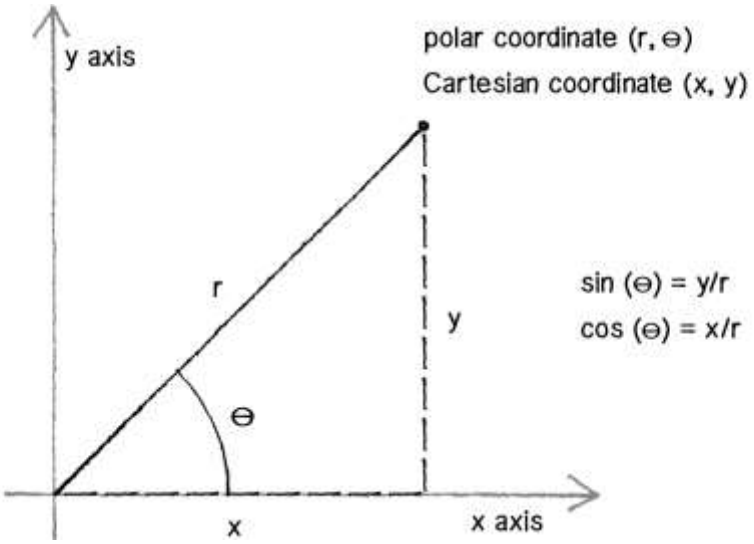
Polar coordinates

Any time we display a shape in ProcessingJS, we have to specify a pixel location, a set of `x` and `y` coordinates. These coordinates are known as ***Cartesian coordinates***, named for René Descartes, the French mathematician who developed the ideas behind Cartesian space.

Another useful coordinate system known as ***polar coordinates*** describes a point in space as an angle of rotation around the origin and a radius from the origin. Thinking about this in terms of a vector:

- Cartesian coordinate—the *x,y* components of a vector
- Polar coordinate—the magnitude (length) and direction (angle) of a vector
  However, the drawing functions in ProcessingJS don't understand polar coordinates. Whenever we want to display something in ProcessingJS, we have to specify locations as *(x,y)* Cartesian coordinates. However, sometimes it is a great deal more convenient for us to think in polar coordinates when designing. Happily for us, with trigonometry we can convert back and forth between polar and Cartesian, which allows us to design with whatever coordinate system we have in mind but always draw with Cartesian coordinates.



The Greek letter θ (theta) is often used to denote an angle, and a polar coordinate is conventionally referred to as *(r, θ)* instead of *(x, y)*. Thus, when dealing with polar coordinates, we'll now use "theta" as the preferred variable name for the angle.

<pre>**sine(theta) = y/r → y = r * sine(theta)

cosine(theta) = x/r → x = r * cosine(theta)** </pre>

For example, if `r` is 75 and `theta` is 45 degrees (or PI/4 radians), we can calculate `x` and `y` as below. The functions for sine and cosine in ProcessingJS are `sin()` and `cos()`, respectively. They each take one argument, an angle measured in degrees.

```
var r = 75;
var theta = 45;

// Convert polar to cartesian
var x = r * cos(theta);
var y = r * sin(theta);
```

This type of conversion can be useful in certain applications. For example, to move a shape along a circular path using Cartesian coordinates is not so easy. With polar coordinates, on the other hand, it’s simple: increment the angle!

Here's how we can make a simple rotating shape using polar coordinate conversion:

```
// Convert a polar coordinate (r,theta)
//   to cartesian (x,y):
// x = r * cos(theta)
// y = r * sin(theta)
```

```
var r = height * 0.40;
var theta = 0;

var draw = function() {
  background(255, 255, 255);

  // Translate the origin point to the center of the screen
  pushMatrix();
  translate(width/2, height/2);

  // Convert polar to cartesian
  var x = r * cos(theta);
  var y = r * sin(theta);
    // Draw the ellipse at the cartesian coordinate
  ellipseMode(CENTER);
  fill(127);
  stroke(0);
  strokeWeight(2);
  line(0, 0, x, y);
  ellipse(x, y, 48, 48);

  // Increase the angle over time
  theta += 1;
  popMatrix();
};
```