REVIEW tutorial challenges

1. Draw a SVG object from a path string.

In the starting template code, you'll find a path string defined.

mybigheart="M340.8,98.4c50.7,0,91.9,41.3,91.9,92.3c0,26.2-10.9,49.8-28.3,66.6L256,407.1L105,254.6c-15.8-16.6-25.6-39.1-25.6-63.9c0-51,41.1-92.3,91.9-92.3c38.2,0,70.9,23.4,84.8,56.8C269.8,121.9,302.6,98.4,340.8,98.4z"

(Did you know? You can rip paths from svg images you find on the web. You can also create SVG paths drawing freehand in Inkscape (and other apps)!)

mybigheart is just the SVG path string at this point, not a graphics object.

1. a) Use it to actually draw a heart by using the 'path' function which is a property of the Raphael paper. Pass it the path string as an argument, and store the results in a variable myHeart1.
(Paper.path documentation: https://www.vincentbroute.fr/mapael/raphael-js-documentation/index.html#Paper.path).

1.b) Give it some attributes using the attr function which is a property of all Raphael objects. Recall that the attr function takes an object of attribute /object pairs.

2. The transform attribute

https://dmitrybaranovskiy.github.io/raphael/reference.html#Element.transform

Raphael objects have a property called transform. You can set it (or animate it!) like any other attribute:

```
object.attr({transform : val})
```

Its value is a string with letters (S, T, and R, for scale, rotation, translation) and numbers. The meaning of the numbers depend on the letters they follow (like for path).
S – followed by one number specifying the scale magnification factor. Anything less than 1 shrinks the object, anything bigger than 1 magnifies it.
T – followed by 2 numbers: translation in x, translation in y
R – followed by a number specifying rotation in degrees

Thus "S2T100,50" would mean a transforming to double the size, shifted 100 to the right, and 50 down)

Explore! Try various translations, rotations, and scaling of the heart.

2.a Now, instead of using .attr() to set the transform string, use .animate!

Note: Transform doesn't change the path attribute. Every time it the object is drawn, it starts with the path attribute and *then* transforms it.

3. Rapahael.transformPath

Unfortunately, paths are "hard coded". They don't have 'x' and 'y' attributes like rectangles, or 'cx' and 'cy' attributes like circles and ellipses that can be used to set their location. That means, we actually have to change each and every point in the path if we want to move it. Argh.

Luckily, Raphael provides is a method for doing that, Raphael.transformPath.
It takes 2 arguments: a path string, and a transform string.
It returns a new string you can use as an argument for creating new Raphael path objects. [1]

SO – make a few more objects based on transformations of the mybigheart path. Give them different attributes.

One way this is useful: Moving path objects around your screen. Since transforms are always relative to the path string, the if (and only if) your object is "located" at (0,0) then the two transform T values will correspond to the actual location on the screen (acting like x,y or like cx,cy). Thus if you want to move a path this way *first* use **Rapahael.transformPath** to move it to the top left corner of the screen (or draw it there in the first place). Then you are good to go!

[One hour tutorial can end here … For two-hour tutorials, continue below!]

4. Multi-stage animation

Now, do a 2 (or more!) -stage animation.

Write one function (call it foo) that animates the transform attribute specifying new values for scale, transform, and/or rotation, and a second function (call it bar) for specifying different values.

Remember: The Raphael object animate method can take 3 arguments: the object specifying attributes to animate with their new values, the number of ms the animation will last, and the last one is the name of a function to call when the animation is complete.

Thus obj.animate(arg1, 1000, bar) would animate the obj to the attributes values specified in the arg1 object, it would last one second, and would then call bar.

4.a Use your functions to animate multiple objects all at the same time! (You could "synchronize" their animations by putting the animate calls in the same

---

[1] It actually returns an array, or list of strings, which you can make into a regular string with the .toString() method on the array object – nice for viewing in a console.log message, for example.

functions you have already written, or you could write new animation functions for the new objects). Careful how you use the callbacks that get called when the animation finishes! Don't start a million animations or your computer will freeze!

5. Still have time to explore?

Make the animations depend on some kind of interactivity (though I do not recommend using the mouse to position an SVG path object – save that for Challenge 6). For example, change the speed of rotation with a slider or a mouse click, or any of a billion things of that nature).

[You should be perfectly capable of accomplishing this next Challenge, and some may find the solution useful for their own app coding, but you are not required to know bounding boxes and transform paths for this course.]

6. Mega-bonus: path "position".

Background to this challenge: transform, rotate, and scale all change how a graphical object is **drawn** but not its properties such as x, y, (cx, cy), height, width, path, etc. So what if you want to actually change the "location" of a Raphael path object? As we said, it doesn't have 'x' and 'y' (or 'cx and 'cy') that you can change to give it a new location. The whole (possibly very long) path property string has to be changed! But how?

So here is the challenge: create a Raphael path object (or use one of the hearts you created already), and move it to a location (by changing its path attribute) determined by a mouse click on the background.

Hints:

a) SVG objects have a "bounding box," which is the smallest box completely enclosing the path. You can get that box from a Raphael object using the method .getBBox(). You can use the bbox properties x, y, width, and height, to calculate the center of the box.

b) Your goal is to create a new path string that is shifted by the difference between your mouse click location and the center of the bounding box. So
   a. compute that difference (let's say we call the values diffX and diffY). Use diffX and diffY to create a transform string ("TdiffX,diffY"),
   b. Now get the path string from the object you want to move (to get an attribute value from a Raphael object, pass its name to the .attr() method).

c) Now you have the two arguments you need to pass to Raphael.transformPath() to generate the new path string you desire!

d) The value that Raphael.transformPath() returns can be used with the .attr() method or the .animate() method to set a new value for the path attribute on your object.