```
        // Your implementation here, where callbackFunction
        // is called as follows once the animation concludes:
        callbackFunction();
    };
```

24. Add the following animation effects to the tic-tac-toe case study shown in Section 6.7:

    - "Fade-ins" for X's and O's as the player clicks on the grid
    - "Fade-outs" for X's and O's as a new game is set up
    - Any sort of "ending animation" (color changes, movement, etc.) upon the conclusion of a game

    If you did any of the preceding animation-workalike exercises, you may use what you wrote there to make this exercise easier.

25. Write short JavaScript `canvas` programs that draw the following on a `canvas` element of your choosing. Exact dimensions, positions, and color values are up to you, as long as what you draw corresponds reasonably to the plain English descriptions:

    (a) A blue square at the center of the `canvas`

    (b) A black border surrounding the perimeter of the `canvas`

    (c) A 50% translucent red rectangle overlapping a 50% translucent green rectangle

    (d) An orange "X" whose lines span the upper-left to lower-right corners and the lower-left to upper-right corners of the `canvas`, respectively

    (e) A solid, brown hexagon

26. Write short JavaScript `canvas` programs that draw the following on a `canvas` element of your choosing. Exact dimensions, positions, and color values are up to you, as long as what you draw corresponds reasonably to the plain English descriptions:

    (a) A grid of lavender squares, one `canvas` pixel apart, filling the entire `canvas` (there is more than one approach to drawing this)

    (b) A "graph paper"-style grid consisting of light green lines that fills the entire canvas (again, there is more than one approach)

    (c) A honeycomb pattern at least three hexagons across and three hexagons down

    (d) A polka-dot pattern with pink dots on a brown background

    (e) A simplistic number "8" consisting of overlapping purple circles

27. Write short JavaScript `canvas` programs that draw the following objects on a `canvas` element of your choosing. Exact dimensions, positions, and color values are up to you, as long as what you draw corresponds reasonably to the plain English descriptions:

    (a) A "fake 3D" green wireframe cube at the bottom right of the `canvas`

    (b) A "fake 3D" solid cube, with its three visible faces colored in varying shades of gray, at the top center of the `canvas`

    (c) Reasonable facsimiles of a baseball, a golf ball, and a tennis ball, painted with gradients for a 3D effect

    (d) A yellow smiley face with a radial gradient to give it a faux spherical effect

    (e) A ringed planet, painted with gradients for a 3D effect

28. Write short JavaScript `canvas` programs that draw the following "scenes" on a `canvas` element of your choosing. Exact dimensions, positions, and color values are up to you, as long as what you draw corresponds reasonably to the plain English descriptions:

    (a) A simple sunset scene, with a reddish sun setting into a green horizon under a gray-blue sky

    (b) A similar sunset scene as part (a), but with the sun setting into a dark blue "ocean" horizon and with a partial reflection showing on the ocean surface

    (c) A red "sphere" (i.e., a circle with a radial gradient) and the fake 3D solid cube from Exercise 27b, with recognizably shaped gray "shadows" underneath

    (d) Two stick-figure people, one wearing a black hat and another with long hair

    (e) A simple skyline scene, where black buildings with yellow-lit windows are set against a dark blue sky (*Tip:* Try using a loop that draws buildings with random sizes and window counts from left to right.)

29. Implement a simple pixel-based paint program web page using the `canvas` element. Allow the user to choose colors and brush sizes. Color and brush size selection may be implemented outside of the `canvas`, using buttons, dropdown menus, or other appropriate web page elements with corresponding event handlers.

30. If you have access to a touch event-capable web browser, implement a painting web page as in the previous exercise, but have it respond to touch events rather than mouse events.

31. If you have access to a multitouch event-capable web browser, enhance the touch-capable painting web page from the previous exercise so that multiple touches generate multiple simultaneous brush strokes, based on the placement and location of the user's fingers.