Drawing Circles and Other Regular Polygons

Mike Bailey
mjb@cs.oregonstate.edu
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.

These are different for each $\theta$. 
First, We Need to Understand Something about Angles

Fortunately, 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.

\[ \cos \theta = X \]
\[ \sin \theta = Y \]
How People used to Lookup Sines and Cosines – Fortunately We Now Have Calculators and Computers
First, We Need to Understand Something about Angles

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

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

\[
\sin \theta = \frac{Y}{R}
\]
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.

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

\[
\sin \theta = \frac{Y}{R} \quad \quad Y = R \times \sin \theta
\]
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.

Why 2.*PI ?

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

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.

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

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

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 2π 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:

```c
float rad = radians( deg );
float deg = degrees( rad );
```
Circle, Pentagon, Octagon!

```java
void draw() {
    stroke(0, 0, 0);
    fill(255, 50, 50);
    Circle(200, 200, 100, 20);
    fill(50, 255, 50);
    Circle(300, 300, 100, 5);
    fill(50, 50, 255);
    Circle(400, 400, 100, 8);
}
```
If We Move the Mouse, We Could Get:
Or, even:
And, there is no reason the X and Y radii need to be the same...

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

```c
void draw( )
{
    stroke( 0, 0, 0 );

    fill( 255, 50, 50 );
    Ellipse( 200, 200, 150, 75, 20 );

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

    fill( 50, 50, 255 );
    Ellipse( 400, 400, 150, 75, 8 );
}
```
There is also no reason we can’t gradually change the radius …

```c
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 ...

```java
void draw( ) {
    stroke( 50, 50, 255 );
    strokeWeight( 5 );
    noFill( );
    Spiral( 300, 300, 20, 200, 1000, 10 );
}
```
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, 0 ) );
        int blue = int( map( i,   0, numobjects – 1, 255, 0 ) );
        fill( red, 0, blue );
        Circle( int(x), int(y), r, numsegs );
        ang = ang + dang;
    }
}
```
We Can Also Use This Same Idea to Arrange Things in a Circle

```c
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;
    }
}
```