Waves

If you’re saying to yourself, “Um, this is all great and everything, but what I really want is to draw a wave onscreen,” well, then, the time has come. The thing is, we’re about 90% there. When we oscillate a single circle up and down according to the sine function, what we are doing is looking at a single point along the x-axis of a wave pattern. With a little panache and a for loop, we can place a whole bunch of these oscillating circles next to each other.

This wavy pattern could be used in the design of the body or appendages of a creature, as well as to simulate a soft surface (such as water).

Here, we’re going to encounter the same questions of amplitude (height of pattern) and period. Instead of period referring to time, however, since we’re looking at the full wave, we can talk about period as the width (in pixels) of a full wave cycle. And just as with simple oscillation, we have the option of computing the wave pattern according to a precise period or simply following the model of angular velocity.

Let’s go with the simpler case, angular velocity. We know we need to start with an angle, an angular velocity, and an amplitude:

```
var angle = 0;

var angleVel = 0.2;

var amplitude = 100;
```

Then we’re going to loop through all of the x values where we want to draw a point of the wave. Let’s say every 24 pixels for now. In that loop, we’re going to want to do three things:

1. Calculate y location according to amplitude and sine of the angle.
2. Draw a circle at the *(x,y)* location.

3. Increment the angle according to angular velocity.

```
for (var x = 0; x <= width; x += 24) {

    // Calculate y location according to amplitude and sine of angle

    var y = amplitude * sin(angle);

    // Draw a circle at the x, y location

    ellipse(x, y+height/2, 48, 48);

    // Increment the angle according to angular velocity

    angle += angleVel;

}
```

Let’s look at the results with different values for `angleVel`:

Notice how, although we’re not precisely computing the period of the wave, the higher the angular velocity, the shorter the period. It’s also worth noting that as the period becomes shorter, it becomes more and more difficult to make out the wave itself as the distance between the individual points increases. One option we have is to use `beginShape()` and `endShape()` to connect the points with a line.

```
angleMode = "radians";


var angle = 0;

var angleVel = 0.1;


background(255, 255, 255);

stroke(0, 0, 0);

strokeWeight(2);

noFill();


beginShape();

for (var x = 0; x <= width; x += 5) {
    // Here's an example of using the map function instead of an amplitude variable
    var y = map(sin(angle), -1, 1, 0, height);
    // Within beginShape and endShape, you use vertex to set the points of your shape
    vertex(x, y);
    angle += angleVel;
}
endShape();
```

The above example is static. The wave never changes, never undulates, and that's what we've been building up to. This additional step of animating the wave is a bit tricky. Your first instinct might be to say: “Hey, no problem, we’ll just let angle be a global variable and let it increment from one cycle through `draw()` to another.”

While it’s a nice thought, it won't work. If you look at the statically drawn wave, the right-hand edge doesn’t match the left-hand; where it ends in one cycle of `draw()` can’t be where it starts in the next. Instead, what we need to do is have a variable dedicated entirely to tracking what value of angle the wave should start with. This angle (which we’ll call `startAngle`) increments with its own angular velocity.

Here it is, with the start angle incorporated. Try changing the different numbers to see what happens to the oscillating wave.

```
angleMode = "radians";


var startAngle = 0;
var angleVel = 0.23;
var amplitude = 100;


var draw = function() {
  background(255);
  // In order to move the wave, we start at a different angle each time
  startAngle += 0.015;
  var angle = startAngle;


  for (var x = 0; x <= width; x += 24) {
    // Calculate the y location according to amplitude and sine of the angle.
    var y = amplitude * sin(angle);
    // Draw a circle at the x, y location
    stroke(0, 0, 0);
    fill(0, 0, 0, 50);
    strokeWeight(2);
    ellipse(x, y+height/2, 48, 48);
    // Increment the angle according to angular velocity
    angle += angleVel;
  }
};
```