Miscellaneous item #1: Sort functions

Go back to our updating bar chart example, and add sort functionality. Can sort in different ways: currently, the array values are given in order of descending value, so that's where they show up.

What if you want to sort the state abbreviations in alphabetical order?

Need to add a sort function to updateData, to sort the array before drawing it in D3.

Making a radio button

Really, we want two radio buttons, so that we can switch between them

A radio button is a kind of input:

< input type = "radio" id = "radioButton1" value = "alphabetical"
onchange = "radioChange(value)“ >

Add a label to tell us what it’s for:

< label for="radiobutton1“ > Sort Alphabetical </ label >

Create a JS function that runs when the radio button changes state. (Just a placeholder for now)

```
function  radioChange ( value ) {
   console.log ( value );
}
```

Pairing radio buttons

Really, we want two radio buttons, so that we can switch between them

Copy the previous code to make a second radio button and label.

< input type ="radio“ id="radioButton**2**"
value="decreasing" onchange="radioChange(value)“ >

<label for="radiobutton**2**“ > **Sort by Decreasing Value** </ label >

And then add a name attribute to both radio buttons to tell them that they are related to each other (one turns on, the other turns off). Also, add checked parameter to the decreasing radio button, so that it is on by default.

< input type = "radio" id = "radioButton1" **name=“sort”**
value = "alphabetical" onchange = "radioChange(value)“ >

< input type ="radio“ id="radioButton2“ **name=“sort”**
value="decreasing" onchange="radioChange(value)“ **checked** >

Setting up the sort function

Create a new variable to keep track of what kind of sorting to use. Set a default value for the first time the program is run.

```
var sortOrder = "decreasing";
```

First, add a static sort to check that it works (and set it up in ascending order so that it looks different than before)

```
loadData .sort ( function ( a , b ) {
      return   a .totalPop  -   b .totalPop ;
});
```

Reverse the order of a and b in the return statement to get back to decreasing value.

Since this sort will only run once and it agrees with the default value, don't need to change anything here.

Connecting radio buttons to updateData

We also need to run the sort function every time the data updates, in case the order has changed. Two ways to write this; one is clearer to read, the other takes fewer lines of code:

```
function updateData ( selectedYear ) {

   var newData  =  nestedData .filter ( function ( d ) {
            return d.key  = =  selectedYear  } ) [ 0 ] .values

   return   newData .sort  ( function ( a ,b ) {
            return  a .totalPop  -  b .totalPop;
   });
}

function updateData ( selectedYear ) {

    return  nestedData .filter ( function ( d ) {
            return  d.key  = =  selectedYear } ) [ 0 ] .values .sort (
            function ( a ,b ) {  return  a .totalPop  -   b .totalPop;   } );
}
```

Listening to the radio button

If we want to update the data based on which radio button is selected, we need to add code to keep track of what's been clicked.

Use radioChange( ) to update the sortOrder tracker variable:

```
function radioChange ( value ) {
   sortOrder  =  value ;
}
```

Now, add an if statement to updateData( ) to check the value of sortOrder (notice that this only works when we move the slider after changing the radio button)

```
if ( sortOrder = =  'alphabetical' ) {
      console .log ( 'alphabetical' );
}
else {
      console .log( 'descending' );
}
```

Change the sort behavior based on sortOrder

Now, we can move actual code inside of the if statement:

```
if ( sortOrder = =  'alphabetical' ) {

      return   nestedData .filter ( function ( d ) {
            return d.key  = =  selectedYear } ) [ 0 ] .values
                  .sort ( function ( a , b ) {
                        return  a .fullname .localeCompare ( b .fullname );
                  });
}
else {

     return   nestedData .filter ( function ( d ) {
            return d .key  = =  selectedYear } ) [ 0 ] .values
                  .sort ( function ( a ,b ) {
                        return  b .totalPop  -  a .totalPop;
                  });
}
```

Update the chart when the radio button is clicked

Really, we don't want to wait for the next time the user moves the slider for the radio button to take effect. Instead, we want the chart to update immediately when the radio button is called.

We want to add an updateData call inside the radio button to accomplish this, but updateData requires information about the selectedYear (currently only available when the slider updates).

Create a tracker variable to store the year. Set it to the default slider value, and update it each time the slider moves:

```
var currentYear = 1987;

function  sliderMoved ( value ) {
   newData = updateData ( value );
   currentYear  = value;
   drawPoints ( newData );
}
```

Call update and drawPoints from radio

Now, update radioChange to call updateData and drawPoints when the radio button is clicked.

```
function  radioChange ( value ) {

   sortOrder = value;
   newData = updateData ( currentYear );
   drawPoints ( newData );

}
```

Miscellaneous item #2: Live loading data

Sometimes, you want to load files when you need them, rather than at the beginning of your script.

Folder contains two data files, foodImports_AD and foodImports_AF.
Want to switch between them when the user clicks on a button.

Start by adding HTML buttons that will load new files when pressed:

< button type = "button“ value = "AE“
onclick = "buttonClicked (value) “ > Load AE </ button >

< button type = "button“ value = "AF"
onclick = "buttonClicked (value) “ > Load AF</ button >

Reload data function

Create a reload data function that takes an input value, does a string concatenation to build a filename, and runs d3.csv. This file can then call your chart's updateData function to re-sort variables and re-draw graphs, etc.

```
function reloadData ( inputName ) {

    d3.csv ( 'foodImports_' + inputName + '.csv' ,

        function ( error, newData){
            console.log(newData);

            //call your update function from here!!
            //updateData(newData);
    });
}
```

Button clicked function

Create a function to call the reloadData function and pass it the button value (which contains the string piece needed for the filename) when the button is clicked:

```
function  buttonClicked ( value ) {
   reloadData ( value );
}
```

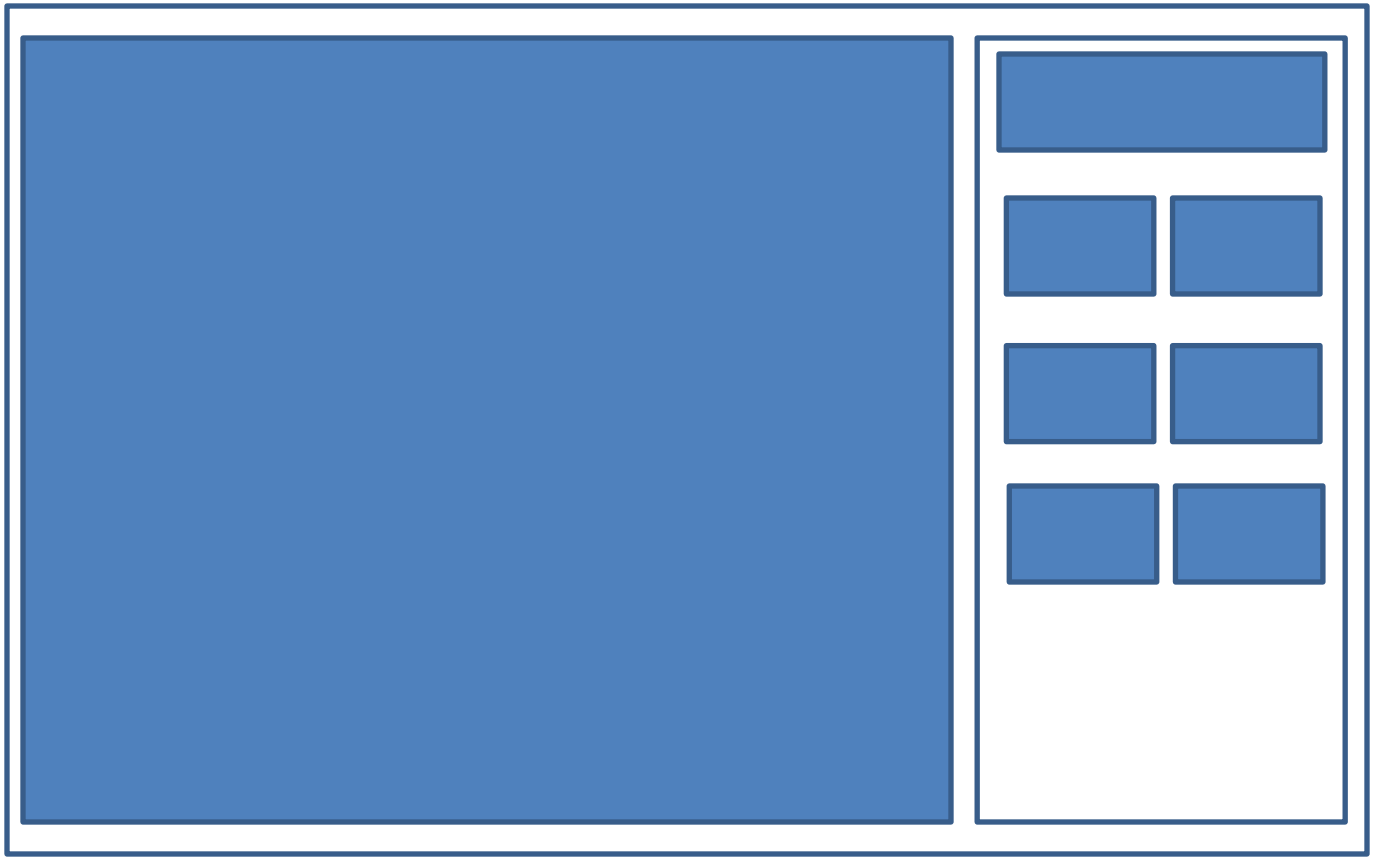
Note that you could do this same thing inside of a .on( ) event for a d3 element, from a slider or dropdown, or in a bunch of different ways. The important thing is to use some action to call reloadData, and then call your update function from *inside* it, not right after.

This will likely **not work** consistently, because updateData will run before newData loads, depending on the speed of your internet connection:

```
newData = reloadData ( value);
updateData (newData);
```

Miscellaneous item #3: Populating HTML elements from D3

In some cases, you might want to use D3 to populate a div or HTML table rather than using a tooltip. Added a sidebar to clustered bubbles example that we'll program next week:



Update the HTML using Javascript

Create classes for each div, add styles in CSS

Give the divs that you want to update an id also

In the circle element, add a .on( ) event that updates the HTML elements:

```
.on( 'mouseover' , function ( d ) {
        document .getElementById ( 'title' ) .innerHTML  =  ( d.bank );
        document .getElementById ( 'rank' ) .innerHTML  =  ( d.rank );
        document .getElementById ( 'assets' ) .innerHTML  =  ( d.assets );
        document .getElementById ( 'country' ) .innerHTML  =  ( d.country );
});
```

Miscellaneous item #4: Wrap text

D3's default text items don't wrap text nicely. Going back to the timeline example from a few weeks ago to demo word wrap code.

Code is from a Stack Overflow example. We need to edit it a bit to get it to work in this scenario.

Currently, the word wrap code takes in three arguments:

- A d3 selection that the text should go into
- The actual text that we want to use
- The width that it should wrap to

For our code example, we want to change this to take just the d3 selection and the width, since the text label information is already stored in our data.

Update wrap function

First, update the wrap function itself:

```
wrap ( text, width ) {

 text.each ( function ( ) {
       var text = d3 .select ( this ),
        words =  text .datum( ) .text .split ( / \ s + / ) .reverse ( )…
```

In this case, text is a selection of a single DOM element. Use the .datum( ) command to access the data stored inside of the DOM element. The original .csv file uses "text" as the name of the column that contains the label text, so we use that property to get out the text from the file.

Now, add a class to the text objects that we're adding to the DOM, so that we can select them and pass them to the wrap function.

Call the wrap function from d3

Now, actually use the wrap function on a text element drawn in d3.

First, add a class to the text objects whose content we want to wrap, so that we can select them and pass all of the objects to the wrap function.

```
lines .append ( 'text‘ )
      .attr ( 'class‘ , 'textBox‘ );
```

Then, call the wrap function, pass it the selection containing all of the text elements, and tell it how wide you want the final text to be:

```
wrap ( d3 .selectAll ( '.textBox‘ ) , 100 );
```

Miscellaneous item #5: Resizing tooltips

Sometimes, you might want to set either the width or height of your tooltips to be constant, and change the other dimension.

Go back to our tooltip example from week 6, using the Bootstrap version. Added some dummy text to the .csv file for testing.

First, update the tooltips to use the tooltipText attribute instead of the data values.

```
.attr ( 'title‘ ,  function ( d ) {
        return d .tooltipText ;
} );
```

Bootstrap automatically creates tooltips with fixed width, and adjusts the height to fit the text.

Styling tooltips

We can adjust the style of Bootstrap tooltips using CSS, using the .tooltip selector.

```
.tooltip-inner{
      width: 50;
}
```

If you want it to use a range of sizes, you can give it a min-width and a max-width.

If you want your tooltips to be always the same height, set a height value (notice that this will cause problems when there's too much text to fit in the box, if you don't also allow it to increase the width)

```
.tooltip-inner {
     min-width : 100 px ;
     max-width : 100 % ;
     height : 50px ;
}
```

Miscellaneous item #6: Multiple files

Sometimes, it would be nice to split one long piece of code into several smaller files instead. Fortunately, that's easy to do in Javascript.

Create new empty files for the code to live in:

draw.js
HTMLfunctions.js

Copy and paste relevant code into the files

Link the files (in the correct order!!) in the HTML file:

< script src = "./HTMLfunctions.js“ >< / script >
< script src = "./draw.js“ >< / script >
< script src = "./index.js“ >< / script >

Caution: Be aware of load order and variable scope!

Javascript files that are called from the same HTML document can "see" one another. If you declare a global variable in one, it will be accessible in all of them.

Like all HTML, scripts in the document are read from top to bottom. nestedData is defined in index.js. Put a console.log( ) statement at the top of draw.js.

Put the same statement inside the drawPoints function, and run it again.

loadData is defined inside of the d3.csv function in index.js. Try console.logging loadData from draw.js.

D3.csv slows index.js down enough that load order isn't a problem; in some cases, using different file orders in the HTML will cause errors. To illustrate, add a call to drawPoints ( ) at the beginning of index.js, and move it to the top of the HTML scripts list.