Week 8

Drawing Complex Shapes

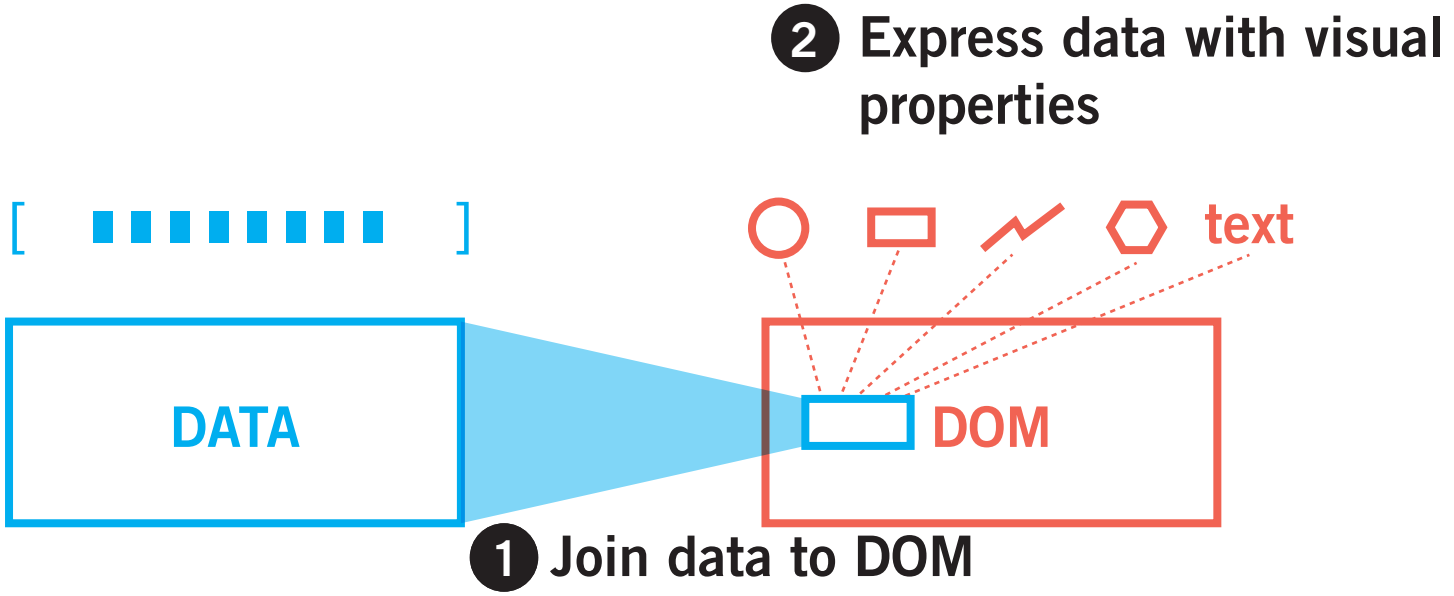
Using d3 Generators

Generalizing the Data Viz Process

Acquire
Parse
Filter
Mine
Represent
Refine
Interact

"REPRESENT" IN d3 - DESIGN INTENTION

The basic idea is to "join" a piece of data to a DOM element, and then use the visual attribute of the DOM element to express the data



“REPRESENT” IN d3 - DESIGN IMPLEMENTATION

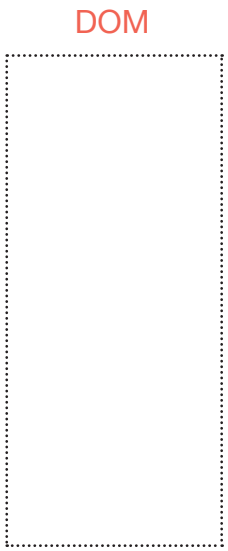
```
.selectAll() - .data()-.enter() - .append()
                       .exit() - .remove()
                       .transition() - .attr()...
```

creates / removes / updates DOM elements, so that there are matching number of data elements to DOM elements, and these DOM elements are visually updated to reflect underlying data

RECAP OF `enter / exit / update`

`.selectAll()`

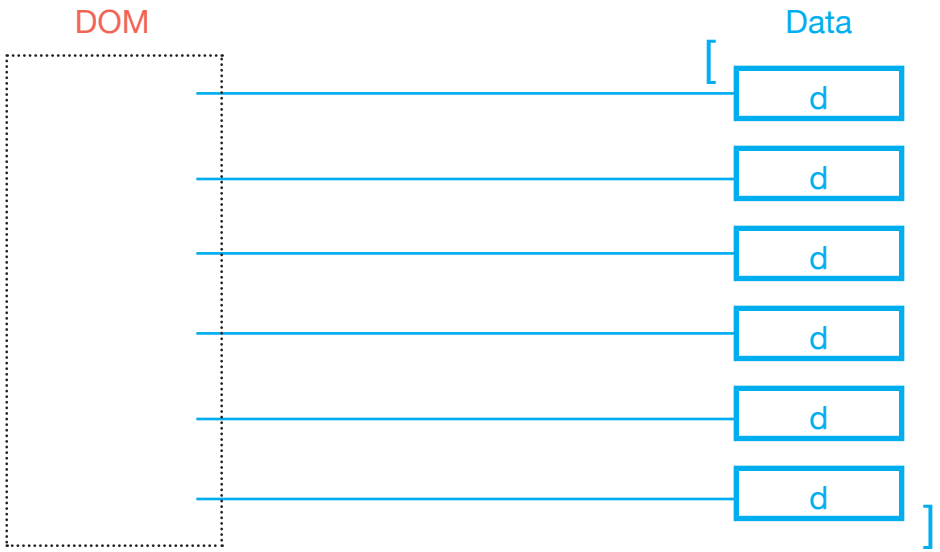
This tries to select all DOM elements that fit the criteria. Often, this results in an empty selection.



RECAP OF `enter / exit / update`

`.data()`

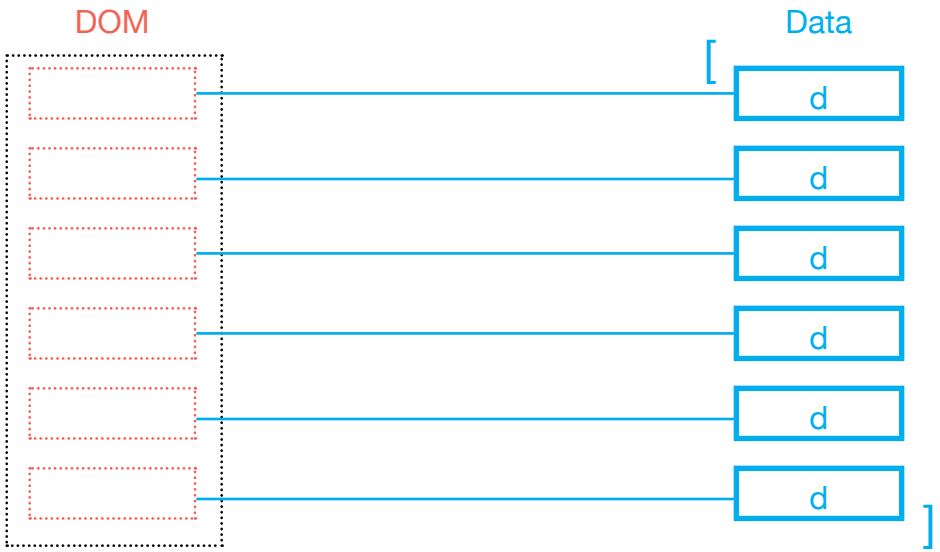
This tries to bind each element in the data array to each DOM element in the selection



RECAP OF `enter / exit / update`

`.enter()`

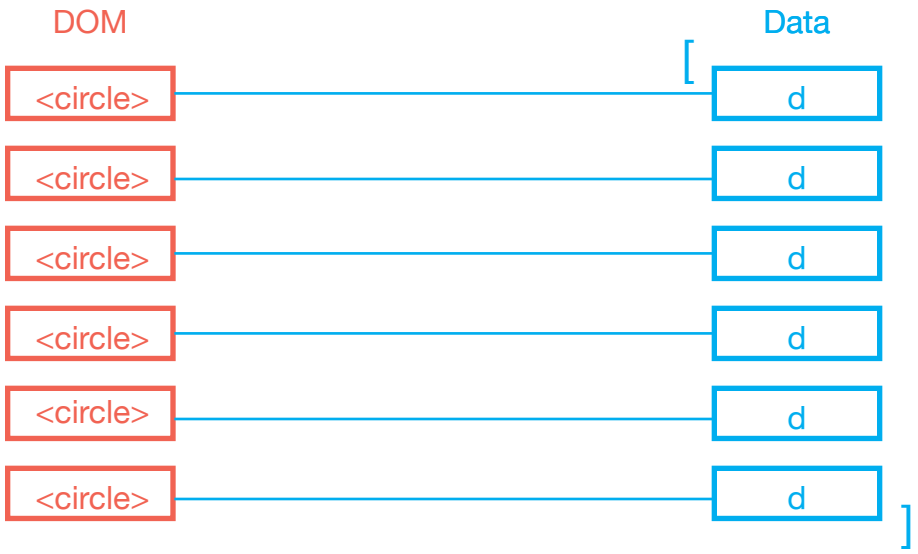
For each mismatch, create an empty placeholder



RECAP OF `enter / exit / update`

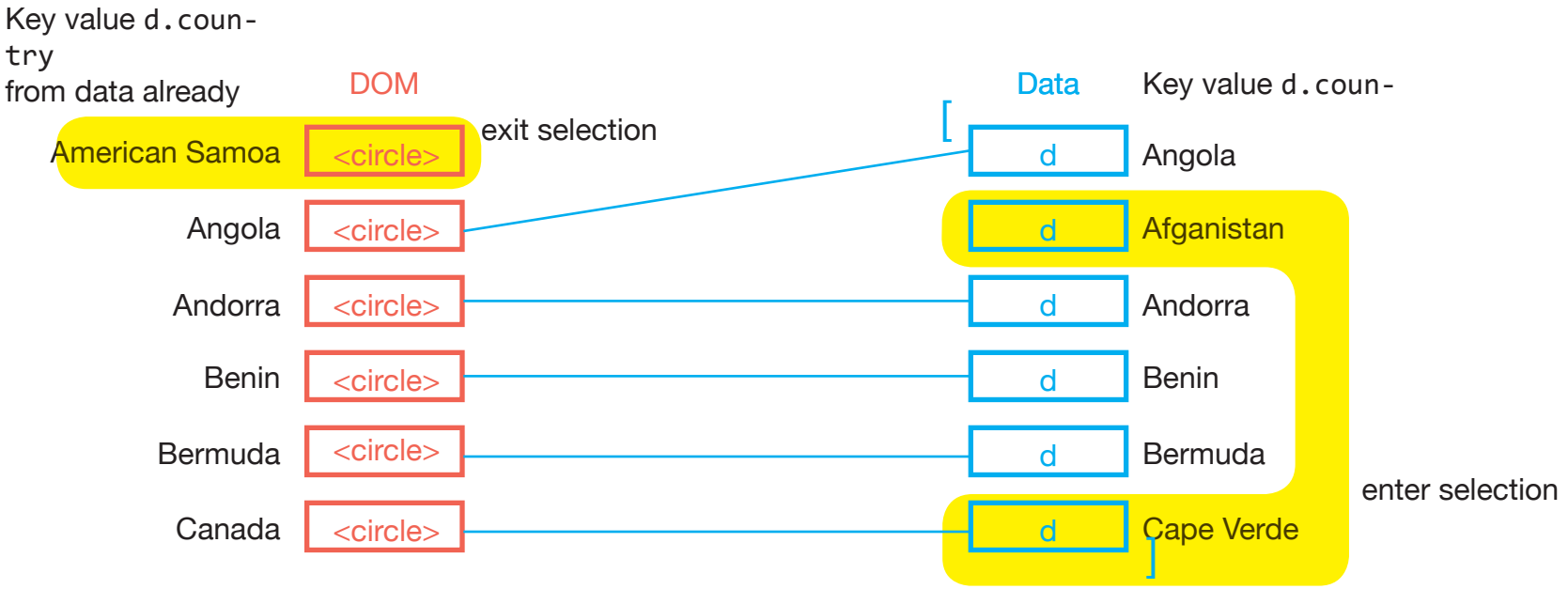
`.append()`

For each empty placeholder, append some DOM element (could be anything!)



RECAP OF `enter / exit / update`: ENSURE OBJECT CONSTANCY

With a key function, data elements are matched to DOM elements based on a key value.

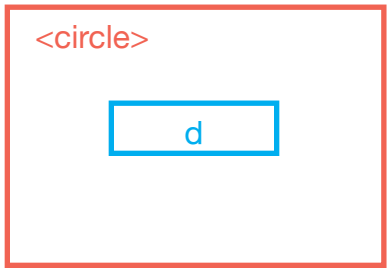


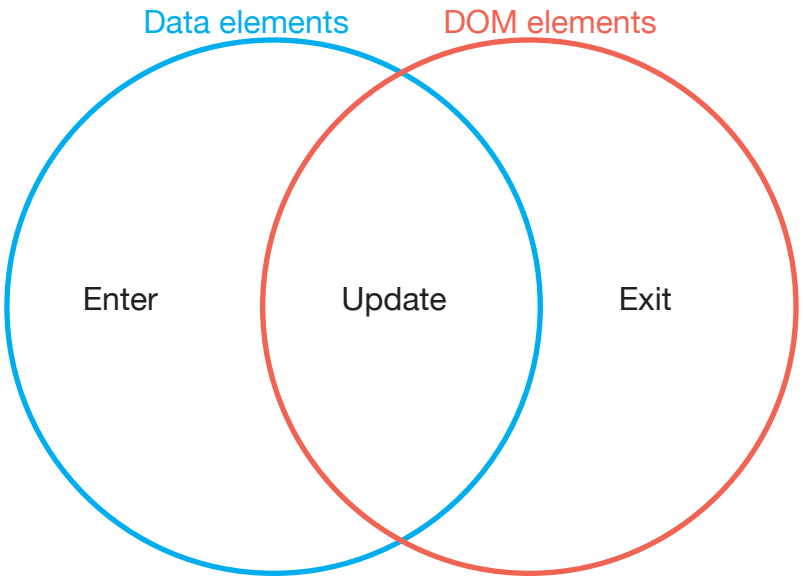
RECAP OF `enter / exit / update:` ACCESSOR FUNCTIONS

Accessor functions allow you to access the data element bound to individual DOM elements from the selection. They are usually of the form:

```
...
    .attr('r', function(d,i){...};)
...
```

where argument `d` represents the data element (either a value or an object), and `i` represents the index of the DOM element within the overall selection.





With key functions and object constancy, it's entirely possible to have an enter, exit, and update set all at the same time.

WHAT ABOUT MORE COMPLEX SHAPES?

The svg <path> element is the basis for drawing more complex shapes.

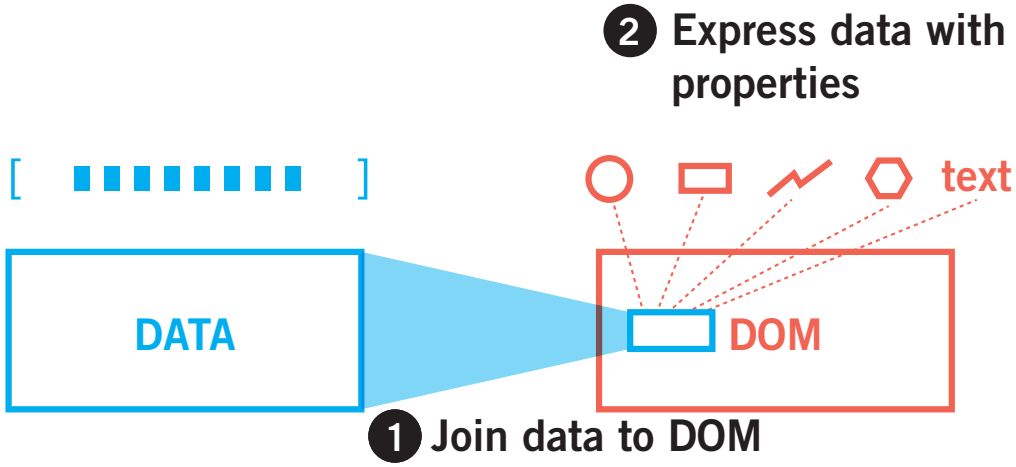
Why does it have a "d" attribute? It does NOT mean "data", but rather "geometry".

```
<svg>
...
   <path "d"= "...">
</svg>
```

How do we draw this complex shape?

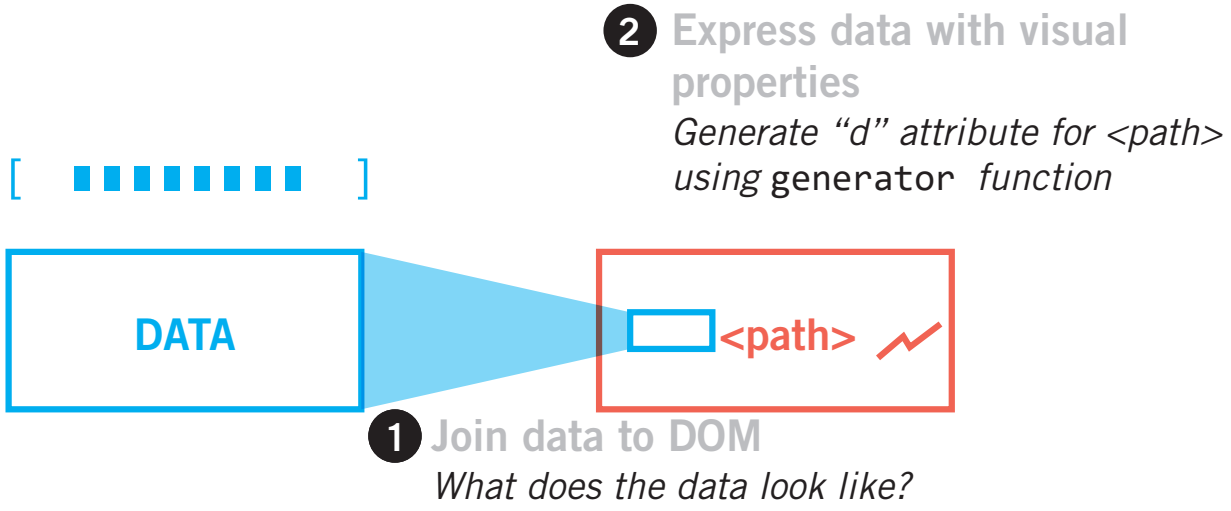
WHAT ABOUT MORE COMPLEX SHAPES?

The same design intentions hold, but the implementation is a little more complicated.



WHAT ABOUT MORE COMPLEX SHAPES?

The same design intentions hold, but the implementation is a little more complicated.

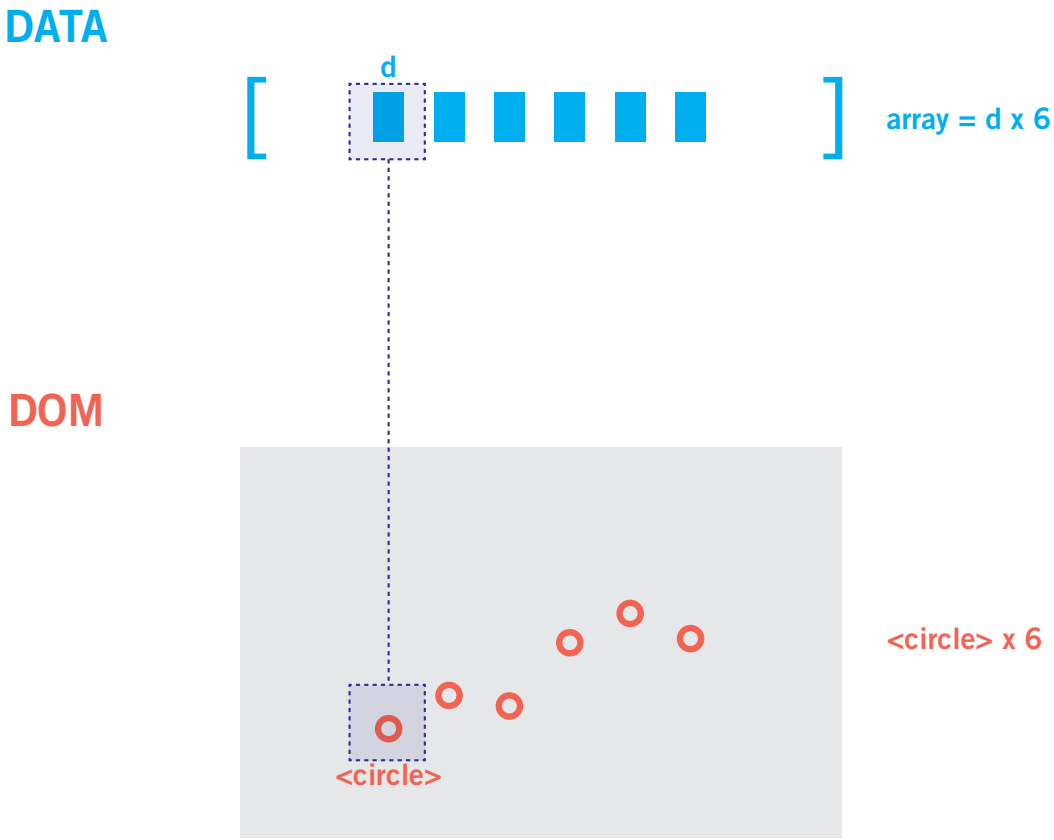


Exercise 1

Let's learn to draw a basic <path> based on a time series.

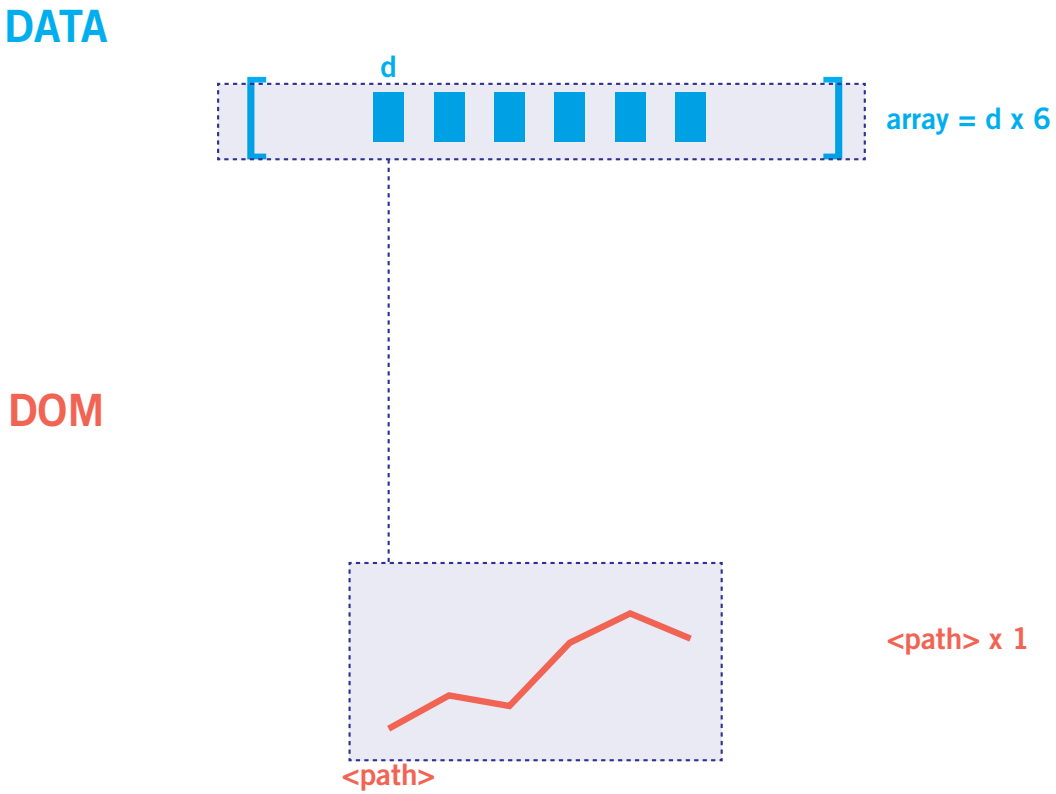
A TIME SERIES <path> REQUIRES A DIFFERENT FORM OF UNDERLYING DATA

Scatterplot: 6 <circle> elements, joined to 6 data elements "d"



A TIME SERIES <path> REQUIRES A DIFFERENT FORM OF UNDERLYING DATA

Line graph: 1 <path> element, joined to 1 data element "array"



HOW DOES THE CODE WORK?

Line graph: 1 <path> element, joined to 1 data element "array".
Which of the following is right?

```
//Pattern 1

plot.selectAll('path')
     .data(dataArray)
     .enter()
     .append('path')
     ...
```

```
//Pattern 2

plot.append('path')
     .datum(dataArray)
```

HOW DOES THE CODE WORK?

Line graph: 1 <path> element, joined to 1 data element "array".
Which of the following is right?

```
//Pattern 1

plot.selectAll('path')
     .data(dataArray)
     .enter()
     .append('path')
     ...

//But this works!
plot.selectAll('path')
     .data([dataArray])
     .enter()
     .append('path')
     ...
```

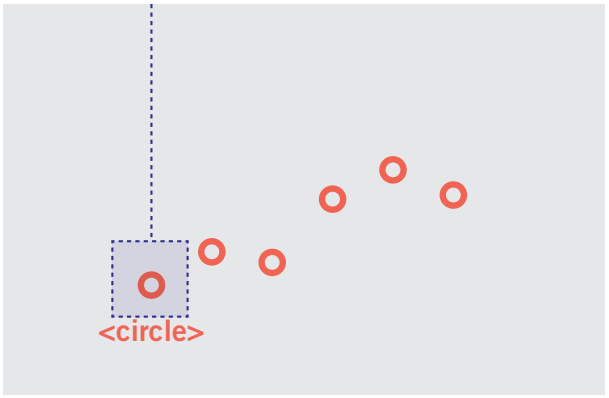
```
//Pattern 2

plot.append('path')
     .datum(dataArray)
```

DATA

array = d x 6

DOM



<circle> x 6

```
// “n-to-n” join

plot.selectAll(‘circle’)
     .data(dataArray)
     .enter()
     .append(‘circle’)
     ...
```

WE JOINED DATA TO DOM, BUT HOW DO WE GEN-ERATE THE RIGHT VISUAL APPEARANCE / GEOM-

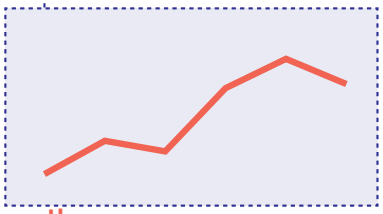
DATA

array = d x 6

```
// “1-to-1” join

plot.append(‘path’)
     .datum(dataArray)
     .attr(‘d’,...)
```

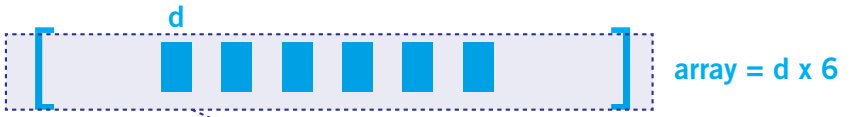
DOM



<path> x 1

WE JOINED DATA TO DOM, BUT HOW DO WE GEN-ERATE THE RIGHT VISUAL APPEARANCE / GEOM-

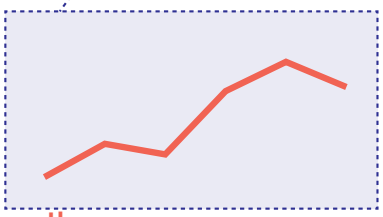
DATA



array = d x 6

```
function generator(array){
     //converts array of data points
     //to path geometry “d”
}
```

DOM



<path> x 1

<path>

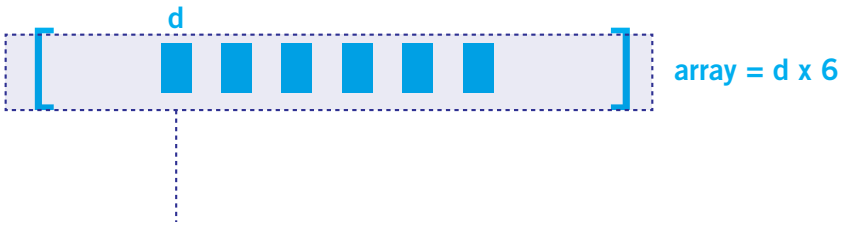
```
// “1-to-1” join

plot.append(‘path’)
     .datum(dataArray)
     .attr(‘d’, generator)
     .style(...)
```

“LINE” GENERATOR

```
var lineGenerator = d3.svg.line()
    .x(function(d){ return ...})
    .y(function(d){ return ...})
    .interpolate(‘basis’);
```

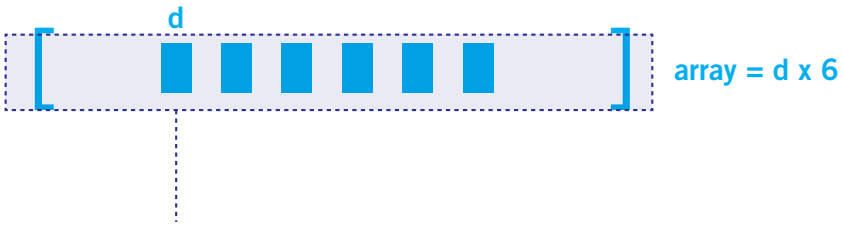
DATA



“AREA” GENERATOR

```
var lineGenerator = d3.svg.line()
    .x(function(d){ return ...})
    .y0(function(d){ return ...})
    .y1(function(d){ return ...}
    .interpolate(‘basis’);
```

DATA



Exercise 2

How to update and transition a <path> DOM object.

How to use the tooltip pattern.

d3.map( )

A map structure allows easy look up between “a” and “b”, akin to a dictionary.

```
var newMap = d3.map();

newMap.set(“Christmas”, new Date(2015,11,25);
newMap.get(“Christmas”).getDay(); //5-->Friday
```

Exercise 3

Fun with some useful utilities:

javascript `Date` object

`d3.map()`

Exercise 4

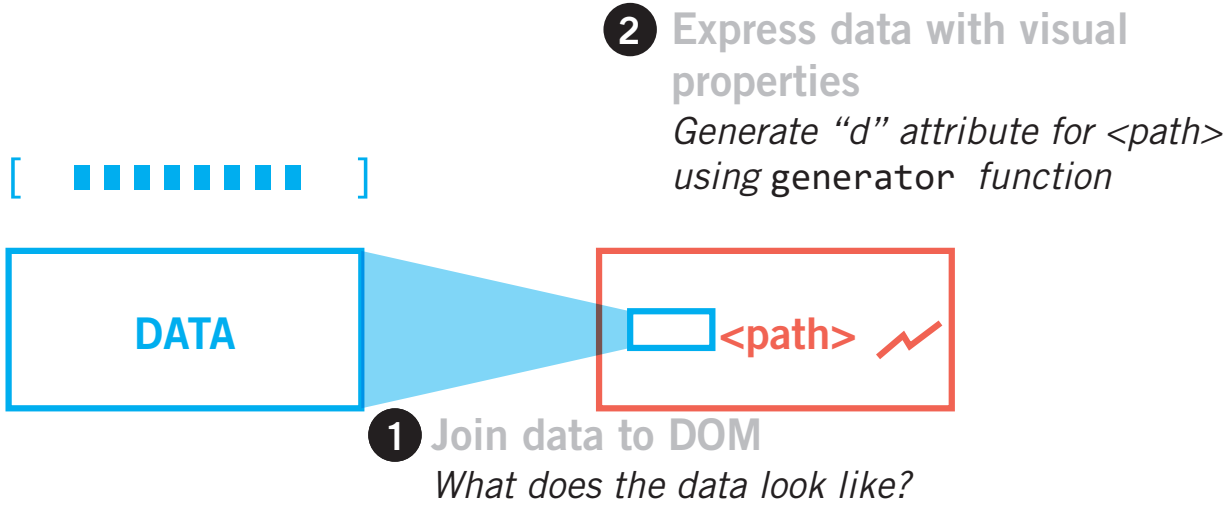
A more complicated example, involving array transformations

http://www.siqizhu.net/flightprices/

WHAT ABOUT MORE COMPLEX SHAPES?

The tricky part is often how to create the right data structure.

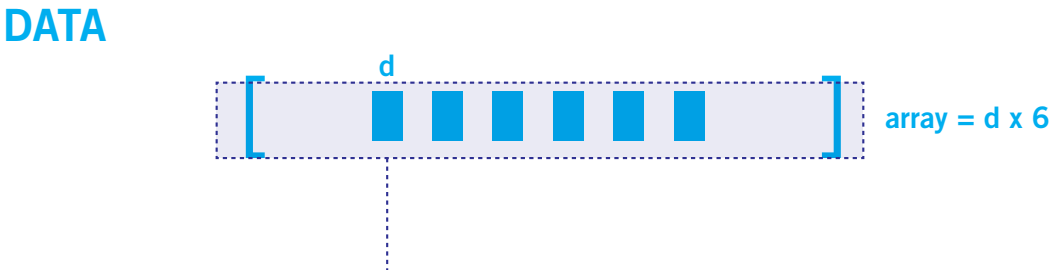
Often times, we have to create more structure out of a one-dimensional array.



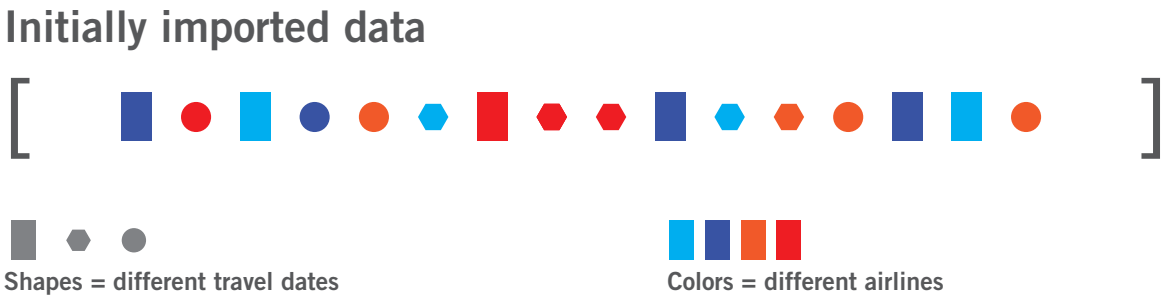
EXAMPLE: FLIGHT PRICES

The data is multi-dimensional: B x T x L x F
- When you book; (B)
- When you travel; (T)
- Which airline; (L)
- What flight of a particular airlines; (F)

The imported array is one-dimensional: 1 x N



LOOK AT IT A DIFFERENT WAY:



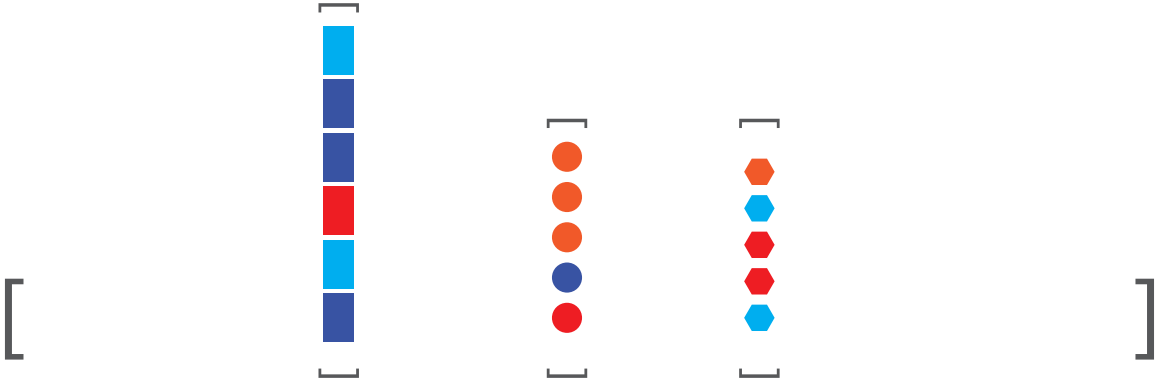
LOOK AT IT A DIFFERENT WAY:

Initially imported data

Shapes = different travel dates

Colors = different airlines

Group by travel date, so we can do a price vs. travel date comp:

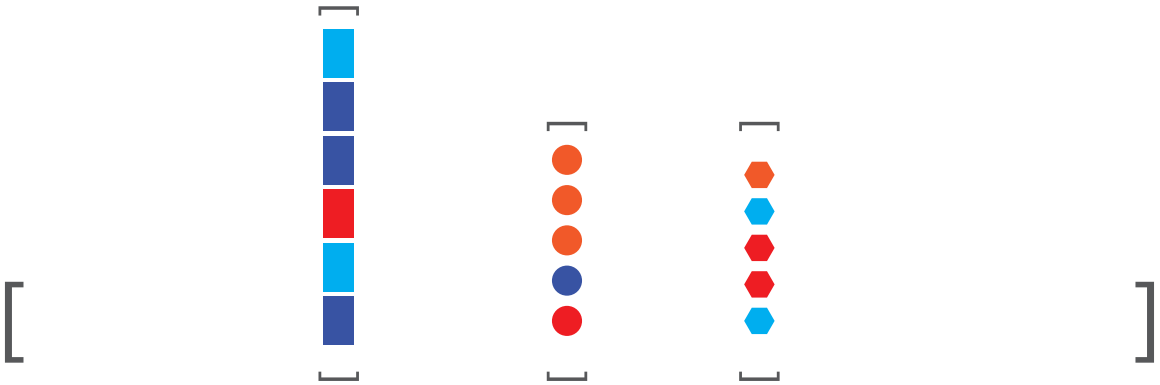


USING `d3.nest( )` TO CREATE NESTED STRUCTURE

```
var nestedData = d3.nest()
    .key(function(d){ return d.travelDate; })
    .entries(dataArray);
```

LOOK AT IT A DIFFERENT WAY:

Group by travel date, so we can do a price vs. travel date comp:



Average price of all flights by date: