Drawing Circles and Other Regular Polygons, I
First, We Need to Understand Something about Angles

If a circle has a radius of 1.0, then we can march around it by simply changing the angle that we call $\theta$. 
First, We Need to Understand Something about Angles

One of the things we notice is that each angle $\theta$ has a unique $X$ and $Y$ that goes with it.

*The $X$ and $Y$ are different for each $\theta$.***
First, We Need to Understand Something about Angles

Centuries ago, people developed tables of those X and Y values as functions of θ. They called the X values **cosines** and the Y values **sines**. These are abbreviated cos and sin.

\[
\begin{align*}
\cos \theta &= X \\
\sin \theta &= Y
\end{align*}
\]
In Earlier Times, People Looked up Sines and Cosines in Books and on Slide Rules – Fortunately We Now Have Calculators and Computers
Cosines and Sines are Really Ratios

If we were to double the radius of the circle, all of the X’s and Y’s would also double.

So, really the cos and sin are *ratios* of X and Y to the circle Radius

\[
\cos \theta = \frac{X}{R} \quad \sin \theta = \frac{Y}{R}
\]
Cosines and Sines are Really Ratios

So, if we know the circle Radius, and we march through a series of $\theta$ angles, we can determine all of the $X$’s and $Y$’s that we need to draw a circle.

\[
\cos \theta = \frac{X}{R} \quad \rightarrow \quad X = R \times \cos \theta
\]

\[
\sin \theta = \frac{Y}{R} \quad \rightarrow \quad Y = R \times \sin \theta
\]
Processing Doesn’t Include Regular Polygon-Drawing Function, So We Add Our Own to the End of the Program

```
function Shape( xc, yc, r, numsegs )
{
    let dang = (2.*PI) / float( numsegs );
    let ang = 0.;
    beginShape( );

    for( let i = 0; i <= numsegs; i = i + 1 )
    {
        let x = xc + r * cos(ang);
        let y = yc + r * sin(ang);
        vertex( x, y );
        ang = ang + dang;
    }

    endShape( );
}
```

**numsegs** is the number of line segments making up the circumference of the circle.

numsegs=36 gives a nice circle.

5 gives a pentagon.
8 gives an octagon.
4 gives you a square. Etc.

Why **2.*PI**?
Why 2.*PI? 

let dang = (2.*PI) / float( numsegs );

We commonly measure angles in degrees, but scientists, engineers, and computers like to measure angles in something else called radians.

There are 360° (degrees) in a complete circle.
There are 2π (~6.28) radians in a complete circle.

The built-in cos( ) and sin( ) functions expect angles to be given in radians.

Processing has built-in functions to convert between the two:

let rad = radians( deg );
let deg = degrees( rad );
function draw( )
{
    fill( 255, 50, 50 );
    Shape( 200, 200, 100, 36 );

    fill( 50, 255, 50 );
    Shape( 300, 300, 100, 5 );

    fill( 50, 50, 255 );
    Shape( 400, 400, 100, 8 );
}
And, there is no reason the X and Y radii need to be the same…

```javascript
function Shape2( xc, yc, rx, ry, numsegs ) {
    let dang = (2.*PI) / float( numsegs );
    let ang = 0.;
    beginShape( );

    for( let i = 0; i <= numsegs; i = i + 1 ) {
        let x = xc + rx * cos(ang);
        let y = yc + ry * sin(ang);
        vertex( x, y );
        ang = ang + dang;
    }

    endShape( );
}
```
There is actually no reason the X and Y radii need to be the same …

function draw( )
{
    fill( 255,  50,  50 );
    Shape2( 200, 200, 150, 75,  36 );

    fill(  50, 255,  50 );
    Shape2( 300, 300, 150, 75,   5 );

    fill(  50,  50, 255 );
    Shape2( 400, 400, 150, 75,   8 );
}