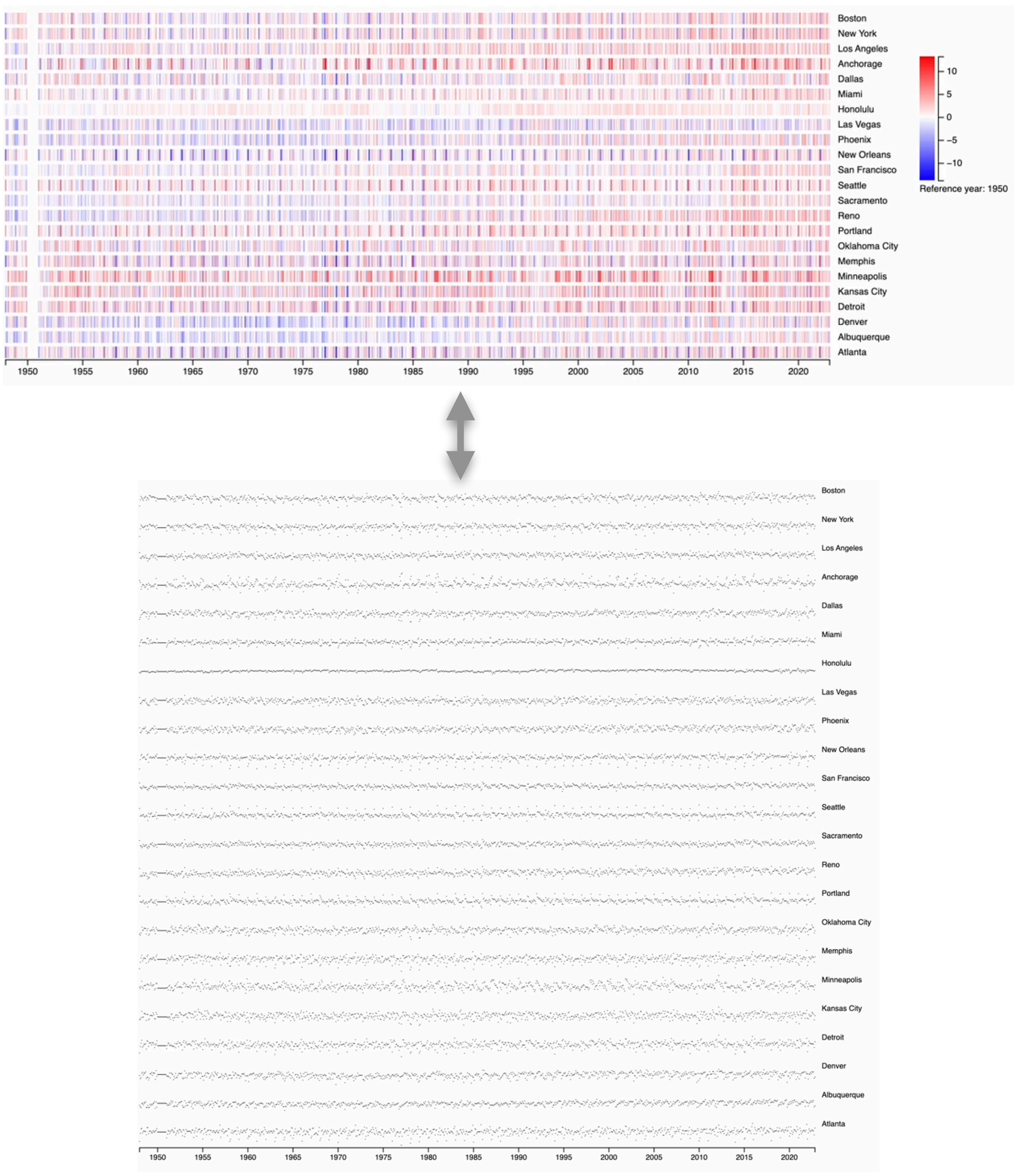
INF552 (2023-2024) - PC s05

Time-series data are often visualized using line plots, mapping values and their evolution over time to the y-axis (time-to-space mapping, illustrated in slide 9 of INF552-2023-course.pdf). But it is possible to map values to other visual encoding channels, such as color, as illustrated on slide 30 of that same slide deck.

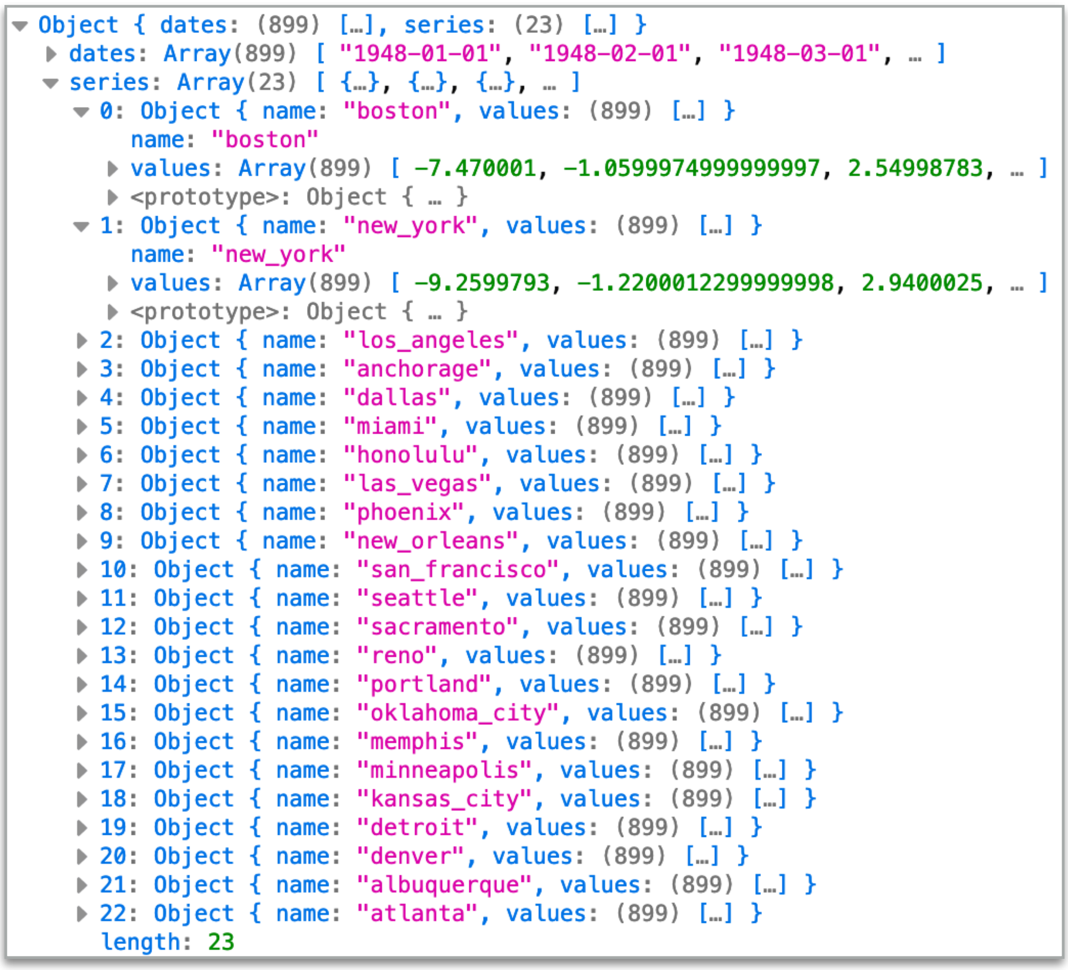
Today we visualize temperature variations in a set of US cities, from 1948 to 2022, using monthly averages. We use both visual mapping strategies (temperature to y-position, or temperature to color). See Figure below.

Goal: the code to generate the color-strip visualization in the Figure below is already made available to you in the code skeleton. The goal now is to create an animated transition between that visualization and a classic time-series plots, back and forth, with staging and delay.



Data Structure

Code in the skeleton already takes care of parsing and transforming the data into a structure that is more suitable for our purposes. Each time-series is a JS object with two fields: its name and an array of temperature values over time:



```
▼ Object { dates: (899) […], series: (23) […] }
  ▶ dates: Array(899) [ "1948-01-01", "1948-02-01", "1948-03-01", … ]
  ▼ series: Array(23) [ {…}, {…}, {…}, … ]
    ▼ 0: Object { name: "boston", values: (899) […] }
        name: "boston"
      ▶ values: Array(899) [ -7.470001, -1.0599974999999997, 2.54998783, … ]
      ▶ <prototype>: Object { … }
    ▼ 1: Object { name: "new_york", values: (899) […] }
        name: "new_york"
      ▶ values: Array(899) [ -9.2599793, -1.2200012299999998, 2.9400025, … ]
      ▶ <prototype>: Object { … }
    ▶ 2: Object { name: "los_angeles", values: (899) […] }
    ▶ 3: Object { name: "anchorage", values: (899) […] }
    ▶ 4: Object { name: "dallas", values: (899) […] }
    ▶ 5: Object { name: "miami", values: (899) […] }
    ▶ 6: Object { name: "honolulu", values: (899) […] }
    ▶ 7: Object { name: "las_vegas", values: (899) […] }
    ▶ 8: Object { name: "phoenix", values: (899) […] }
    ▶ 9: Object { name: "new_orleans", values: (899) […] }
    ▶ 10: Object { name: "san_francisco", values: (899) […] }
    ▶ 11: Object { name: "seattle", values: (899) […] }
    ▶ 12: Object { name: "sacramento", values: (899) […] }
    ▶ 13: Object { name: "reno", values: (899) […] }
    ▶ 14: Object { name: "portland", values: (899) […] }
    ▶ 15: Object { name: "oklahoma_city", values: (899) […] }
    ▶ 16: Object { name: "memphis", values: (899) […] }
    ▶ 17: Object { name: "minneapolis", values: (899) […] }
    ▶ 18: Object { name: "kansas_city", values: (899) […] }
    ▶ 19: Object { name: "detroit", values: (899) […] }
    ▶ 20: Object { name: "denver", values: (899) […] }
    ▶ 21: Object { name: "albuquerque", values: (899) […] }
    ▶ 22: Object { name: "atlanta", values: (899) […] }
      length: 23
```

This data structure is used to generate the initial visualization.

What we will do now is add functions to create animated transitions from the initial color strips to line plots, and conversely.

Animation

A function called `toggleVis()` gets invoked whenever users click on the `switch` button located on the left of the SVG canvas. This event callback animates:
- either between the color strips and the line plots (`transitionToLinePlots()`),
- or conversely (`transitionToColorStrips()`).

The callback is already implemented. Your job is to write the content of these two functions, as detailed in the next two sections.

1. Transition from Color Strips to Line Plots

Write the contents of function `transitionToLinePlots()`:

Step 1.1: make the legend (temperature scale) fade out (just play on the group's opacity);

Step 1.2: then, translate all color strips, labels, as well as the time axis to fill the entire height of the SVG canvas (`900px`);

Step 1.3: then, make all strips `1px` tall;

Step 1.4: then, turn all lines dark gray;

Step 1.5: then, adjust each `<line>`'s y-coordinates based on the datum it is bound to (temperature difference with the year 1950 baseline).

Step 1.6: once your transition works, introduce delay between the animation of each `<line>` of a strip in Step 1.5. This literally involves adding *one* line of code in the right place. Refer to INF552-2023-PC-slides-s05.pdf slide #3.

2. Transition from Line Plots to Color Strips (optional)

Step 2: Write the contents of function `transitionToColorStrips()`so that it does the opposite. Pay attention to sequence and delay so that it looks similar to the final animation shown in the assignment's video.