Tutorial challenges

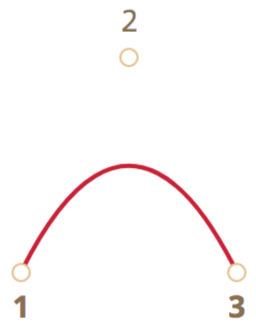
1. **Make 3 circles with Raphael**

Instead of repeating all of our code for each circle we want to make, let's take advantage of the "modularity" of functions, and write a function that creates and returns new circles to us. We can think of it as a "factory" function that manufactures circles.

   a. Write your factory function, 'mkCircle', to take 4 arguments: x, y, r, and f so that you can pass in values for the position, the radius, and the initial fill color for the circle you would like to have made.
   b. In the body of your function, call the Raphael function for creating circles and assign the object it returns to a local variable, 'newcircle'.
   c. Next give the circle the color requested by setting the Raphael fill attribute of the circle to the value passed in to your mkCircle function.
   d. Don't forget to return the circle your factory function makes!
   e. NOW you can create your 3 circles (let's call them c1, c2, and c3), by calling your factory function with the values you like. Make them different colors and spread them out over different locations. I also suggest a radius of at least 25.
   f. Check your work in the browser!

2. Print the cx, cy, and r attributes of one of your circles to the console. You can get attributes of any Raphael object using it's attr method:
   a. For example, foo.attr('bar') returns the value of the bar attribute of the Raphael object foo.

3. **Bezier Curves are defined by 3 points:**



To make a Bezier curve in SVG and Raphael, you use a path string with the middle point command label of 'Q'. So, if p1={0,100}, p2={50,0}, and p3={100,100}, the path string for the Bezier Curve would look like this: "M 0,100, Q 50,0 100,100) " (the last point doesn't even need a command label).

   a. So, let's write another function, make3ptpathstring(), that takes 3 arguments assumed to be Raphael circles. The function should get the cx and cy values from each of the 3 circles passed to the function, and create a pathstring from them representing a Bezier curve. Make sure the function returns the string you created!

b. Next, create an empty Raphael path object (paper.path("")) and assign it to a variable, let's call it 'mypath'.
c. Then, set the 'path' attribute of mypath. to the string that gets returned by make3ptpathstring when you pass it your 3 circles.
d. Check your work – do you get a nice Bezier curve on drawn in your browser?

4. **Now we want to make our circles draggable. You are a pro at this already! As you know, that means keeping track of the state of the mouse press on each circle, adding event listeners to each circle for mousedown, mouse up, and mousemove (like we did in the homework for this week). Oi, who wants to type a huge chunk of almost-identical code 3 times? Not me. Luckily, we can use our circle factory, 'mkCircle', to do this work, and thus only have to write the dragging code once!**

   First, recall that to add event listeners to Raphael objects (e.g. foo) we addEventLister to their HTML 'node' attribute: (foo.node.addEventListener(…)

   We will need to access to our circles in the callback functions (to check their state and to move them).

   a. Before writing our 3 Listeners, notice that you can not use 'newcircle' inside your callback functions. Talk with your mates to understand why! (This is important and your understanding and will be quizzed!)

   So how will we access our circles in the callback functions then? The event object passed in to your callback function has tons of info on it, including a 'target' attribute that is the object that was clicked! Yay! ….. But wait, if 'foo' is our Raphael object, and if we addEventListener on foo.node, then the event.target will be foo,node, not the Raphael object foo that we want. Boo!

   But wait, we can just store our Raphael objects (e.g. newcircle) on newcircle.node when we create newcircle, so that if we have the node, then we can get our circle! (Yay! There is always a way!)

   b. In mkCircle, create an attribute on your newcircle.node to store newcircle, let's call it raphcircle. Just use this line of code:
      i. newcircle.node.raphcircle = newcircle

   Now, inside any of our callbacks for mouse events, we can use the event object to get our Raphael object. If we name the event object argument 'ev', then ev.target.raphcircle is our Raphael circle!

   OK, let's write the circle dragging code.

c. In your circle factory function, before the circle is returned, add a "state" attribute to the circle object, and initialize it to "up".
d. Add event listeners for mousedowns and mouseups, to your new circle that change the state attribute we created earlier to its proper value. Make sure to provide the argument for the event that will get passed in to your callback by the system.
    i. First use the event object to get the circle from the node.
    ii. Now set your state attribute to the new value ("up" or "down").
    iii. **Check your work** to make sure all of your circles are generating callbacks on mousedowns and mouseups with console.log statements in your callback function.
e. Finally, add the eventListener for mousemove and move your circle when it's state is "down" and the mouse is dragged. This is exactly like your homework, except for having to first get the circle from ev.target that you stored on the node.

5. **Now that a circle has moved, you need to update the path attribute of your Bezier curve, mypath**.
    a. Expand your understanding of Bezier curves by dragging your circles around.

BONUS ROUNDS

1) Have you noticed that sometimes your mouse ups don't seem to register? Have you figured out why that happens? Think about it before reading on.

It is because your mouse up event can happen on your Bezier curve since the curve is drawn on top of the circle. When that happens, the circle doesn't generate that event. There are (as always) many solutions. One is to prevent your Bezier curve from catching events:

```
mypath.node.setAttribute("pointer-events", "none");
```

2) You probably also noticed that if you move your mouse fast, you can lose the dragging on your circles. This is for exactly the same reason as you saw on your homework. You can fix it by listening to events on the background, but the solution is a little more involved because you have three circles. Thus you need to first figure out when circle is in the "down" state and then move that one.

To do this with minimal code, take core code for moving your circles and path out of the circle mousemove listener and put it in a "move" function that takes a circle as an argument. Then you can call move() passing in whichever circle needs to be moved. You could even "tighten up" your code by replacing the code in your circle mousemove code with a call to your move() function.