6 [ChapterStart]

Trigonometry

Trigonometry is the study of triangles. It starts with right triangles and the three ratios between the sides: sine, cosine and tangent. Then it generalizes to non-right triangles with the Law of Sines and the Law of Cosines. But this is seldom how trig functions are used in reality. In this chapter you'll see sine and cosine applied to oscillating motion, and you'll make some interesting, dynamic, interactive sketches. You'll apply the tangent function (and its inverse) to a problem of making a field of objects orient themselves to the location of your mouse.

Waves

Sines and cosines make waves when the height of a point on a circle is measured over time. Let's make a circle. Then we'll put a smaller circle on the circumference and as it travels around the circle, its height will draw out a sine wave.

Start a new Processing sketch and create a big circle on the left side of the screen like in Figure 6-1. Try it yourself before looking at the code below:

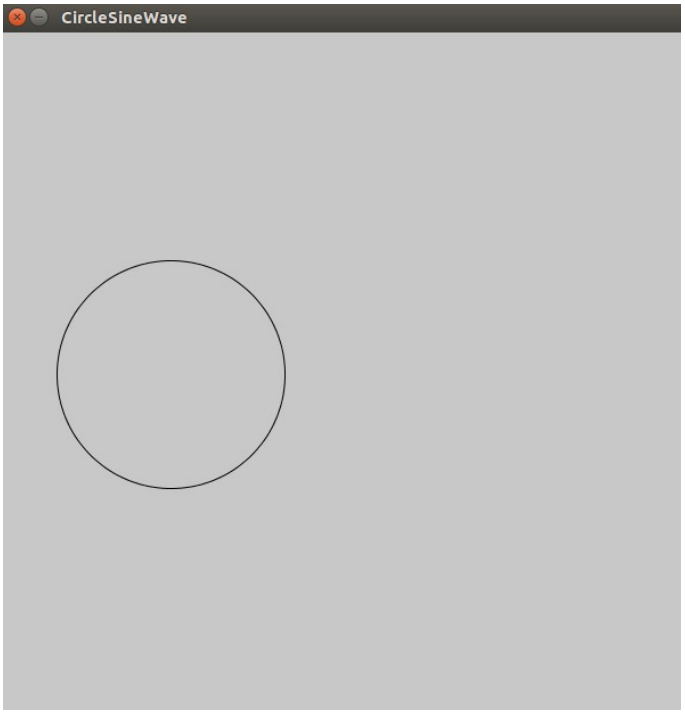


Figure 6-1: The start of the Sine Wave sketch

Here's the code:

```
r1 = 100 #radius of big circle
```

```
r2 = 10  #radius of small circle
t = 0 #time variable

def setup():
    size(600,600)

def draw():
    background(200)
    #move to left-center of screen
    translate(width/4,height/2)
    noFill() #don't color in the circle
    stroke(0) #black outline
    ellipse(0,0,2*r1,2*r1)
```

Now how do we make another circle spin around the big circle? Finding out the x- and y-coordinates of every spot on the circle isn't obvious, but it is obvious what the radius of the circle is, and all we have to do is rotate around the center. Using r as the radius of the circle and the Greek letter theta as the measure of the angle and using trig ratios, we can define a point on the circle as seen in Figure 6-2:

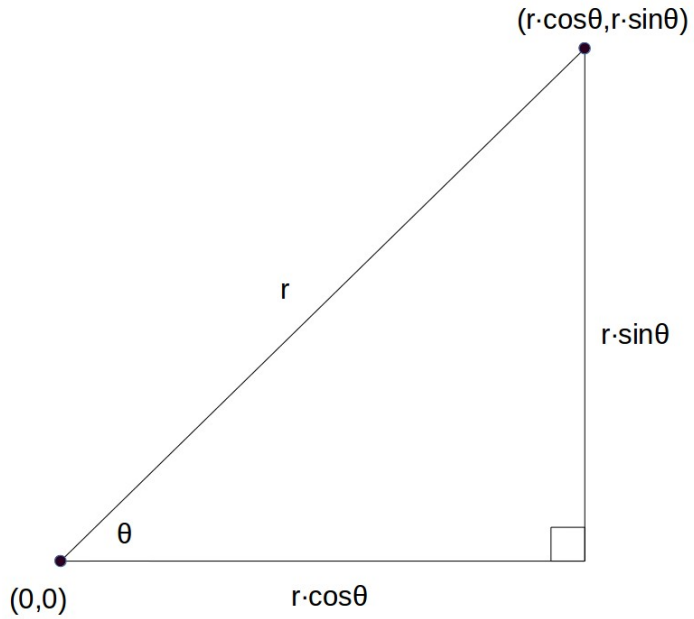


Figure 6-2

It might look more complicated than (x,y) but we only have to code it once and it'll work perfectly! Here's how to use sine and cosine to define the position of the ellipse that will rotate around the first circle.

```
    #circling ellipse:
    fill(255,0,0) #red
    y = r1*sin(t)
    x = r1*cos(t)
    ellipse(x,y,r2,r2)
```

At the end of the draw function, make the time variable go up by a little bit:

```
    t += 0.05
```

If you try to run this right now, you'll get an error message about “local variable 't' referenced before assignment.” Python functions have local variables but we want the draw function to use the global time variable t. So we have to add this line to the beginning of the draw function:

```
global t
```

Now you'll see a red ellipse traveling along the circumference of the big circle, as in Figure 6-3:

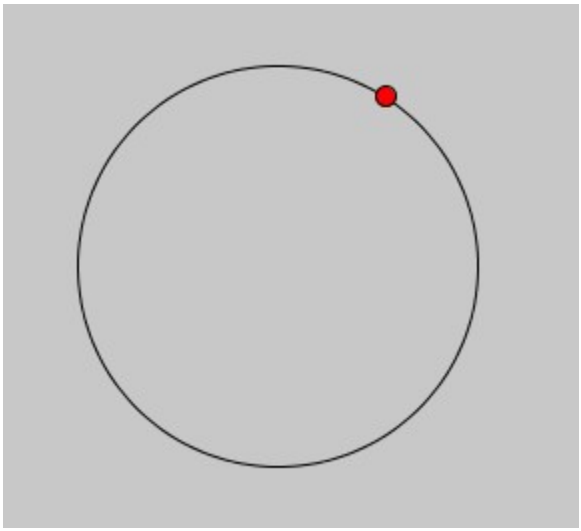


Figure 6-3: The red circle travels along the circumference of the big circle.

Now we'll chose a place over to the right of the screen to start drawing the wave. We'll extend a green line from the red circle to, say, x = 200. Add these lines to your draw function right before “t += 0.05.”

```
    stroke(0,255,0) #green for the line
    line(x,y,200,y)
    fill(0,255,0) #green for the ellipse
    ellipse(200,y,10,10)
```

Run your program and you'll see we've added a green circle that only measures the height of the red circle, as in Figure 6-4:

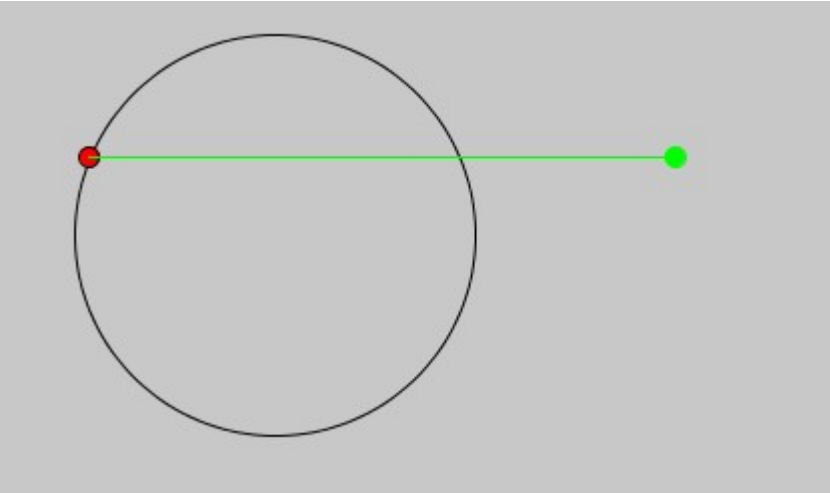


Figure 6-4

Now we want the green ellipse to leave a trail to show its height over time. Leaving a trail really means we save all the heights and display them all, every loop. How do we save a bunch of things? We need a list. Add this line to the variables we declared at the beginning of the program, before the setup function:

```
circleList = []
```

And add it to the “global” line in the draw function:

```
global t, circleList
```

After we calculate x and y in the draw function, save the y-coordinate to the circleList:

```
#add to list:
circleList.insert(0,y)
```

At the end of the draw function (before incrementing t) we'll put in a loop. It will loop through all the elements of the circleList and draw a new ellipse, to look like the green ellipse is leaving a trail.

```
#loop over circleList to leave a trail:
    for i in range(len(circleList)):
        #small circle for trail:
        ellipse(200+i,circleList[i],5,5)
```

A more “Pythonic” way of doing that is to use Python's built-in “enumerate” function. You'll see the same thing.

```
#loop over circleList to leave a trail:
for i,c in enumerate(circleList):
        #small circle for trail:
        ellipse(200+i,c,5,5)
```

This is the start of your Trigonometry class of the future! You made a sine wave, as you see in Figure 6-5:

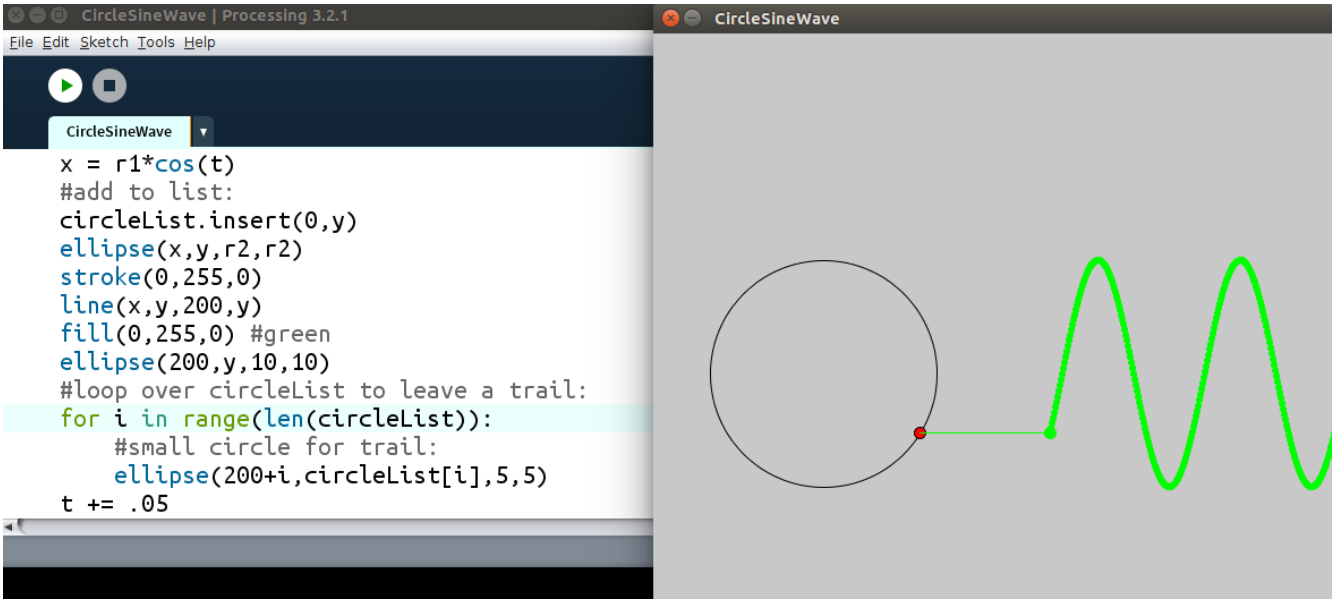


Figure 6-5: A sine wave

If you leave the program going for a while, you'll notice it starts to slow down. That's because the list of points is getting long, into the tens of thousands long! We can put a few lines of code in the draw function (right before the loop) to delete the first points if the list gets long enough:

```
if len(circleList)>300:
        circleList.remove(circleList[-1])
```

Our First Slice of Pi

What kind of math book makes you wait until Chapter 6 to get to pi? Pi is the ratio of the circumference of a circle to its radius. The Greeks and Babylonians noticed the ratio was just over 3. Three and a half? Not even close. Three and a quarter? Closer, but still a bit off. Three and an eighth? Very close. In fact, 3 1/7 stood in for pi for centuries in math classes before the calculator. How can we get closer? Just a little trigonometry. We're going to follow in the footsteps of Archimedes, Ancient Greece's greatest scientist, who used polygons inscribed in a circle to approximate pi.

(rotatng triangles sketch moved to Geometry)

Adding Sliders

At the top of the sketch, before the setup function, type this line:

```
from slider import Slider
```

This means “From the slider.py file, import all the functionality of the Slider class.” Next, before the setup function, create an instance of the Slider object. Of course, you can create more than one by giving them different names:

```
slider1 = Slider(1,120,90)
```

This means “Create a slider with a range of values from 1 to 120, and set it initially to 90. Call this 'slider1.'”

Inside the setup function, set the slider position to an x-y coordinate, in this case (20,20):

```
slider1.position(20,20)
```

The last thing is to assign the value of the slider to a variable and keep updating the value in the draw function. For example, I'm going to create a variable called “num” which will be the number of triangles I put in this sketch. Every loop, the num value will be set to whatever value the slider shows. Add this line to the draw function.

```
num = int(slider1.value())
```

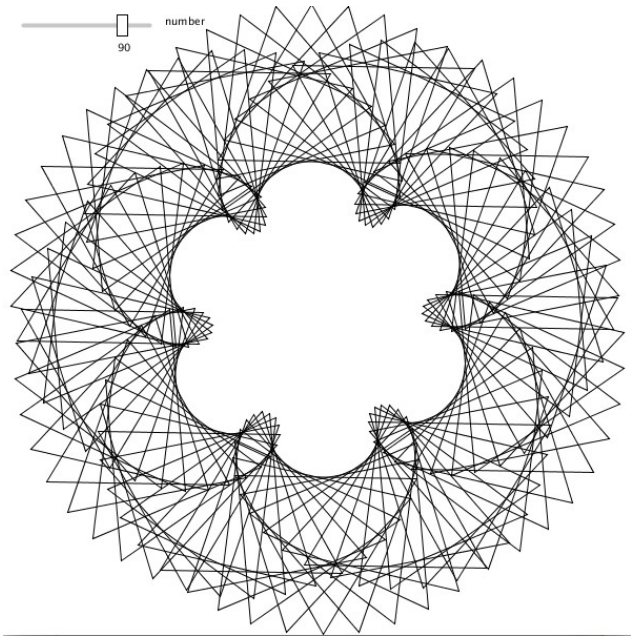
The “int” part makes sure the value assigned to “num” is an integer. That's important if you're using “num” in a loop, as we are in the Rotating Triangles sketch. “num” has to be an integer:

```
for i in range(num):
```

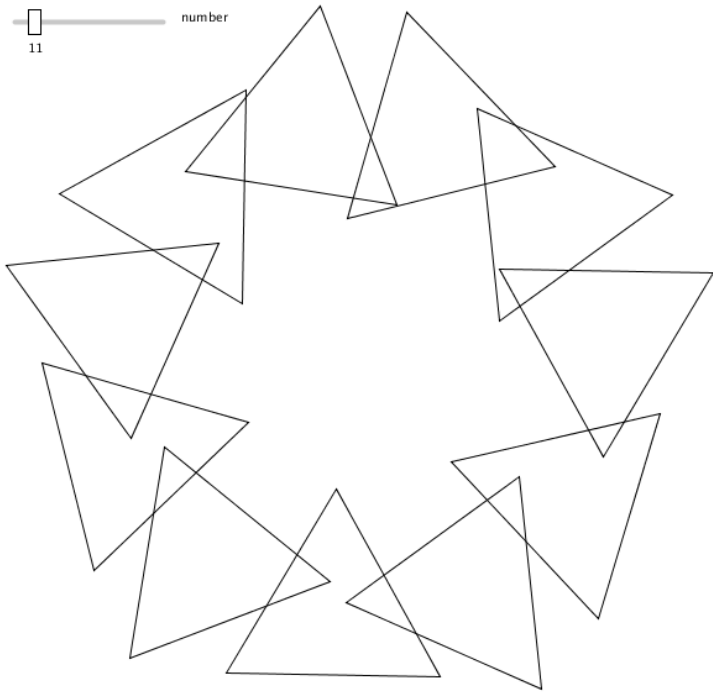
Optionally, you can add a label to your slider, to let your user know what they're changing. After the “position” line, add this line:

```
slider1.label = "number"
```

In the code, you have to replace all the places where you specified the number of triangles (it was 90) with the variable “num.” Now when you run the sketch, you'll have a bunch of rotating triangles. Initially, you'll have the “default” number, as in the figure below:



But when you move the slider you'll see the number update in real time:



Oscillating with Trigonometry

Another sketch that uses trig functions to oscillate back and forth is one by Dave “Bees and Bombs” Whyte. The interesting thing about this oscillation is that unlike our up-and-down sine wave in Figure 6.5, the wave oscillates in and out from the center. Start with the circle of circles template from the Geometry chapter in Figure 6.19:

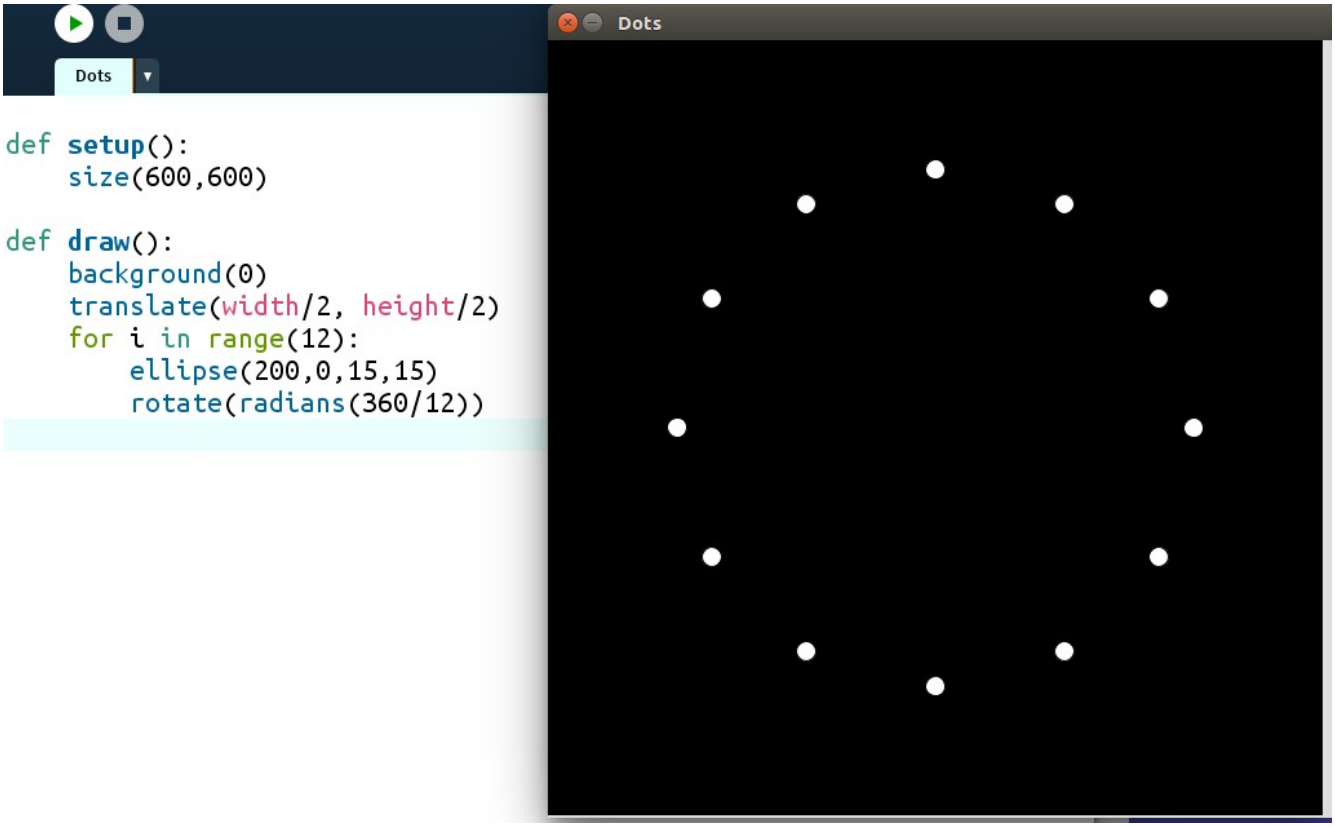


Figure 6.19

Now to oscillate the dots. Create a global time variable, use “sin(t)” in the ellipse location and increment time at the end of the draw function. The code now looks like this:

```
t = 0

def setup():
    size(600,600)

def draw():
    global t
    background(0)
    translate(width/2, height/2)
    for i in range(12):
        ellipse(120 + 90*sin(t),0,15,15)
        rotate(radians(360/12))
    t += 0.1
```

The dots are now oscillating in and out from the center. But they're moving in exactly the same way. Let's make them vary by their “i” value. Change the ellipse line to this:

```
ellipse(120 + 90*sin(t + i),0,15,15)
```

This makes each dot oscillate differently from its neighbors.