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 $\theta$.

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

These are different for each $\theta$.

Fortunately, centuries ago, people developed tables of those $X$ and $Y$ values as functions of $\theta$.

They called the $X$ values cosines and the $Y$ values sines. These are abbreviated cos and sin.

$$\cos \theta = X$$
$$\sin \theta = Y$$
How People used to Lookup Sines and Cosines – Fortunately We Now Have Calculators and Computers

First, We Need to Understand Something about Angles

\[ \cos \theta = \frac{X}{R} \]
\[ \sin \theta = \frac{Y}{R} \]

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

First, We Need to Understand Something about Angles

So, if we know the circle Radius, and we march through a bunch of \( \theta \) angles, we can determine all of the X's and Y's that we need to draw a circle.

\[ \cos \theta = \frac{X}{R} \]
\[ \sin \theta = \frac{Y}{R} \]

Processing Doesn't Include a Circle-Drawing Function, So We Add Our Own

void Circle( int xc, int yc, int r, int numsegs )
{
    float dang = (2.*PI) / float( numsegs );
    float ang = 0.;
    beginShape( );
    for( int i = 0; i <= numsegs; i = i + 1 )
    {
        float x = xc + r * cos(ang);
        float 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=20 gives a nice circle.
5 gives a pentagon.
8 gives an octagon.
4 gives you a square. Etc.
float dang = (2.*PI) / float(numsegs);

Why 2.*PI?

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

There are 360° in a complete circle.
There are 2\pi radians in a complete circle.
The built-in cos() and sin() functions expect angles given in radians.
Processing has built-in functions to convert between the two:
float rad = radians(deg);
float deg = degrees(rad);

Circle, Pentagon, Octagon!

Circle, Pentagon, Octagon!

Or, even:
void Ellipse( int xc, int yc, int rx, int ry, int numsegs )
{
    float dang = (2.*PI) / float( numsegs );
    float ang = 0.;
    beginShape( );
    for( int i = 0; i <= numsegs; i = i + 1 )
    {
        float x = xc + rx * cos(ang);
        float y = yc + ry * sin(ang);
        vertex( x, y );
        ang = ang + dang;
    }
    endShape( );
}

And, there is no reason the X and Y radii need to be the same...

void Spiral( int xc, int yc, int r0, int r1, int numsegs, int numturns )
{
    float dang = numturns * (2.*PI) / float( numsegs );
    float ang = 0.;
    beginShape( );
    for( int i = 0; i <= numsegs; i = i + 1 )
    {
        float newrad = map( i, 0, numsegs, r0, r1 );
        float x = xc + newrad * cos(ang);
        float y = yc + newrad * sin(ang);
        vertex( x, y );
        ang = ang + dang;
    }
    endShape( );
}

There is also no reason we can’t gradually change the radius...

There is actually no reason the X and Y radii need to be the same...
We Can Also Use This Same Idea to Arrange Things in a Circle

```cpp
void draw()
{
    stroke( 0, 0, 0 );
    int numobjects = 10;
    float radius = 200.;
    int xc = 300;
    int yc = 300;
    int numsegs = 20;
    int r = 50;
    float dang = (2.*PI) / float( numobjects - 1 );
    float ang = 0.;
    for( int i = 0; i < numobjects; i = i + 1 )
    {
        float x = xc + radius * cos(ang);
        float y = yc + radius * sin(ang);
        int red = int( map( i, 0, numobjects - 1, 0, 255 ) );
        int blue = int( map( i, 0, numobjects - 1, 255, 0 ) );
        fill( red, 0, blue );
        Circle( int(x), int(y), r, numsegs );
        ang = ang + dang;
    }
}
```