Week 5 Tutorial - Animation with SVG and Raphael.js

This challenge set comes with starting template code for your convenience.

Learning Outcomes

- Conditional execution

```
if (a === b) { // '===' test for equality
  do these statements
} else { // else is optional
  do these other statements
}
```

- State variables for remembering information between function calls
- Basic interactive SVG graphics with Raphael

But first a note

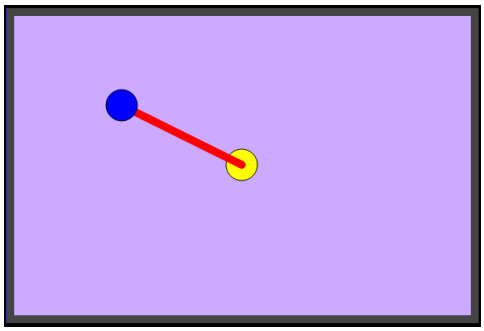
Raphael has its own way of listening to mouse events. For example,
raphelement.click(handler)
listens for "clicks" on element, and calls the handler function when it receives them. Similarly, each event has its own method (raphelement.mousedown, raphelement.mouseup, etc).

However, if we grab the ' node ' property of the Raphael graphical objects, we get the SVG object that behaves just like other DOM objects. Then we can add listeners in the way we know already:

```
raphelement.node.addEventListener(event, function(ev){...});
```

This is the way we will do it in this course. Don't use Raphael event functions!

Challenges



1. The paper is not a graphical object. In order to color it, add events, etc, we fill it with a rectangle.
   a. Get a rect from the Raphael paper so that it fills the whole paper.
   b. Give it a fat colored border using the 'stroke' and 'stroke-width' attributes.
   c. Using the 'node' property of Raphel graphical objects, add an event listener for 'mousedown' and print out a message to the console when the events occur.
   d. Run your code in the browser. What happens as you click around? What happens iif you click on the border? Can you think of an explanation for this behavior?
   e. Fill your rectangle, and run again clicking around. Now what happens? Can you explain this now?
2. Draw a line on the canvas, with one end in the center of the canvas (note I've already provided variables for the canvas dimensions).
   a. Make the line colored and thick.

3. Draw 2 circles centered on each end of the line.
    a. Make the circle that is NOT on the center of the page to have the color blue.
    b. add mousedown and mouseup listeners to the blue circle, printing console messages on the events to make sure everything is working.
4. Next let's make the blue circle (and that end of the line) draggable.
    a. Create a "state variable" to remember whether the mouse is down or up on the blue circle, and set it apprporiately in the mousedown and mouseup listeners.
    b. Add a mousemove listener to the blue circle, and make sure the callback function takes an argument to hold the MouseEvent that the system will pass it whenever the event occurs.
    c. Get the mouse coordinates (the offsetX and offsetY properties of the MouseEvent passed to your callback function) and print them out to the console.
    (check your code inthe browser)
    d. If the mouse is pushed on the blue dot, then use the mouse coordinates to set the new cx and cy attributes of the circle
    (check your code in the browser)
    e. Also update the "path" attribute of the line to that one end of the line is always at the center of the blue circle.
5. What happens if you try to drag the blue circle too quickly? Why? Can you think of a fix?
6. Add a mousemove listener to your background rectangle that does the same thing as the mousemove listener on the blue circle.
    Now try to move your mouse quickly.

Bonus round) What happens when you try to drag your blue circle over the center circle? Why? Can you think of a fix? Hint: Graphical objects are drawn in the order they were created.

Double Bonus Round) Animate the center circle between two sets of values for some attributes. Hint: the raphael animate function can take three arguments: the target attributes, the time it takes to reach them, and a function to call when it is finished animating. Hint #2: define two different functions that call the animate method on a raphael element with different attribute sets....

...