**Basic Drawing in Processing**

**Carson Foster**

Drawing Basic Shapes

The Point

 The most basic ‘shape’ that you can draw is a point. Yes, a single point. Processing’s syntax for drawing a point is quite simple:



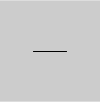
The two arguments represent the x and y coordinates of the point.



In this example, the point function creates a single point in the middle of the screen. The mechanism pictured here is quite useful: after the size function is used, the variable names width and height can be used to refer to the width and height of the drawing space, respectively.

The Line

From the humble point, we now arrive at the line: a connection between two points. In Processing, the syntax to draw a line is the same as the previously discussed representation: the coordinates for the start point, and then the coordinates for the end point, as you can see here:





In the following example, a line is drawn in the center third of the drawing space.



The Triangle

The line had two points, so can you make a guess as to what sort of shape we’ll be discussing now? That’s right, very observant of you. The triangle is our next shape, and is represented as you would expect:



These three points define the triangle, in the way you would imagine.



As you can see, this command draws a nice isosceles triangle in the middle of the drawing area.

The Square

As the next step in the logical progression of shapes, we next look at the syntax for drawing a square. The three necessary arguments to draw a square in Processing are the coordinates of the upper left-hand point of the square, and then a number that is both the width and height of the square.



In the following example, we’ll draw a square at the bottom right of the screen, and its area will be one fourth of the sketch’s area.

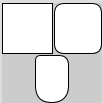


The Rectangle

The more general form of the square, the rectangle is the next shape we’ll cover. The rectangle has a couple of different sets of arguments you can use. All of them include the basic representation of the rectangle that we discussed: the coordinates of the top-left corner, and then the width and the height. The first set of arguments is just these. The other two specify a radius/radii for drawing rounded rectangles. The second set specifies a single radius that it applies to all corners. The last set of arguments needs the 4 regular arguments, and 4 arguments representing the separate radii of each corner, starting at the top left and moving clockwise.



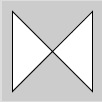
As you can see, there are a couple options to choose from when drawing a rectangle. The following example will show all of them.



I’ve spaced the rectangles out a little bit to make sure that their outlines are visible. Take note of how the different corner radii appear.

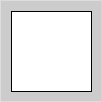
The Quadrilateral

Next, we’ll discuss the more general form of the rectangle: the quadrilateral. As you now definitely suspect, a quadrilateral is specified by four points in Processing. These points must be provided in clockwise or counterclockwise order.



The same points can construct different shapes if they are provided in a different order:



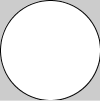
This creates a bowtie shape, while the following will create a rectangle:



The Circle

We’ll next be discussing the curved shapes, so we’ll start with the simplest, the circle. A circle in Processing is described in a similar way to the common representation we have already looked at.



The coordinates still refer to the center of the circle, but the third argument, instead of the radius, represents both the width and the height of the circle (the diameter).

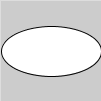


In this example, a circle is inscribed in the center of the sketch window.

The Ellipse

The ellipse is the more general form of the circle, and we’ll describe an ellipse using the same four parameters as a rectangle: coordinates (but this time of the center), and width and height.



Be mindful that the x and y coordinates describe the **center** of the ellipse. In some other languages, ellipses are represented with the same arguments, but the x and y coordinates represent the upper-left corner of the bounding rectangle of the ellipse (the rectangle that could be drawn surrounding the ellipse). This is not the case in Processing.

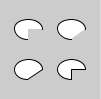


Pay attention to how the arguments affect the actual ellipse shown in this example.

The Arc

More complicated than our other two curved shapes, we next speak about the arc, which is essentially part of an ellipse. The syntax for creating is arc is as follows:



The first four arguments correspond to the same arguments for an ellipse. The start and stop arguments are angles (in radians) to start and stop the drawing of the ellipse, in clockwise order. This means that π/2 is pointing down, and that the start argument must be smaller than the stop argument, so sometimes negative values will be in order. The last argument is an optional mode, which can be one of 3 values: PIE, CHORD, or OPEN. See the example to find out how each one works.



The arcs proceed in clockwise order, so the top left arc is the first arc function call, and the bottom left arc is the third arc function call.

The Bezier Curve

The Bezier curve is a type of curve commonly used in computer graphics. I’ll only provide a brief introduction, but there are plenty of online resources if you would like to research further. In Processing, the command we’ll be using only uses cubic Bezier curves, which is flexible enough for basic applications. Essentially, you can represent a cubic Bezier curve with four points:



The first point and the last point are anchor points: the Bezier curve starts at the first point and ends at the last point. The second and third points are called control points. They usually don’t lie on the actual Bezier curve, but they direct the curve instead. What this means, in short, is that the Bezier curve goes, from the starting anchor, in the direction of the first control point, and approaches the ending anchor from the direction of the second control point.



This example shows the Bezier curve, as well as the lines between the starting anchor and first control point and the ending anchor and second control point. As you can see, the line leaves the starting anchor in the direction of the first control point and enters the ending anchor from the direction of the second control point. The two unfamiliar functions relate to color and will be discussed shortly.

Irregular, Closed Polygons

Now, we’ll discuss how to create irregular, closed polygons, which are shapes made up of line segments that do not necessarily have to be the same length and connect the end and beginning. The functions discussed here have more applicable uses, but they are too advanced for the scope of this series. The syntax we will use is as follows:



To start, call the beginShape function. To end the shape, call the endShape(CLOSE) function. In between these too functions, make any number of calls to vertex, which takes a point as its arguments. As the names imply, each call to vertex adds a vertex (corner) to the polygon.



As you can see, this sequence of function calls draws an arrowhead-like shape to the sketching area.

Color, Stroke, and Fill

text

Transformations

text