Dynamic D3 Charts
Joins, Interaction & Transitions
Sources of Change/Dynamics

- User Input (interactions)
- Animation/Transitions
- New Data

http://www.youtube.com/watch?v=hVimVzgtD6w
When binding data to a selection there are three possibilities:
1) The selection has less elements than the data – enter set
2) The selection size is equal to the data size – update set
3) The selection size has more elements than the data – exit set

Data Joins allow us to deal with all three situations, declaratively (what) with no looping and conditional structures.
First, `selectAll` finds all existing text elements and matches data to text elements 1 to 1 in order given (data and found elements)

The results overlap determines the enter, update, and exit sets.

First, what happens if selection has less elements than data: `enter()` [data is entering the chart!]
1) appends the text elements
2) sets the class to enter
3) sets x positions as function of index of data element, \( I \)
4) sets the shift in y direction

What happens if selection has more elements than data : `exit()` [data is leaving the chart]
1) remove the element

What happens if the selection has same number of elements as the data - neither exit nor enter is called, so it’s just an ‘update()’ of old elements
1) the `text.attr` line before `enter()` set’s the class to ‘update’ for all existing elements with `data`
First, `selectAll` finds all existing text elements and matches data to text elements 1 to 1 in order given (data and found elements)

The resulting overlaps determine the enter, update, and exit sets.

First, what happens if selection has less elements than data: `enter()` [data is entering the chart]
1) appends the text elements
2) sets the class to enter
3) sets x positions as function of index of data element, I
4) sets the shift in y direction

What happens if selection has more elements than data: `exit()` [data is leaving the chart]
1) remove the element

What happens if the selection has same number of elements as the data - neither exit nor enter is called, so it’s just an ‘update()’ of old elements
1) the text.attr line before enter() set’s the class to ‘update’ for all existing elements with data
Notice how position of all updates are always at beginning, so don’t have to ‘recompute’ them. Only have to recompute text every time because the text value (characters) for elements changes. e.g. b h i l replaces a b c in the second arrival. Positions don’t change, just character values.
Only difference here, is that a key is provided for the binding. It’s the identity key which means that we no longer map the first datum to the first element, we use the datum to tell us which element it maps to. So, since they are letters of the alphabet, the datum ‘g’ will will index the ‘g’ element.

Update set: all characters that are in the new dataset that are not already represented with elements
Enter: all characters that are in the data set that are not represented with elements
Exit: all elements that are in the selection that are not represented in the data

What else has to change?

NOTE: Mapping of character to text value is set on enter and will not change. ie. an elements ‘text’ value stays the same …fourth element always holds the value ‘c’

So, we only need to set the ‘text’ attribute on enter()

However, positions change because we no longer just overwrite, but essentially insert values where they belong, so we need to compute positions of all currently existing elements (entered and updated!) based on index i.
Notice how had to update the position of h

Here, in third arrival. b and h arrive and are already there so don’t ned to update their text, however their positions move (ie. they have new index in a larger string) so must update them!
In this case,

the text selection is given the update class, then it is set to transition over 750 ms....all subsequent operators (e.g. .att in this case) happen over that 750 ms.

Here, the x values of the update() elements will move over 750ms. So, all black elements which represent updates will only slide left/right
Transition/Animation

- Use judiciously....they are attention getters
- Don’t accidentally create a masking effect
- Don’t animate over many states (data values...) they are hard to follow and compare. (unless Hans Rosling is talking over your vis!)
Entering elements get their class, and therefore green color, dy, and starting y position of -60 and proper x position, an opacity and text value.

We then start a transition over 750 ms to drop them into place so we set the y attr to 0 and opacity to 1...they'll drop to 0 and go from transparent to solid!

For exit case, we’ll color them red then start a transition from 0 to 60 and make them completely transparent before we remove them.
Side Note: Transform

- Here, we transform the group by half the height, essentially making the halfway mark the zero y value
Transitions – Exiting Elements

• start at zero y in black
• turn red, move down, fade out

```javascript
text.exit()
  .attr("class", "exit")
  .transition()
  .duration(750)
  .attr("y", 60)
  .style("fill-opacity", 1e-6)
  .remove();
```

Entering elements get their class, and therefore green color, dy, and starting y position of -60 and proper x position, an opacity and text value.

We then start a transition over 750 ms to drop them into place so we set the y attr to 0 and opacity to 1...they’ll drop to 0 and go from transparent to solid!

For exit case, we’ll color them red then start a transition from 0 to 60 and make them completely transparent before we remove them.
Entering elements get their class, and therefore green color, dy, and starting y position of -60 and proper x position, an opacity and text value.

We then start a transition over 750 ms to drop them into place so we set the y attr to 0 and opacity to 1...they'll drop to 0 and go from transparent to solid!

For exit case, we’ll color them red then start a transition from 0 to 60 and make them completely transparent before we remove them.
selection.on(type, callback)

- callback is a function
- type can be one of many....

Another Example

Useful Tutorials/Reading

- Thinking with Joins: http://bost.ocks.org/mike/join/
- General Update Pattern I: http://bl.ocks.org/mbostock/3808218
- General Update Pattern II (key functions): http://bl.ocks.org/mbostock/3808221
- General Update Pattern III (Transitions): http://bl.ocks.org/mbostock/3808234