Session 6 Tutorial

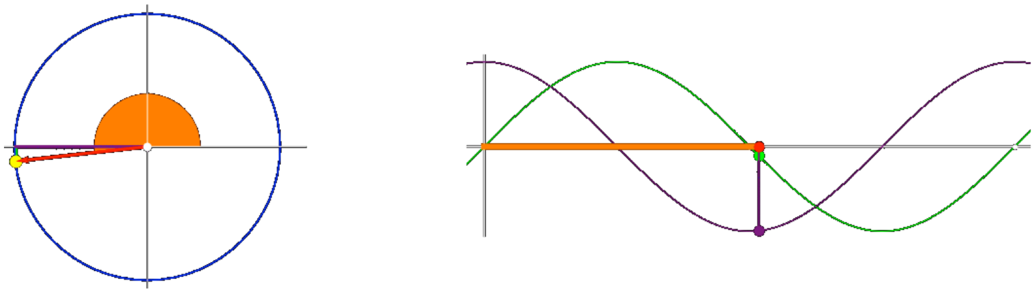
Timer animation - the real thing

The template project to start your coding with is in: **timeranimation.zip**. Download it and put it in a convenient place.

Learning Outcomes

- setInterval
- Math.PI, Math.sin, Math.cos

So first a note



Angle in [0, 2Pi] (orange), cos in [-1,1] (purple), sin in [-1,1] (green)

Goal: In this tutorial, we are going to animate a dot to move around the page in a sinusoidal fasion.

Challenges

1. Notice the map() function provided in main.js. Thought challenges:
    a. What happens when x is equal to a?
    b. What happens when x is equal to b?
    c. What happens when x is exaclty half way between a and b?
2. Draw a small circle in the middle of the paper
3. Create a draw() function that gets called periodically by a setInterval timer:
    a. create a variable called *frameLength* and initialize it to 100

    b. create a variable called *time* to keep track of the number of milliseconds since your page loads in to the browser, and initialize it to 0
    c. Write a function called *draw* (no arguments necessary) that increments *time* by *frameLength* and prints out a console message telling the time.
    d. Call the setInterval function so that it calls your ‘draw’ function every *frameLength* milliseconds.
    e. Check your work

4. Animate the dot to move back and forth in the x direction in a sine wave fashion:
    a. In the draw function, create a new variable called *a* (for "angle") and set it to *time* mulitplied by 2 Pi and divided by 1000.
    b. Think: what will *a* be when time is equal to 1000?
    c. Create another variable called *sa* and assign it to be the sin of *a*
    d. Change your console.log function to print sa
    e. Check your work. What is the range of the *sa* values being printed to the console?
    f. Next, use the map function to map the value of *sa* from [-1,1] into [0, width of the paper].
    g. Save that number returned by the map function in a variable, and use it to set the *cx* attribute of your dot
    h. Check!
5. Create a variable for controlling the rate of bouncing (number complete bounces per second) :
    a. Create a variable, *xrate* (outside the draw function, since we don't need to create it anew on every frame)
    b. Can you see how to use this variable to affect the rate of bouncing?
    c. OK, there are a few ways to do this, an easy one is to multiply the angle *a* by *xrate* inside the sin function.

If you got here, great - you have learned everything you need to from this session. The rest is "bonus".

6. Change the frameLength to make the motion smoother or choppier. Does the bouncing rate change? Why/why not? This is actualy a good insight to get for animation because sometimes you won't have control over the framerate.

7. Make the ball bounce in the y dimension, too. I suggest using the cos function.

   Now you are doing general purpose animation with timers (not using prepackaged behaviors provided by a graphics library). You can animate *anything*. With this particular code, you could easily create an analog clock simulation!