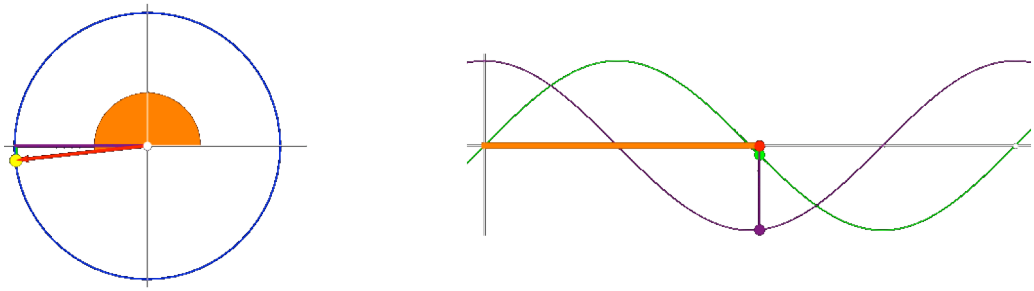
Timer animation - the real thing

Learning Outcomes

- setInterval
- Math.PI, Math.sin, Math.cos
- animation (the right way) with timers

So first a note



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

[see https://nm2207.org/creativeweb/tutorials/timeranimationweb/SinCos.html for animated explanation]

Note: We like functions that produce numbers bounded between known values, especially in [-1,1] because they are easy to map to other ranges (such as screen dimensions)! (Hint, hint).

Important: Raphael's animate() function was handy (just give it the goal values and the time you want to take to get there). But now **forget about it**! From now on we are doing animation the "real" way – by ticking a timer and computing for ourselves the incremental updates to graphical objects.

Challenges

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

1. Notice the map() function provided in main.js of the template. Run through the function "manually":
    a. What happens when x is equal to a?
    b. What happens when x is equal to b?
    c. What happens when x is exactly half way between a and b?
2. Draw a small circle in the middle of the paper

let timerID = setInterval(myFunc, num_ms) – starts a clock that calls myFunc every num_ms milliseconds. Notice the 'callback' function pattern that you are already familiar with! Here is an example:

```
let tickCount=0
let myTimer = setInterval( function() {
    console.log("Number of ticks is now " + tickCount)
    tickCount++
  }, 1000);          // 1000ms = 1 second
```

(Of course, the function could also have been defined and named elsewhere rather than passed as an anonymous function to setInterval)

setInterval returns a value that we think of as the timer ID which can be used to stop the clock when we want. Above, we use myTimer to hold the ID so we can stop that clock by calling:

```
clearInterval(myTimer)
```

3. We will create a draw() function (to be called periodically be a setInterval timer) for drawing a single 'frame' of the animation, and some variables for controlling the rate of frame updates and speed of the animated object:
    a. create a variable called *frameLength* and initialize it to 100 (our frames will initially be drawn every 100ms)
    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* multiplied by 2 Pi and divided by 1000.
    b. Think & talk: what will *a* be when time is equal to 1000? 2000? 3000? What happens as time goes up smoothly? Talk about this with your roomies and understand it in terms of the diagram (and animation) above!
    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? Do they match your expectation for the sin?
    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 actually a good insight to understand 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.

8. Make the x and y rates different from each other!

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