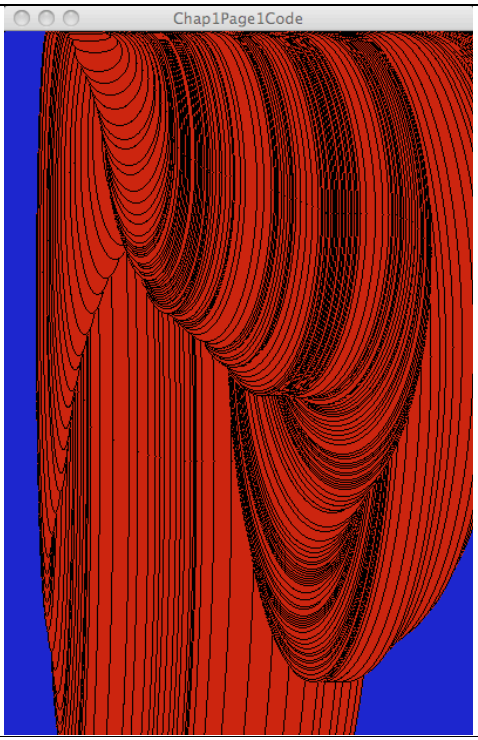
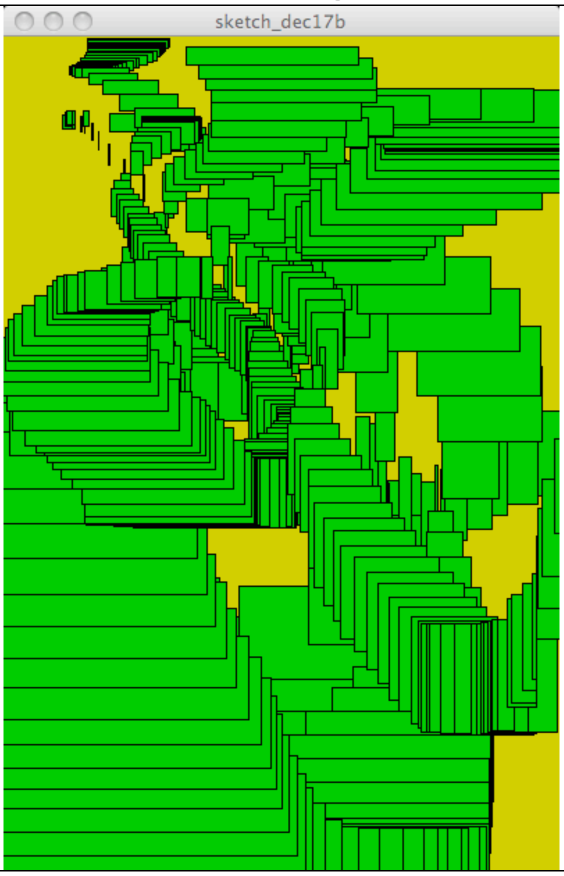
The Junior Woodchuck Manual of Processing Programming for Android Devices

The Image	*The Code*

	void setup( )
	{
	  size(400, 600);
	  background(0, 0 , 200); // blue
	  fill(200, 0, 0); //red
	}
	
	void draw( )
	{
	   ellipse(mouseX, mouseY,
	            mouseX/2, mouseY*2);
	}

Chapter 2
Getting Lost…

The Image	*The Code*

	**void setup( )
	{
	  size(400, 600);
	  smooth( );
	  background(200, 200 , 0);
	  fill(0, 200, 0);
	}
	
	void draw( )
	{
	   rect(mouseX, mouseY,
	           (mouseX – mouseY),
	             (mouseX + mouseY) / 10);
	}**

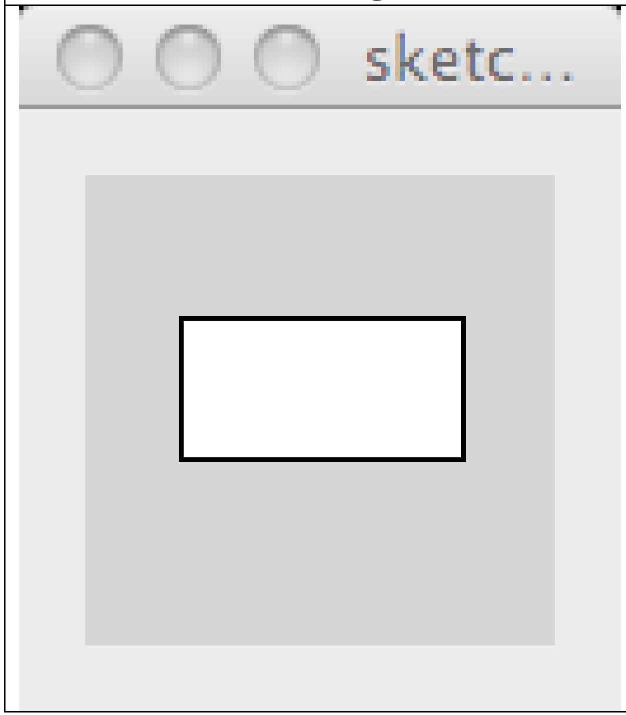
One way to have some fun programming is to get sorta' lost and then find our way out.

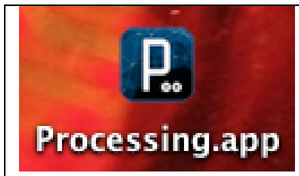
Sometimes it is better to try stuff and then figure out how it really works. In this chapter, you are going to try some stuff by drawing things and coloring them.

So… let's get started!

Section 1

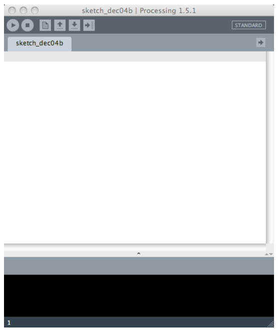
Where are we ???

The Image	The Code
	`rect( 20, 30, 60, 30 );`



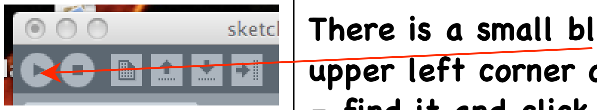
Find the Processing Icon or the Processing program on your computer and open it.

Your background will look different from the image on the right but the P will be the same.

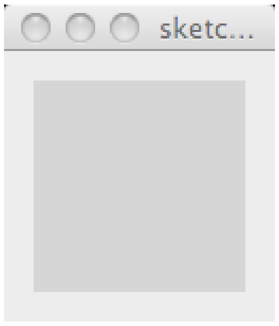


When Processing is open, you will see an empty window on your screen. This is where you will write your programs. This is called the "IDE".

The images shown here were done on a Macintosh. A windows computer may look a little different/



There is a small black triangle in a dark gray bar the upper left corner of the IDE. This is the "run" button – find it and click it.



You should another window with the word "sketch" and some other stuff in the title bar.

You have just run your first Processing program. It does not do much – in fact, it does not do anything but it is a working program.

If you want the program to actually do something (like the image on the first page), you have to write a program or "write some code" (like the code on the first page).

It is not difficult, but there are a lot of rules that you have to follow. We will talk about some of them later but, first, let's write some code and draw something in the window. Then you can draw what you want to draw without interference from the teacher.

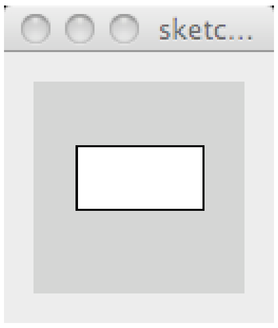
Programming languages like Processing use "functions" as the building blocks for writing programs. Each function does a specific job.

We will begin by drawing a rectangle (or box) on the window.

Type this code into the IDE window exactly as it is shown below.

```
    rect( 20, 30, 60, 30 );
```

and click the "run" button.



Hopefully, you will see this. If you do, you have just written your first Processing program and run your second Processing program. Try changing some of the numbers. Keep them smaller than 100.

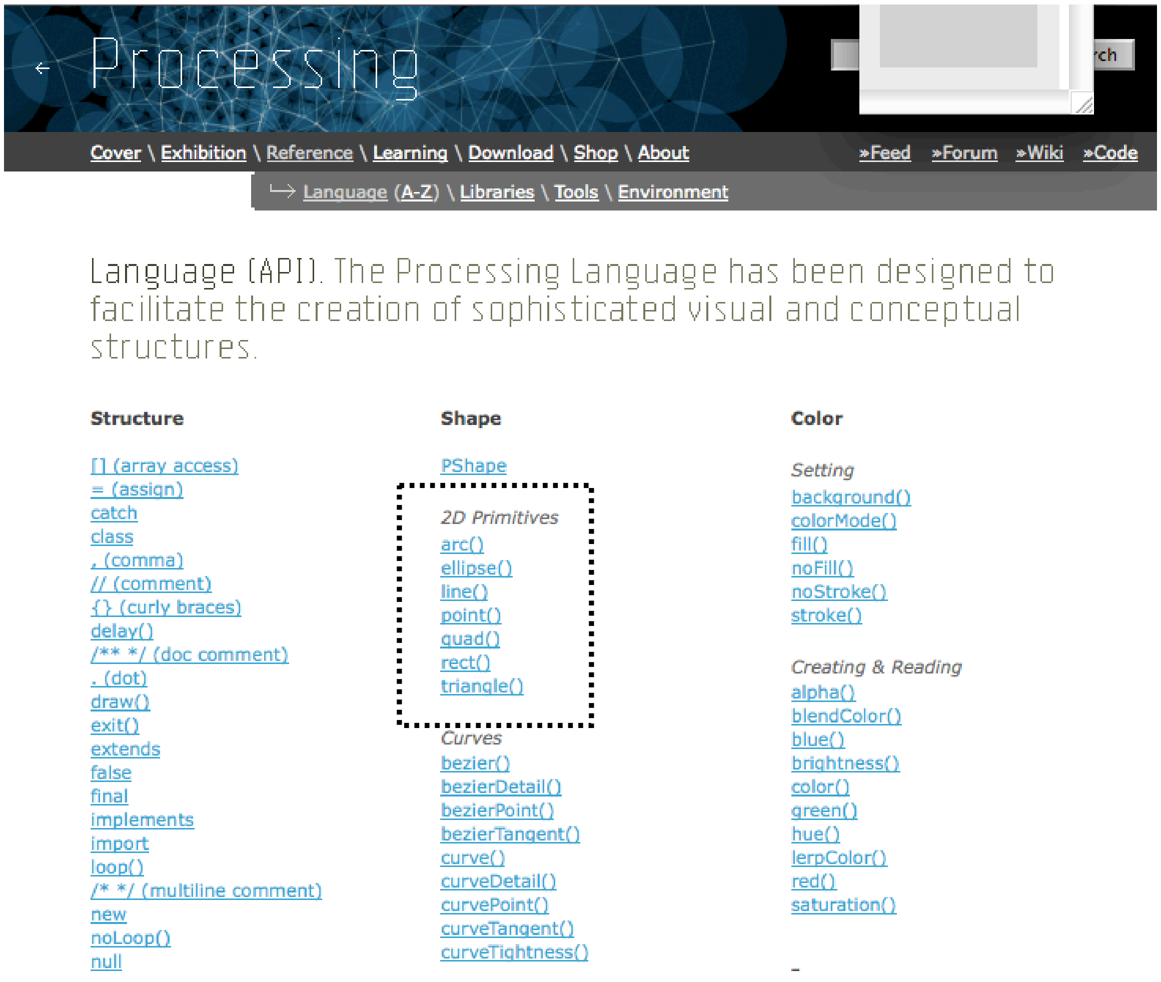
OK - how did your teacher know how to do this. He did not wake up this morning and figure it out. He used some information that Processing gives us to find out how to draw the rectangle.

JARGON ALERT
JARGON ALERT

This information on a web page called the "API". There will be tons of these three letter abbreviations. You can ask what they stand for and your teacher will tell you if he knows (he may make up some words if he doesn't). The important thing is not knowing what words they replace but what the letters mean.

You can go to the first page of Processing's API using this link:
http://www.processing.org/reference/index.html

This link will take you to the first page that has a list of all of the functions that Processing has and that you can use in your programs. The next page shows you part of this page:



Processing

Cover \ Exhibition \ Reference \ Learning \ Download \ Shop \ About »Feed »Forum »Wiki »Code

Language (A-Z) \ Libraries \ Tools \ Environment

Language (API). The Processing Language has been designed to facilitate the creation of sophisticated visual and conceptual structures.

Structure

[] (array access)
= (assign)
catch
class
, (comma)
// (comment)
{} (curly braces)
delay()
/** */ (doc comment)
. (dot)
draw()
exit()
extends
false
final
implements
import
loop()
/* */ (multiline comment)
new
noLoop()
null

Shape

PShape

2D Primitives
arc()
ellipse()
line()
point()
quad()
rect()
triangle()

Curves
bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

Color

Setting
background()
colorMode()
fill()
noFill()
noStroke()
stroke()

Creating & Reading
alpha()
blendColor()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

	There are almost three hundred functions in Processing. You will use only a few of these in the course. If you have time, you can play with any of them. Some of them are easy to use and others are a bit complicated but you should be able to figure out how to use them. In the middle column is a set of function named 2D Primitives. The 2D part means two dimensions: width and height. The primitives means that these are used a the building blocks of more complicated drawings and art. Let's look at this list a bit closer:

2D Primitives

arc()
ellipse()
line()
point()
quad()
rect()
triangle()

If you know what a radian is, the arc( ) function might be fun to tinker with. We will ignore it for a while.

These functions can be used to draw shapes in the window.

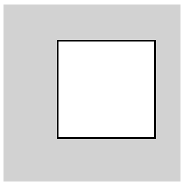
How can we tell that these are functions?

In Processing, functions end with parentheses ( ). The parentheses can be empty or have "stuff" inside them but there are always parentheses (remember – lots of rules…).

Since you used the rect( ) function to write your first program:

```
rect( 20, 30, 60, 30 );
```

Click on the word rect( ) in the API and see what Processing tells us about the rect( ) function.

Name	rect()
Examples	 `rect(30, 20, 55, 55);`
Description	Draws a rectangle to the screen. A rectangle is a four-sided shape with every angle at ninety degrees. By default, the first two parameters set the location of the upper-left corner, the third sets the width, and the fourth sets the height. These parameters may be changed with the **rectMode()** function.
Syntax	`rect(x, y, width, height)`
Parameters	x — int or float: x-coordinate of the upper-left corner
	y — int or float: y-coordinate of the upper-left corner
	width — int or float: width of the rectangle
	height — int or float: height of the rectangle

This is part of what Processing tells us about the rect( ) function. It tells us what the function does and what "stuff" we have to put into the parentheses if we want to use the function.

Remember those "rules" we have to follow. They are called "syntax" rules. Look for the line marked: syntax.

```
  Syntax       rect(x, y, width, height)
```

This tells us that if we want to draw a rectangle, we have to provide four sets of information: the x and y location of the rectangle and the width and height of the rectangle.

If you are not sure what the x and y location of the rectangle is, look below the syntax line and the API will tell you what x and y are:

```
Parameters
x      int or float: x-coordinate of the
         upper-left corner
y      int or float: y-coordinate of the
         upper-left corner
width  int or float: width of the rectangle
height int or float: height of the rectangle
```

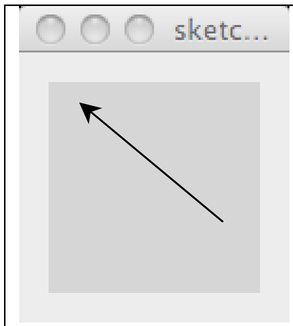
JARGON ALERT
JARGON ALERT

The stuff in the parentheses can be called the parameters or the arguments. Your teacher usually uses the work arguments. You can use either.

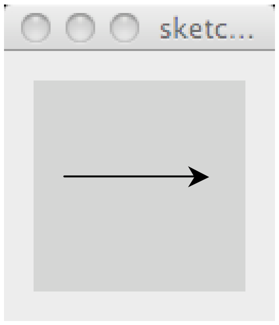
More
JARGON ALERT

The x and y coordinate s of the rectangle are called the rectangle's anchor points. This is the x and y point on the window that anchors the rectangle.

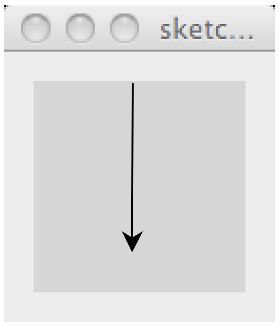
WAIT A MINUTE… We are using x and y but what do they really represent and where are they?



The upper left corner is the (0, 0) point of the window.



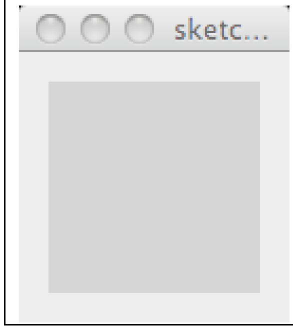
x refers to places that are sideways or across, or horizontally located in the window.



y refers to places that are up and down or vertically located in the window.

If we use an x value and a y value, we can tell Processing exactly where to put something in the window.

The unit of measure or counting is called the pixel. A pixel is the smallest dot you can draw and see on the screen. In the old days, there were about 50 pixels in an inch but today there is no rule. You have to experiment with your computer to see what looks good.



This window measures 100 pixels across (x direction) and 100 pixels down (y direction);

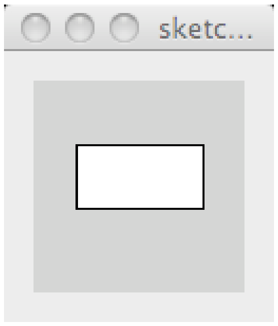
Why 100? That is what we call the default size

JARGON ALERT
JARGON ALERT

The word default means that this is what Processing uses if we do not tell it to use something else.

Let's go back to your first program and look at the defaults that Processing uses.

You wrote this code:
  rect(20, 30, 60, 30);
which produced this drawing:



The entire window except for the part covered by the rectangle is a gray - this is the default background color.

The rectangle is filled with white - this is the default fill color.

The rectangle is outlined with a black line. This line is called the stroke and the strokeColor is black.

Using functions in the API you can alter the background color, the fill color, the stroke color and the width of the stroke line.

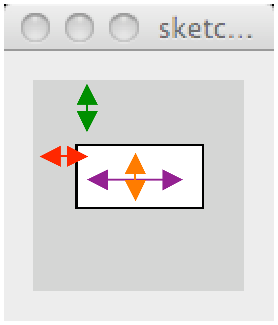
You can even turn off the fill color and the stroke. We will look at color in Section 2 of this chapter.

For now let's go back to your code:
  rect(20, 30, 60, 30);
The numbers in the parentheses are the parameters or arguments. Earlier we looked at the API to find out what those numbers mean :

```
Syntax    rect(x, y, width, height)
```

and we read further to learn this:

```
Parameters
x      int or float: x-coordinate of the
         upper-left corner
y      int or float: y-coordinate of the
         upper-left corner
width  int or float: width of the rectangle
height int or float: height of the rectangle
```



So let's translate all of this to our drawing:

```
rect( 20, 30, 60, 30 );
```

- The first argument tells Processing to locate the x point 20 pixels from the left edge.

- The second argument tells Processing to locate the y point 30 pixels down from the top edge.

- The third argument tells Processing to draw the rectangle 60 pixels wide.

- The fourth and last argument tells Processing to draw the rectangle 30 pixels high.

If you want to draw in a bigger window, make this the first line in your program BEFORE you draw anything:

```
    size(  400, 400 );
    rect( 20, 30, 60, 30 );
```

size( ) *must always* be first.

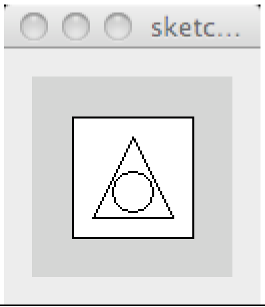
If you want to know how size( ) works. Look it up in the API.

Spend a few minutes drawing the other shapes to get familiar with how the work and how big a pixel is. The more you tinker now, the better it will be later.

Here are some things you should try to find out as you tinker:

- What is the anchor point (x, y) for the ellipse?
- How do you draw a circle?
- What are the punctuation rules you must follow to get your program to work?

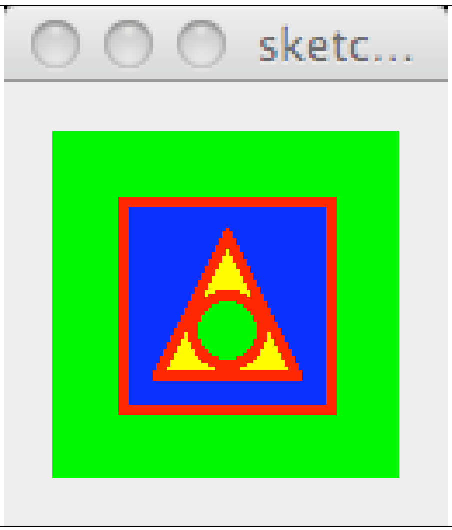
🡐 Draw this picture.



Section 2

Who brought the map???

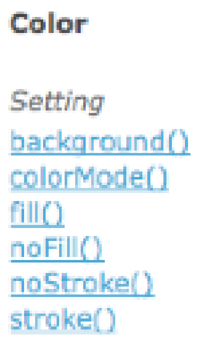
The Image	*The Code*

	background(0, 255, 0); // green
	stroke(255, 0, 0); // red
	strokeWeight(2);
	
	fill(0, 0, 255); // blue
	rect(20, 20, 60, 60);
	
	fill(255, 255, 0); // yellow
	triangle(50, 30, 30, 70, 70, 70);
	
	fill(0, 255, 0); // green
	ellipse(50, 57, 20, 20);

There was a time in the far dim past when televisions were black and white. Eventually someone figured out how to use color on the television and things were very different.

The same was true for computer screens. There was only black and white – not even gray. In fact, gray was a major improvement for computer programmers.

Then a smart programmer somewhere figured out how to put color into programs and programming became much more fun (*funner* is not a word… sigh…).


Color

Setting
background()
colorMode()
fill()
noFill()
noStroke()
stroke()

Processing allows us to color what we draw on the screen.

On the left is part of the API that we can use for adding color to our programs. Pretend that the function colorMode( ) is not there for now.

Each of these functions allows us to color or remove color from the shapes we draw on the window.

Before we look at the functions, let's look at color first.

We will work with colors in the same way we mix paint. Processing gives us three basic colors:
- red
- green
- blue

When you see the letters, RGB or rgb, it is usually referring to red, green, and blue.

We mix different amounts of these three colors to make the color we want to use. Processing always mixes the colors in the order:
  red, green, blue

One way to think about how this works is to pretend you are in a room with three light bulbs that are connected to dimmers that let you vary the amount of light coming from each bulb:

- one bulb is red
- one bulb is green
- one bulb is blue

When the three bulbs are turned off, the room is dark or black.
When the three bulbs are turned on all the way, the room is white.

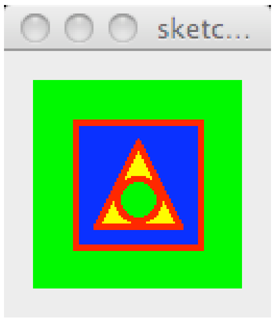
By changing the settings of the dimmers, you can create different colors of the room lighting.

In Processing a color is completely off when it is set to zero – makes sense. When a color is completely on, it setting is 255 -- ???? Save this for later… much later…

If you want to set color to red, you would type:
  255, 0, 0
which tells Processing you want full on red and zero green and zero blue.

If you want to set the color to blue, you would type:
  0, 0, 255
which tells Processing you want zero red, zero green, and full on blue.

We use these numbers as arguments when we use the functions to set the color.



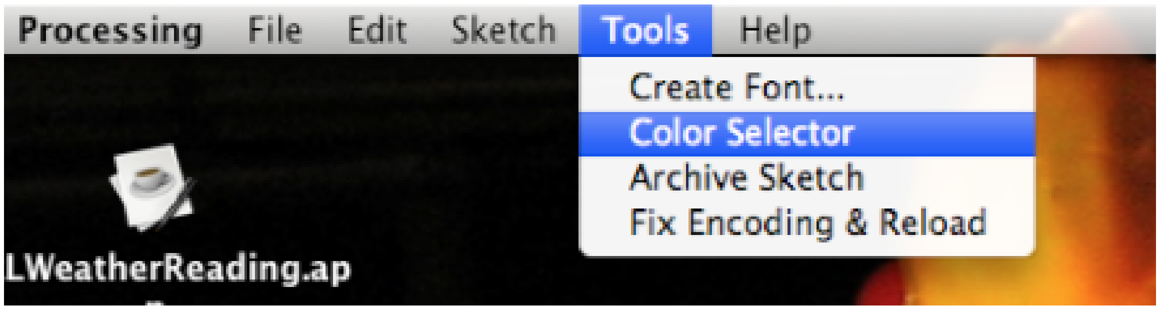
For example, in the image at the start of this section, the background color of the window is green. This was colored using the background function like this:

```
    background( 0, 255, 0 );
```

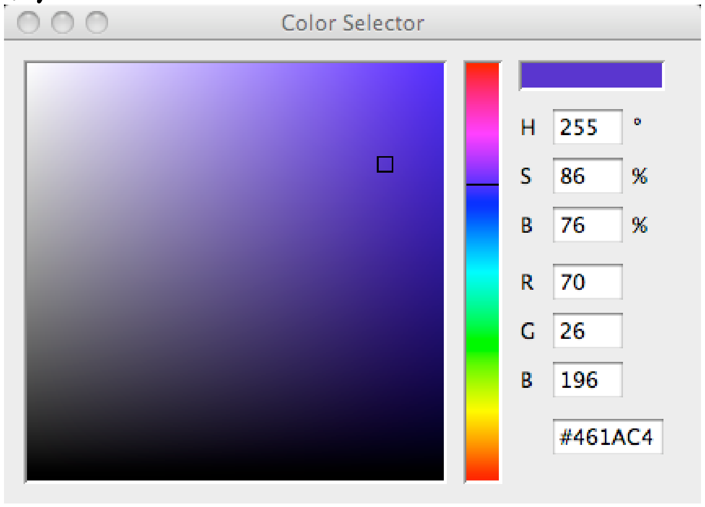
This told Processing to set the background color to zero red, full on green, and zero blue.

You should go to the API and see how to use the other functions (except colorMode( )) in your code.

	You may be thinking, "How do I know what numbers to use for the color amounts?" There are several answers: - One is to just guess and take whatever the numbers make - Another is to experiment and keep careful notes like the mad scientists in the old monster movies. - A third way is to use Processing. Below is the menu bar that Processing gives us:



There is a Tools menu item. Under Tools is a Color Selector option. If you choose this, you will see this:



H 255 °
S 86 %
B 76 %
R 70
G 26
B 196
#461AC4

You can use this to find the color you want. You move the line in the multicolored bar to the general area of the color you want and then click in the window on the actual color you are looking for (the little box) and Processing will tell you the RGB values to use.

The HSB values are used in a different color mode and the stuff in the lower right corner also represents the color. For now, you want to use the RGB values.

One function you may want to use is not listed in the color functions. This is the strokeWeight( ) function.

This sets the width of the stroke or the line that is drawn around the shapes you draw. The argument is the width of the line in pixels.

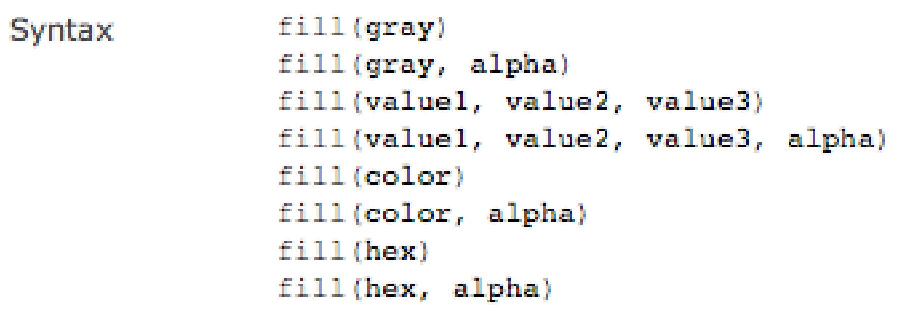
What about Black, White, and Gray

We can set a color to black (0), white (255), or any shade between black and white if we use just one number. Processing understands that if it sees just a single number when we are telling it what color we want to use, that we want black, white, or some shade of gray.

Twins, Triplets, Quadruplets, Octuplets . . .

What???

You know what we mean when we say two brothers or sisters are twins. Processing has sorta' the same thing… Let's look at the API for the fill function ( ):



```
Syntax          fill(gray)
                fill(gray, alpha)
                fill(value1, value2, value3)
                fill(value1, value2, value3, alpha)
                fill(color)
                fill(color, alpha)
                fill(hex)
                fill(hex, alpha)
```

There are eight fill( ) functions in this list - sorta' like eight siblings or octuplets.

That's right -- there are eight different ways to use fill( ) in our code. Each of these functions sets the color of the inside of the shapes we draw. So they all do the same thing. The difference is how we tell the fill( ) function what color we want to use.

Processing can figure out what color we want by looking at the arguments we use. If we put a single number between 0 and 255 in the parentheses, it knows we want black, white or some shade of gray.

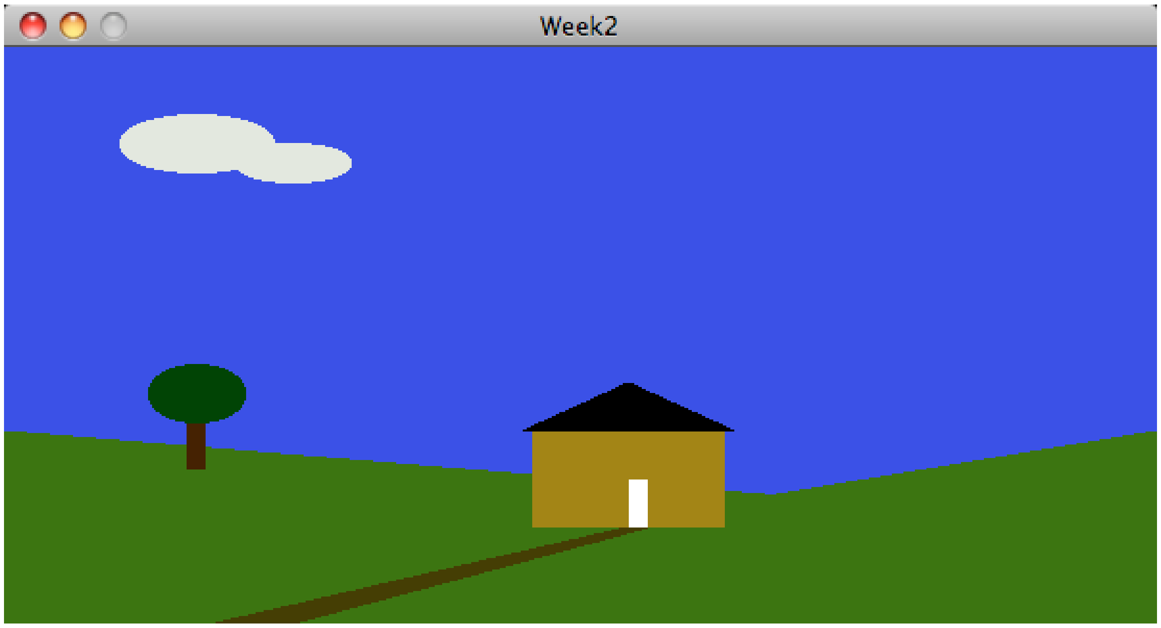
If we use three numbers between 0 and 255, it knows we want an RGB color.

We will talk about the other six siblings of fill in the list later. We have more than enough to use right now.

You need to do some tinkering with code and figure out how this stuff works. Here are some things you should try to figure out as you tinker.

- What is drawn first and what is drawn last in my program?
- What happens to any "old" shapes I have if I change the fill color?
- If I set the fill color to green, and draw a bunch of shapes, how many will be green?
- Is the order that I draw the shapes important?
- If I want to draw a small shape inside a big shape, what should I draw first?

Here is picture your teacher drew. Surly you can do much better…



Your Assignment for next time:

- Draw a Picture.
- Explore the functions in the API. Work with the functions in the 2D Primitives section under Shape and the functions in the Setting section under Color.
- Bring your code with you.

One last thing (teachers never let you go…)

Save your program first – this is very important.

Then, put the following line of code at the very end of your program. This must be the LAST line of code:

```
 saveFrame( "day2.jpg" );
```

Save your program again.

Now run your program.

The saveFrame( ) function actually takes a picture of your program's graphics window or frame and saves it in the folder that has your program file. The name of the picture will be day2.jpg.

You can take this picture and print it, mail it to someone as an attachment, or put it on your face book page.

If you print it, remember that it has a lot of color and could use up a lot of ink. Check with the folks at home before you print this.

We will electronically collect your prints next time. We do NOT want paper.

See you in a week…