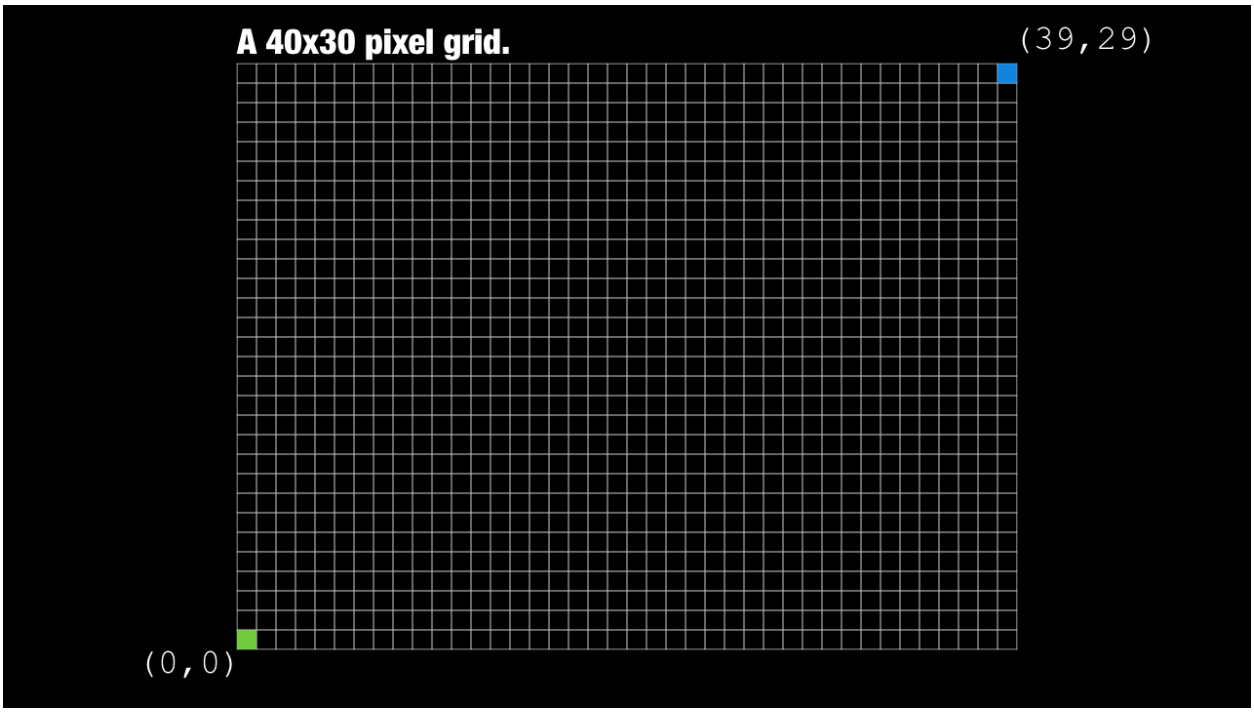
Drawing

Writing a program consists of creating a series of instructions for the computer to perform. There are many elements to learn about this task, but we will start with the simple idea of getting a few instructions working that accomplish something we want to happen.

I'm going to introduce you to a few functions (you can think of each of these as an instruction) that will allow you to make a drawing using Swift and the Tin framework. For example, to draw a circle you can use the ellipse function. But first, before we get into how these functions work, you need to learn about the coordinate system.

Coordinate system

When your program runs it will open a window. Inside the window is a view, which fills the entire window. Your program can draw in the view. The view is made up of a grid of pixels. Each pixel is a very small dot which can be set to a color. In order to draw using code you need to be able to describe the location of a pixel in the view. Each pixel in the view can be located by providing a location value along two axis. We will refer to these pairs like this: (x,y) where the first number is for the x (horizontal) axis, and the second number is for the y (vertical) axis. The coordinate system we use sets the origin - or the (0,0) point - to the bottom left pixel in the view. In the figure below, the green square is at the origin (0,0) point.



In the figure, the view is 40x30 pixels in size. Or, we could say that the view width is 40, and the view height is 30. On a modern computer this is a very small view. I'm using this small size here so that we can clearly see individual pixels, and identify them. Modern computer displays are very high resolution, that is, they have a large number of very small pixels that are very close together. When the pixels are small enough, the viewer does not discern the individual pixels

but instead only perceives the shapes made by groups of pixels. In this figure the top right pixel is colored blue, and is located at (39,29).

Primitive Shapes

We will start by using 4 functions that can draw simple shapes: a line, a rectangle, an ellipse, and a triangle. Each function needs us to provide some information - where should the shape be drawn? How big is the shape?

Line

We draw a line by calling the line function and specifying the two end points of the line. Here is an example.

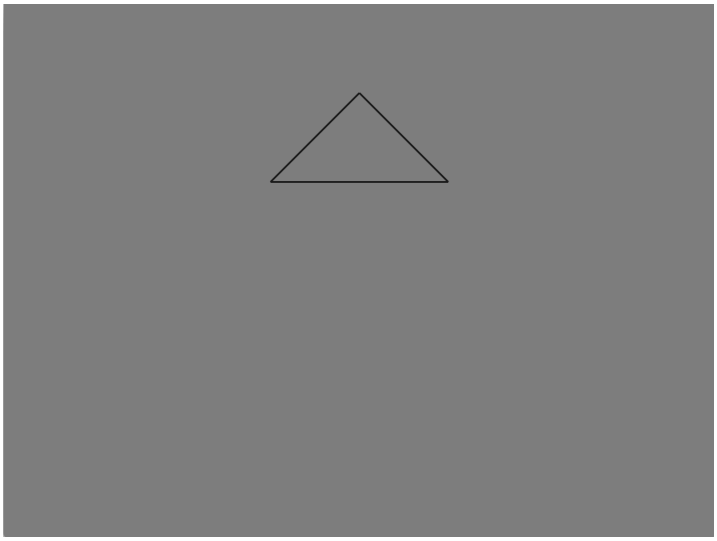
```
line(x1: 0, y1: 300, x2: 800, y2: 300)
```

The name of the function is line. A function can have multiple input parameters, written inside the function's parentheses. When there is more than one input parameters, they are separated by commas. In this example, a line from point (0,300) to point (800,300) is drawn.

Lets look more closely at the 4 input parameters for the line function, to better understand how parameters work in Swift. Each of these parameters has an argument label before the argument value - a colon is used to separate the label from the value. In the example, the first parameter is labeled x1, and the argument value 0 is passed to the function for this input. The second parameter is labeled y1, and the argument value is 300. The use of argument labels in Swift makes it easier to read and understand the purpose of input values for functions.

Draw a more complex shape by drawing multiple primitive shapes. This example creates a triangle by drawing three lines, one after the other. The line function is used three times, each time the combination of input values is different, and as a result, three different lines are drawn.

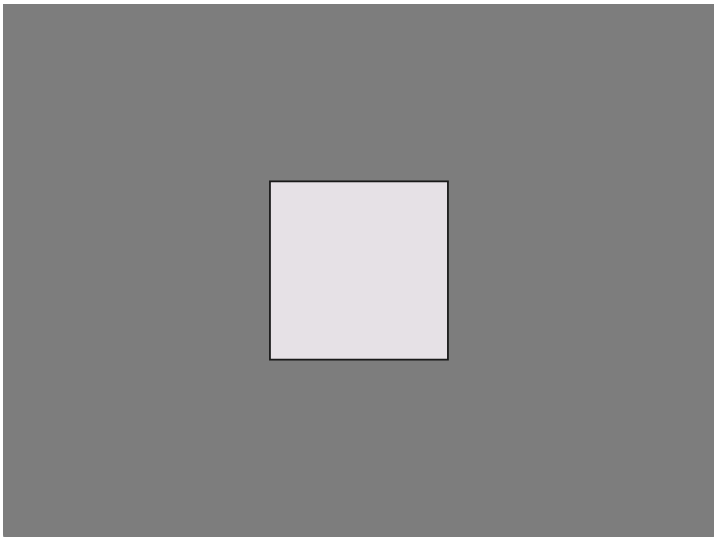
```
line(x1: 350, y1: 250, x2: 400, y2: 350)
line(x1: 400, y1: 350, x2: 450, y2: 250)
line(x1: 450, y1: 250, x2: 350, y2: 250)
```



Rect

Draw a rectangle using the function named rect, and specify a point (using an x and y value) and a width and a height. The point (x,y) describes the bottom left corner of the rectangle.

```
rect(x: 300.0, y: 200.0, width: 200.0, height: 200.0)
```



Ellipse

An oval shape can be drawing using the ellipse function. The oval is specified by describing a rectangle, and the oval is drawn inside that rectangle. As we saw with the rect function, a rectangle requires a position, width, and height. If the width and height values are equal, the resulting ellipse will be a circle. The position values for ellipse are different than rect - they represent the center of the shape, instead of the bottom left corner.

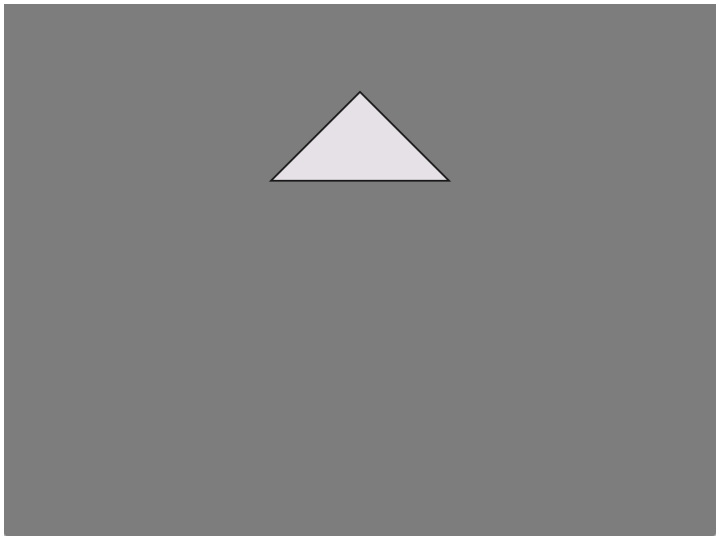
```
ellipse(centerX: 650, centerY: 500, width: 80, height: 80)
```

The example makes a circle that is 80 pixels wide and tall, with the center of the circle at (650,500).

Triangle

The triangle function draws a triangle shape using 6 inputs that specify the 3 vertices of the triangle. This example draws the same triangle that we previously made using three lines, with one call to the triangle function.

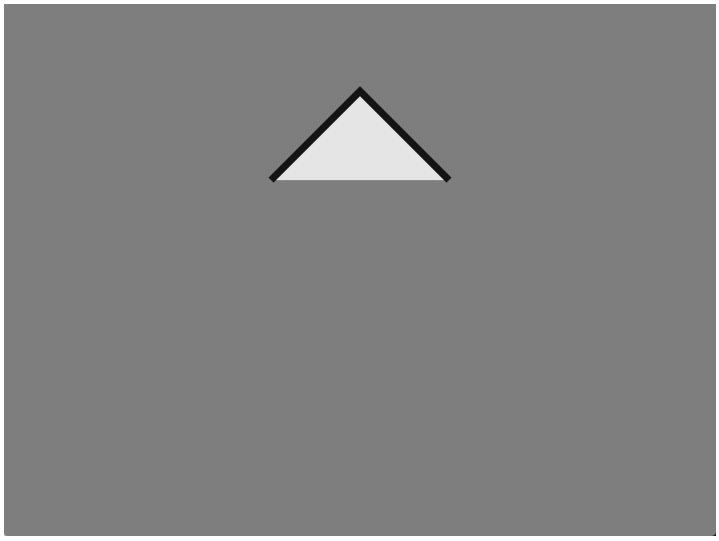
```
    triangle(x1: 300.0, y1: 400.0, x2: 500.0, y2: 400.0, x3: 400.0, y3: 500.0)
```



The example makes a triangle with the 1st vertex at (300,400), the 2nd vertex at (500,400), and the 3rd vertex at (400,500).

Path

A path can consist of an arbitrary number of points. To start the path, use the function pathBegin. After using pathBegin, any number of points on the path can be specified using multiple pathVertex calls. Finally, the pathEnd function can be used to stop the path, and cause it to be drawn. Here is an example.



```
        lineWidth(8.0)
        pathBegin()
        pathVertex(x: 300, y: 400)
        pathVertex(x: 400, y: 500)
        pathVertex(x: 500, y: 400)
        pathEnd()
```

Background

Use the background function to clear the view background, setting the color to a constant value. The argument gray sets the clear color to a grayscale value, in the range [0,1].

```
    background(gray: 0.5)
```

The alternate red, green, blue version allows you to specify any RGB color value. The RGB values are in the range [0,1]

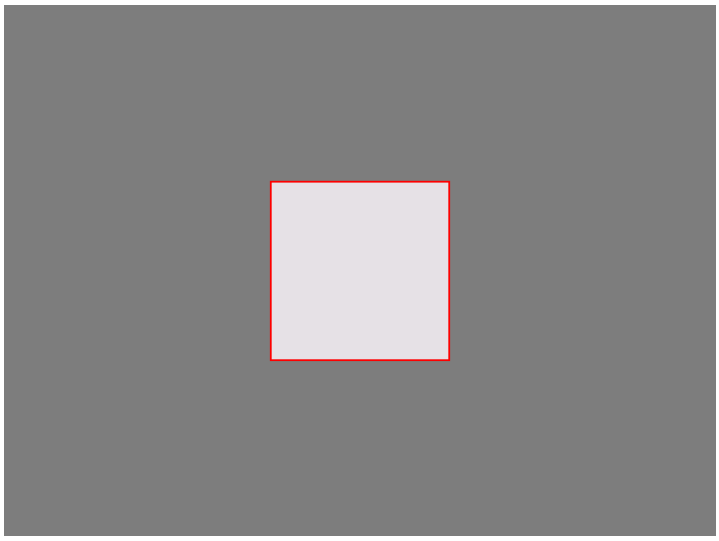
```
    background(red: 0.1, green: 0.2, blue: 0.5)
```

This example clears the view to a constant color red 0.1, green 0.2, blue 0.5.

Stroke and Fill Color

Tin drawing operations use the current state of the Stroke and Fill attributes to specify color. Stroke refers to the outline of shapes, while fill paints the interior space of the shape. The function strokeColor is used to set the stroke attribute state. For example, to make a shape with a bright red outline.

```
    strokeColor(red: 1, green: 0, blue: 0, alpha: 1)
```



The color is specified using red, green, and blue color components, with values in the range 0.0 to 1.0. The alpha value specifies transparency - 0 is completely transparent, 1 is totally opaque, and values in between provide transparency.

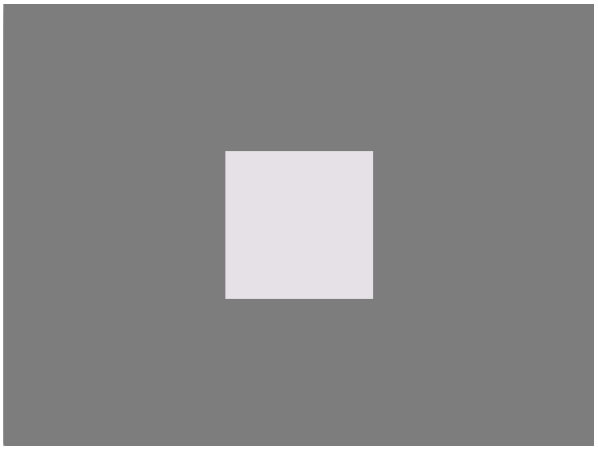
Another version of the strokeColor function takes one argument, in the range 0.0 to 1.0, to make a grayscale color. For example, the following will set the stroke color to be a black outline.

```
    strokeColor(gray: 0.0)
```

To draw a shape with no stroke outline, use the function strokeDisable.

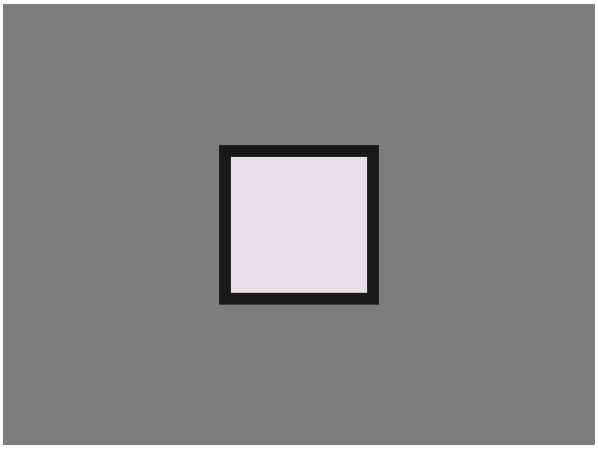
```
    strokeDisable()
```

This example is the same white rectangle, with no stroke.



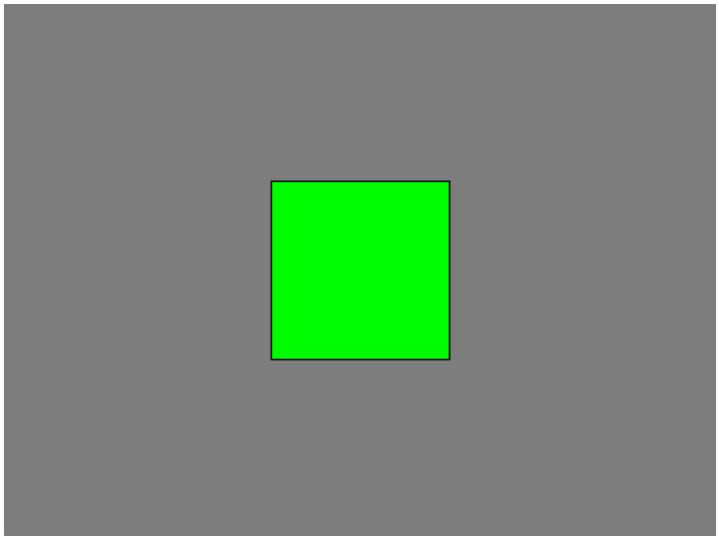
The width of the stroke for any shape can be set using the lineWidth function.

```
    lineWidth(16.0)
```



The function fillColor is used to set the fill attribute state. For example, to make a shape with a bright green interior fill.

```
    fillColor(red: 0, green: 1, blue: 0, alpha: 1)
```



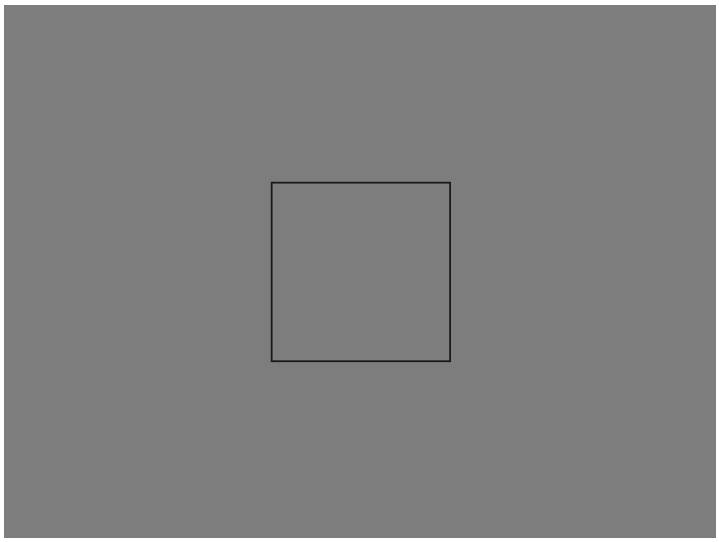
Just like strokeColor, there is a version of fillColor that takes one argument to make a grayscale color. For example, the following will set the fill color to be a 75% gray color.

```
    strokeColor(gray: 0.75)
```

To draw a shape with no fill, use the function fillDisable.

```
    fillDisable()
```

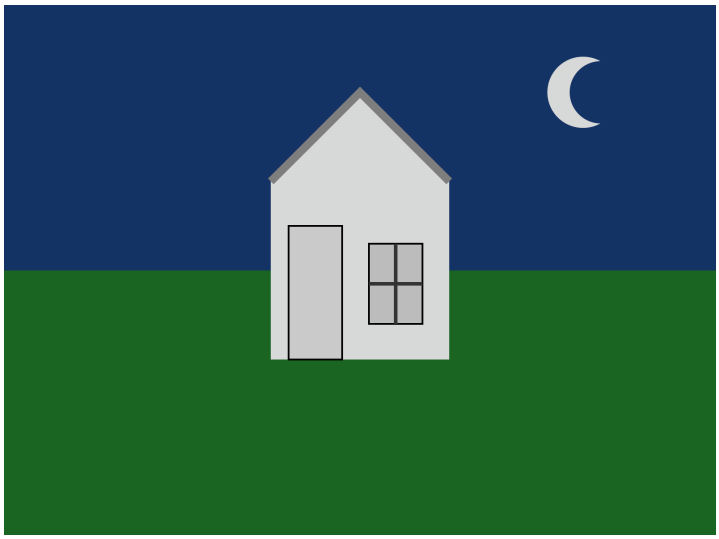
This example is the same rectangle, with a black stroke, no fill.



Keep in mind that the strokeColor, fillColor, and lineWidth attributes are part of the state for all drawing operations. When you change one of these attributes, all future drawing will use that attribute value until you change the state. In other words, if you set the fill color to red, all shapes that you draw will have a red fill, until you change the fill to another color.

Use many simple shapes to make more complex drawings.

With practice and patience, you can make drawings building with these simple shapes. Here is a simple house at night.



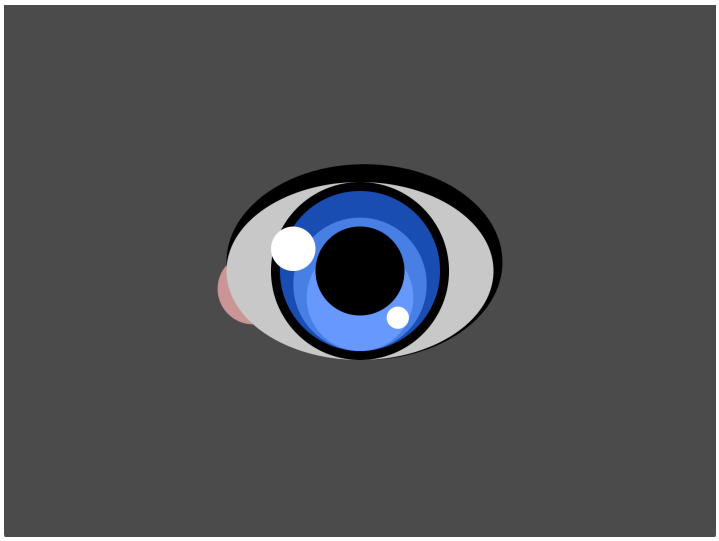
```
        // sky
        strokeDisable()
        fillColor(red: 0.1, green: 0.2, blue: 0.4, alpha: 1.0)
        rect(x: 0.0, y: 0.0, width: 800.0, height: 600.0)

        // ground
        fillColor(red: 0.1, green: 0.4, blue: 0.15, alpha: 1.0)
        rect(x: 0.0, y: 0.0, width: 800.0, height: 300.0)

        // house

        fillColor(gray: 0.85)
        rect(x: 300.0, y: 200.0, width: 200.0, height: 200.0)
        // peak
        triangle(x1: 300.0, y1: 400.0, x2: 500.0, y2: 400.0, x3: 400.0, y3: 500.0)

        // roof
        strokeColor(gray: 0.5)
        lineWidth(9.0)
        pathBegin()
        pathVertex(x: 300, y: 400)
        pathVertex(x: 400, y: 500)
```

```
        pathVertex(x: 500, y: 400)
        pathEnd()

        // door
        strokeColor(gray: 0.0)
        lineWidth(2.0)
        fillColor(gray: 0.8)
        rect(x: 320.0, y: 200.0, width: 60.0, height: 150.0)

        // window
        fillColor(gray: 0.75)
        rect(x: 410.0, y: 240.0, width: 60.0, height: 90.0)
        lineWidth(4.0)
        strokeColor(gray: 0.2)
        line(x1: 410.0, y1: 285.0, x2: 470.0, y2: 285.0)
        line(x1: 440.0, y1: 240.0, x2: 440.0, y2: 330.0)


        // moon
        strokeDisable()
        fillColor(gray: 0.85)
        ellipse(centerX: 650, centerY: 500, width: 80, height: 80)
        fillColor(red: 0.1, green: 0.2, blue: 0.4, alpha:1.0)
        ellipse(centerX: 670, centerY: 500, width: 70, height: 70)
```

Painter's algorithm, and the order of instructions.

When constructing a drawing, the order that shapes are drawn is important. The drawing is rendered using an algorithm known as the Painter's algorithm. The concept is very simple - things drawn later go on top. Just like a painter's brush strokes. Later brush strokes, if they cross earlier strokes, go on top of the older strokes. The same is true for the shapes in our program. Shapes drawn later will go on top of earlier shapes that they overlap.

Another example, with overlapping shapes.



This example was created by Jennifer Weiler.

```
        strokeDisable()
        let x = 250.0
        let y = 200.0

        ///////eyeball
        fillColor(red: 0.8, green: 0.6, blue: 0.6, alpha: 1.0)
        ellipse(centerX: x + 30, centerY: y + 80, width: 80, height: 80)
        fillColor(red: 0.0, green: 0.0, blue: 0.0, alpha: 1.0)
        ellipse(centerX: x + 155, centerY: y + 110, width: 310, height: 220)
        fillColor(red: 0.8, green: 0.8, blue: 0.8, alpha: 1.0)
        ellipse(centerX: x + 150, centerY: y + 100, width: 300, height: 200)
        fillColor(red: 0.0, green: 0.0, blue: 0.0, alpha: 1.0)
        ellipse(centerX: x + 150, centerY: y + 100, width: 200, height: 200)

        // iris
        fillColor(red: 0.1, green: 0.3, blue: 0.7, alpha: 1.0)
        ellipse(centerX: x + 150, centerY: y + 100, width: 180, height: 180)
        fillColor(red: 0.3, green: 0.5, blue: 0.9, alpha: 1.0)
        ellipse(centerX: x + 150, centerY: y + 85, width: 150, height: 150)
        fillColor(red: 0.4, green: 0.6, blue: 1.0, alpha: 1.0)
        ellipse(centerX: x + 150, centerY: y + 70, width: 120, height: 120)
        // pupil
        fillColor(red: 0.0, green: 0.0, blue: 0.0, alpha: 1.0)
        ellipse(centerX: x + 150, centerY: y + 100, width: 100, height: 100)

        //highlights
        fillColor(red: 1.0, green: 1.0, blue: 1.0, alpha: 1.0)
        ellipse(centerX: x + 75, centerY: y + 125, width: 50, height: 50)
        ellipse(centerX: x + 192.5, centerY: y + 47.5, width: 25, height: 25)
```