Math 3: Trigonometry

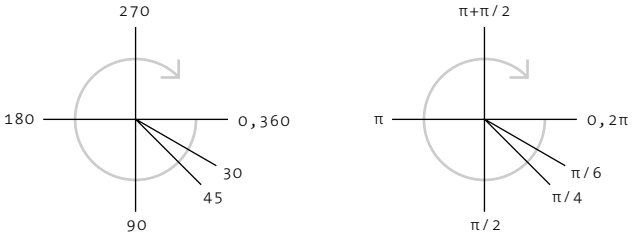
This unit introduces the basics of trigonometry and how to utilize it for generating form.

Syntax introduced:
`PI, QUARTER_PI, HALF_PI, TWO_PI, radians(), degrees()`
`sin(), cos(), arc()`

Trigonometry defines the relationships between the sides and angles of triangles. The trigonometric functions sine and cosine generate repeating numbers that can be used to draw waves, circles, arcs, and spirals.

Angles, Waves

Degrees are a common way to measure angles. A right angle is 90°, halfway around a circle is 180°, and the full circle is 360°. In working with trigonometry, angles are measured in units called radians. Using radians, the angle values are expressed in relation to the mathematical value π, written in Latin characters as "pi" and pronounced "pie." In terms of radians, a right angle is π/2, halfway around a circle is simply π, and the full circle is 2π.



Degree values Radian values

The numerical value of π is a constant thought to be be infinitely long and without a repeating pattern. It is the ratio of the circumference of a circle to its diameter. When writing Processing code, use the mathematical constant `PI` to represent this number. Other commonly used values of π are expressed with the constants `QUARTER_PI`, `HALF_PI`, and `TWO_PI`. Run the following line of code to see the value of π to 8 significant digits.

```
println(PI);  // Prints the value of PI to the text area
```
14-01

In casual use, the numerical value of π is 3.14, and 2π is 6.28. Angles can be converted from degrees to radians with the `radians()` function, or vice versa using `degrees()`.

This short program demonstrates the conversions between these representations:

14-02

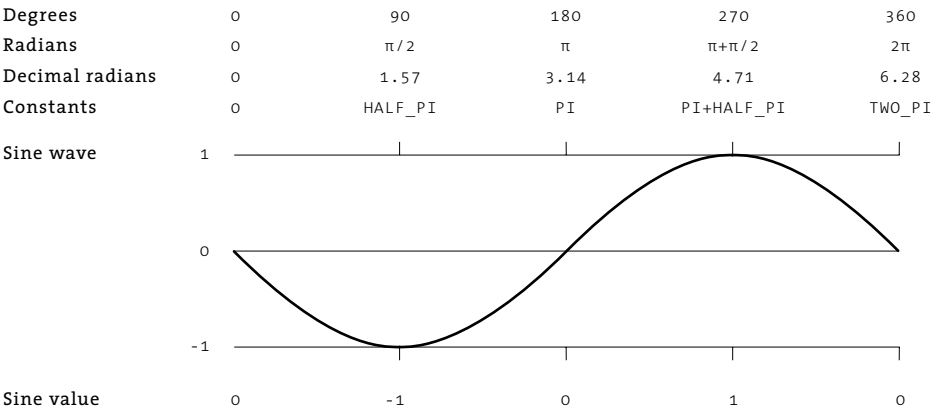
```
float r1 = radians(90);
float r1 = radians(180);
println(r1);  // Prints "1.5707964"
println(r2);  // Prints "3.1415927"
float d1 = degrees(PI);
float d2 = degrees(TWO_PI);
println(d1);  // Prints "180.0"
println(d2);  // Prints "360.0"
```

If you prefer working with degrees, use the `radians()` function in your programs to convert the degree values for use with functions that require radian values.

The `sin()` and `cos()` functions are used to determine the sine and cosine value of any angle. Each of these functions requires one parameter:

```
sin(angle)
cos(angle)
```

The *`angle`* parameter is always specified as a radian value. The values returned from these functions are always between the floating-point values of –1.0 and 1.0. The relationship between sine values and angles are shown here:



As angles increase in value, the sine values repeat. At the angle 0.0, the value of sine is also 0.0, and this value decreases as the angle increases. When the angle reaches 90.0° (π/2), the sine value increases until is it zero again at the angle 180.0° (π), then it continues to increase until the angle reaches 270.0° (π + π/2), at which point it begins decreasing until the angle reaches 360.0° (2π). At this point, the values repeat the cycle. The sine values can been seen by putting a `sin()` function inside a `for` structure and iterating while changing the angle value:

14-03

```
for (float angle = 0; angle < TWO_PI; angle += PI/24.0) {
  println(sin(angle));
}
```

Because the values from `sin()` are numbers between –1.0 and 1.0, they are easy to use in controlling a composition. Multiplying the numbers by 50.0, for example, will return values between –50.0 and 50.0.

14-04

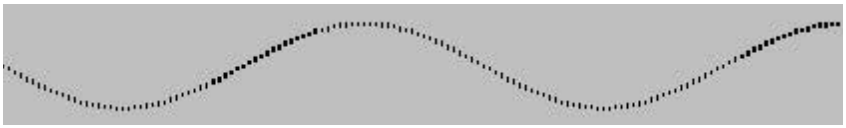
```
for (float angle = 0; angle < TWO_PI; angle += PI/24.0) {
  println(sin(angle) * 50.0);
}
```

To convert the sine values to a range of positive numbers, first add the value 1.0 to create numbers between 0.0 and 2.0. Divide that number by 2.0 to get a number between 0.0 and 1.0, which can then be simply remapped to any range. Alternatively, the `map()` function can be used to convert the values from `sin()` to any range. In this example, the values from `sin()` are put into the range between 0 and 1000.

14-05

```
for (float angle = 0; angle < TWO_PI; angle += PI/24.0) {
  float newValue = map(sin(angle), -1, 1, 0, 1000);
  println(newValue);
}
```

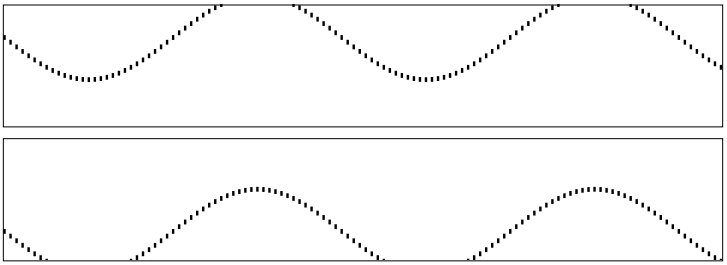
If we set the y-coordinates for a series of points with the numbers returned from the `sin()` function and continually increase the value of the angle parameter before each new coordinate is calculated, the sine wave emerges:

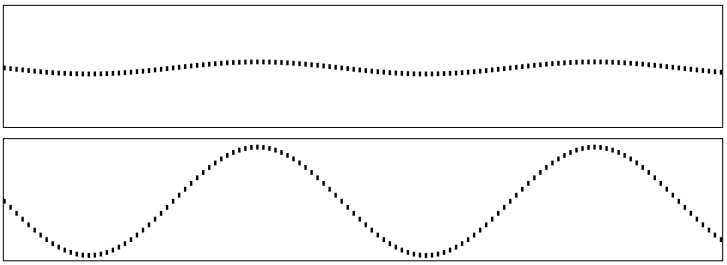


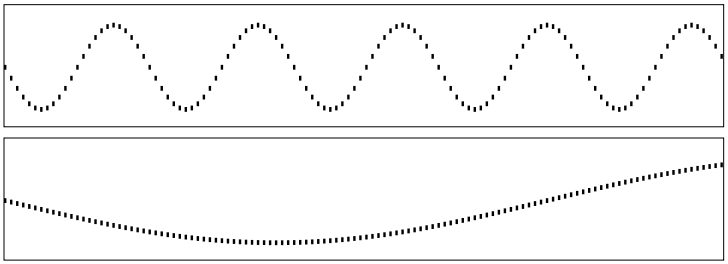
14-06

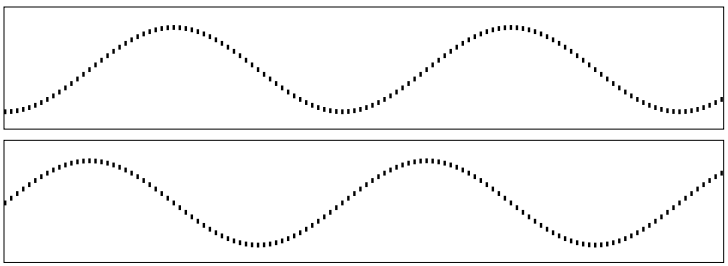
```
size(700, 100);
noStroke();
fill(0);
float angle = 0.0;
for (int x = 0; x <= width; x += 5) {
  float y = 50 + (sin(angle) * 35.0);
  rect(x, y, 2, 4);
  angle += PI/40.0;
}
```

Replacing some fixed numbers in the previous program with variables allows you to control the waveform by simply changing the values of the variables. The `offset` variable controls the y-coordinates of the wave, the `scaleVal` variable controls the









Modulating a sine wave
Different values for the variables in code 14-07 create a range of waves. Notice how each variable affects a different attribute of the wave.

height of the wave, and the `angleInc` variable controls the speed at which the angle increases, thereby creating a wave with a higher or lower frequency.

14-07

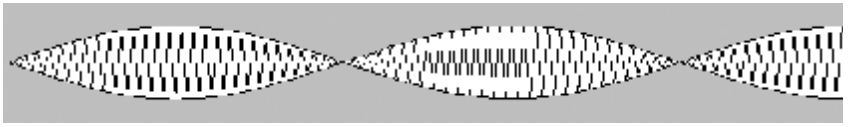
```
size(700, 100);
noStroke();
smooth();
fill(0);
float offset = 50.0;       // Y offset
float scaleVal = 35.0;     // Scale value for the wave magnitude
float angleInc = PI/28.0;  // Increment between the next angle
float angle = 0.0;         // Angle to receive sine values from
for (int x = 0; x <= width; x += 5) {
  float y = offset + (sin(angle) * scaleVal);
  rect(x, y, 2, 4);
  angle += angleInc;
}
```

The `cos()` function returns values in the same range and pattern as `sin()`, but the numbers are offset by π/2 radians (90°).

14-08

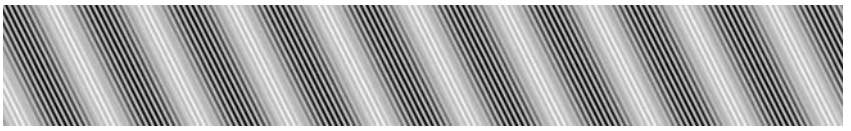
```
size(700, 100);
noStroke();
smooth();
float offset = 50.0;
float scaleVal = 20.0;
float angleInc = PI/18.0;
float angle = 0.0;
for (int x = 0; x <= width; x += 5) {
  float y = offset + (sin(angle) * scaleVal);
  fill(255);
  rect(x, y, 2, 4);
  y = offset + (cos(angle) * scaleVal);
  fill(0);
  rect(x, y, 2, 4);
  angle += angleInc;
}
```

The following examples demonstrate ways to use the numbers from the `sin()` function to generate shapes.



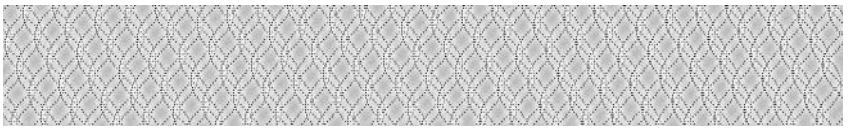
14-09

```
size(700, 100);
float offset = 50;
float scaleVal = 30.0;
float angleInc = PI/56.0;
float angle = 0.0;
beginShape(TRIANGLE_STRIP);
for (int x = 4 ; x <= width+5; x += 5) {
  float y = sin(angle) * scaleVal;
  if ((x % 2) == 0) {  // Every other time through the loop
    vertex(x, offset + y);
  } else {
    vertex(x, offset - y);
  }
  angle += angleInc;
}
endShape();
```



14-10

```
size(700, 100);
smooth();
strokeWeight(2);
float offset = 126.0;
float scaleVal = 126.0;
float angleInc = 0.42;
float angle = 0.0;
for (int x = -52; x <= width; x += 5) {
  float y = offset + (sin(angle) * scaleVal);
  stroke(y);
  line(x, 0, x+50, height);
  angle += angleInc;
}
```



14-11

```
size(700, 100);
smooth();
fill(255, 20);
float scaleVal = 18.0;
float angleInc = PI/28.0;
float angle = 0.0;
for (int offset = -10; offset < width+10; offset += 5) {
  for (int y = 0; y <= height; y += 2) {
    float x = offset + (sin(angle) * scaleVal);
    noStroke();
    ellipse(x, y, 10, 10);
    stroke(0);
    point(x, y);
    angle += angleInc;
  }
  angle += PI;
}
```

Circles, Arcs, Spirals

Circles can be drawn from sine and cosine waves. The example below has an angle that increments by 12°, all the way up to 360°. On each step, the `cos()` value of the angle is used to draw the x-coordinate, and the `sin()` value draws the y-coordinate. Because `sin()` and `cos()` return numbers between –1.0 and 1.0, the result is multiplied by the `radius` variable to draw a circle with radius 38. Adding 50 to the `x` and `y` positions sets the center of the circle at (50,50).

14-12

```
noStroke();
smooth();
int radius = 38;
for (int deg = 0; deg < 360; deg += 12) {
  float angle = radians(deg);
  float x = 50 + (cos(angle) * radius);
  float y = 50 + (sin(angle) * radius);
  ellipse(x, y, 6, 6);
}
```

If the angle is incremented only part of the way around the circle, an arc is drawn. For example, changing line 4 in the preceding program gives the following result:

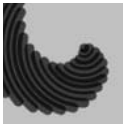
```
noStroke();                                                    14-13
smooth();
int radius = 38;
for (int deg = 0; deg < 220; deg += 12) {
  float angle = radians(deg);
  float x = 50 + (cos(angle) * radius);
  float y = 50 + (sin(angle) * radius);
  ellipse(x, y, 6, 6);
}
```

To simplify drawing arcs, Processing includes an `arc()` function:

```
arc(x, y, width, height, start, stop)
```

Arcs are drawn along the outer edge of an ellipse defined by the `x`, `y`, `width`, and `height` parameters. The `start` and `stop` parameters specify the angles needed to draw the arc form in units of radians. The following examples show the function in use.

```
strokeWeight(2);                                               14-14
arc(50, 55, 50, 50, 0, HALF_PI);
arc(50, 55, 60, 60, HALF_PI, PI);
arc(50, 55, 70, 70, PI, TWO_PI - HALF_PI);
noFill();
arc(50, 55, 80, 80, TWO_PI - HALF_PI, TWO_PI);
```



```
smooth();                                                      14-15
noFill();
randomSeed(0);
strokeWeight(10);
stroke(0, 150);
for (int i = 0; i < 160; i += 10) {
  float begin = radians(i);
  float end = begin + HALF_PI;
  arc(67, 37, i, i, begin, end);
}
```

To create a spiral, multiply the sine and cosine values by increasing or decreasing scalar values. In the following examples, the spiral grows as the `radius` variable increases:

14-16

```
noStroke();
smooth();
float radius = 1.0;
for (int deg = 0; deg < 360*6; deg += 11) {
  float angle = radians(deg);
  float x = 75 + (cos(angle) * radius);
  float y = 42 + (sin(angle) * radius);
  ellipse(x, y, 6, 6);
  radius = radius + 0.34;
}
```

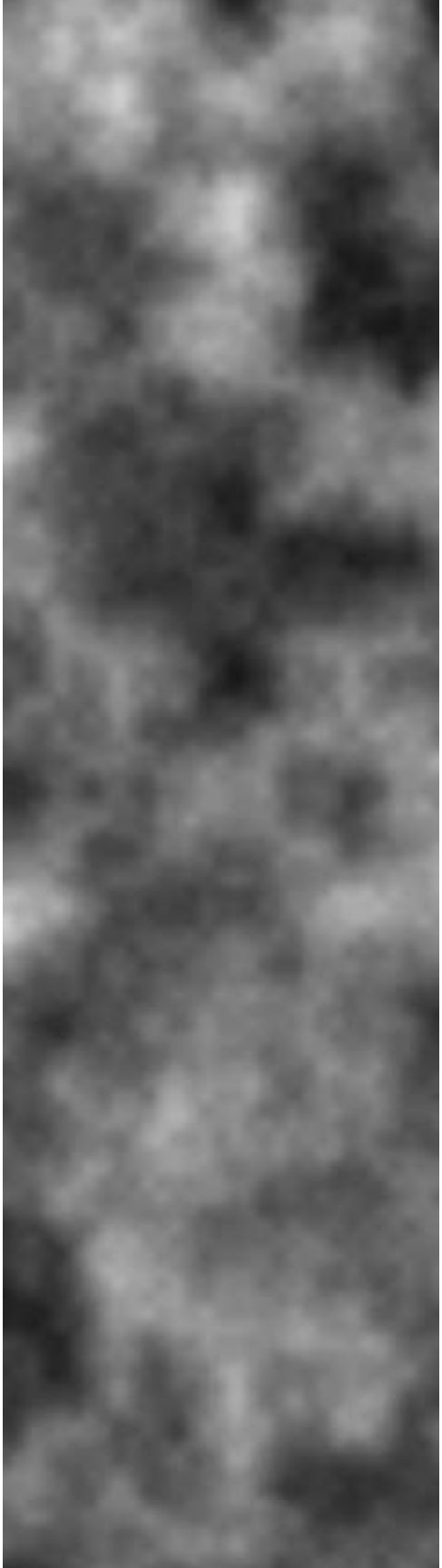
14-17

```
smooth();
float radius = 0.15;
float cx = 33;  // Center x- and y-coordinates
float cy = 66;
float px = cx;  // Start with center as the
float py = cy;  // previous coordinate
for (int deg = 0; deg < 360*5; deg += 12) {
  float angle = radians(deg);
  float x = cx + (cos(angle) * radius);
  float y = cy + (sin(angle) * radius);
  line(px, py, x, y);
  radius = radius * 1.05;
  px = x;
  py = y;
}
```

The content of this unit is applied to controlling movement in Motion 2 (p. 291).

Exercises

1. *Create a composition with the data generated using `sin()`.*
2. *Explore drawing circles and arcs with `sin()` and `cos()`. Develop a composition from the results of the exploration.*
3. *Generate a series of spirals and organize them into a composition.*



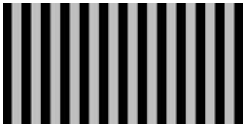
Math 4: Random

This unit introduces the basics of trigonometry and random numbers and explains how to utilize them for generating form.

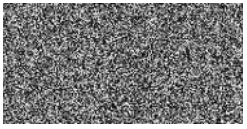
Syntax introduced:
`random(), randomSeed(), noise(), noiseSeed()`

Random compositional choices have a long history, particularly in modern art. In 1913 Marcel Duchamp's *3 Stoppages Étalon* employed the curves of dropped threads to derive novel units of measurement. Jean Arp used chance operations to define the position of elements in his collages. The composer John Cage sometimes tossed coins to determine the order and duration of notes in his scores. Artists integrate chance, randomness, and noise into their work either as a creative exercise or as a way of relinquishing some control to an external force. Actions like dropping, throwing, rolling, etc., deprive the artists of certain aspects of decisions. The world's chaos can be channeled into making images and objects with physical media. In contrast, computers are machines that make consistent and accurate calculations and must therefore simulate random numbers to approximate the kind of chance operations used in nondigital art.

There is an obvious contrast between rigid structure and complete chaos, and some of the most satisfying aesthetic experiences are created by infusing one with the other. The tension between order and chaos can actively engage our attention:



If a composition is obviously ordered, it will not hold attention beyond a quick glance.



Conversely, if a composition is entirely chaotic, it will also not retain one's gaze.

A balance between the two can yield a more satisfying result.

Unexpected values

The `random()` function is used to create unpredictable values within the range specified by its parameters.

```
random(high)
random(low, high)
```

When one parameter is passed to the function, it returns a `float` from zero up to (but not including) the value of the parameter. The function call `random(5.0)` returns