**Introduction to Processing Workshop – Lesson 1**

**Learning Objectives:**

At the end of the lesson, students should be able to

* Use Processing Development Environment (PDE) to write Sketch
* Program basic 2D drawing in Sketch
* Use variables in Sketch
* Use repetition in Sketch
* Write basic function in Sketch

What is Processing?

“Processing is an open source programming language and environment for people who want to create images, animations, and interactions. Initially developed to serve as a software sketchbook and to teach fundamentals of computer programming within a visual context, Processing also has evolved into a tool for generating finished professional work. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production.”

([www.processing.org](http://www.processing.org))

You may download Processing from <http://www.processing.org/download/>

We are using version 2 for this course.

Choose the correct OS and install Processing.

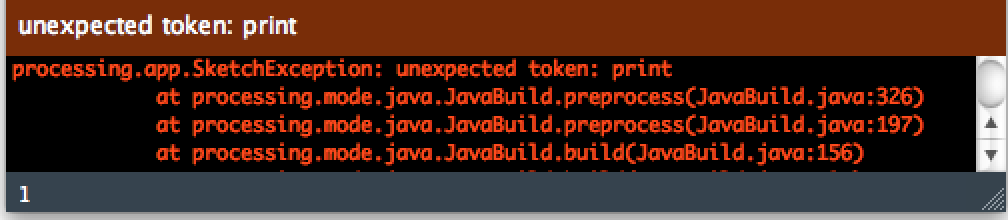
**Task 1: Start Processing from your OS.**

|  |  |
| --- | --- |
| Macintosh HD:Users:aaronyeo:Desktop:Screen Shot 2014-09-05 at 2.27.17 pm.png | Tool Bar  Play, Stop  New, Open, Save, Export to Applet  Message Area  Console  Text Editor |

The Text Editor is for you to type in the Processing code (also known as Sketch).

The Message area is to display a one-line message.

The Console area is to display more technical detail



**Task 2: Draw an Ellipse**

In the editor, type the following:

*ellipse(50, 50, 80, 80)*

and click on the play button in the toolbar

You will get an error!



This is because we need to end every statement with a semi-color.

Try to correct the code by placing the semi-colon behind the line of code.

*ellipse(50, 50, 80, 80)****;***

This line of code means, “draw an ellipse, with the center 50 pixels over from the left and 50 pixels down from the top, with a width and height of 80 pixels.”

|  |  |
| --- | --- |
| Macintosh HD:Users:aaronyeo:Desktop:Screen Shot 2012-07-30 at 8.34.45 PM.png | If you’ve typed everything correctly, you’ll see the ellipse image on the left. |

In programming you have to be very specific about the syntax. The Processing software isn’t always smart enough to know what you mean, and can be quite fussy about the placement of punctuation. You need to practice a lot so that it becomes your second nature.

Save you work and go on to next task.

**Task 3: Circles**

Open a new file and type in the codes below:

*void setup() {*

*size(480, 120);*

*smooth();*

*}*

*void draw() {*

*if (mousePressed) {*

*fill(0);*

*} else {*

*fill(255);*

*}*

*ellipse(mouseX, mouseY, 80, 80);*

*}*

This program creates a window that is 480 pixels wide and 120 pixels high.

It then starts drawing white circles at the position of the mouse.

When a mouse button is pressed, the circle color changes to black. We’ll explain more about the elements of this program in detail later. For now, run the code, move the mouse, and click to experience it.

To exit the program, click on the Stop button on the toolbar.

**Task 4: Standard Examples**

There are many examples done by people in Processing Development Environment.

|  |  |
| --- | --- |
| Macintosh HD:Users:aaronyeo:Desktop:Screen Shot 2012-07-30 at 8.46.35 PM.png | Go to file->Examples  From the Java Examples panel choose  Topics->Simulate->Flocking  Click on Flocking to load the Sketch  Click on play to execute the Sketch  Explore a few of the standard examples on your own. |

**Task 5: Draw a Window**

We use the size() function to set the Processing window

The size() function has two parameters: the first sets the width of the window and the second sets the height.

To draw a window that is 800 pixels width and 600 height:

*size(800, 600);*

Run this line of code to see the result. Put in different values to see what’s possible. Try very small numbers and numbers larger than your screen.

**Task 6: Draw a point**

Use the point() function to set the color of a single pixel

It has two parameters that define a position: the x-coordinate followed by the y-coordinate. To draw a little window and a point at the center of the screen, coordinate (240, 60):

*size(480, 120);*

*point(240, 60);*

**Task 7: Draw points**

Write a program that puts a point at each corner of the Display Window and one in the center.

**Task 8: Draw a line using Points**

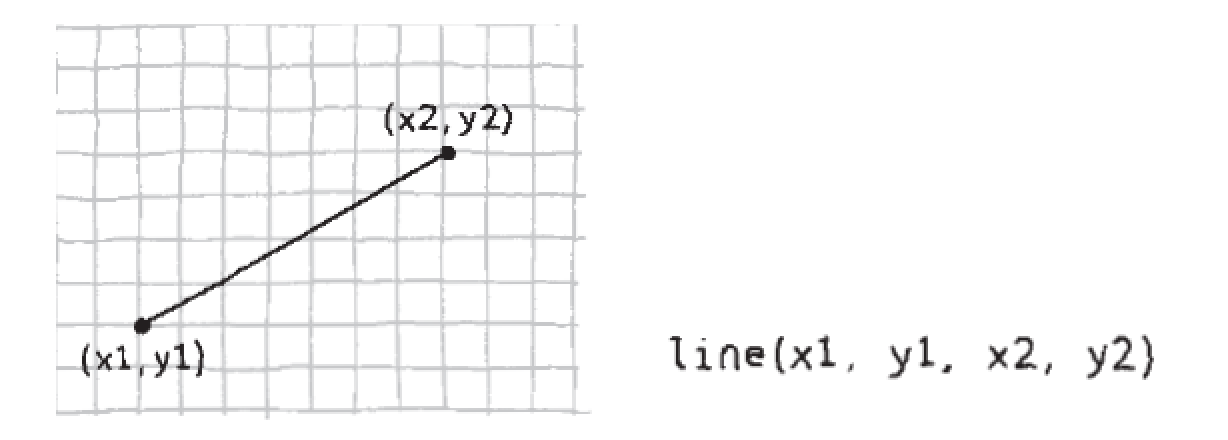
Draw horizontal, vertical, and diagonal lines by placing points side by side.

Is it a laborious work?

**Task 9: Draw a line**

There is a standard function built in processing to draw a line.

To draw a single line, we need four parameters: two for the starting location and two for the end.



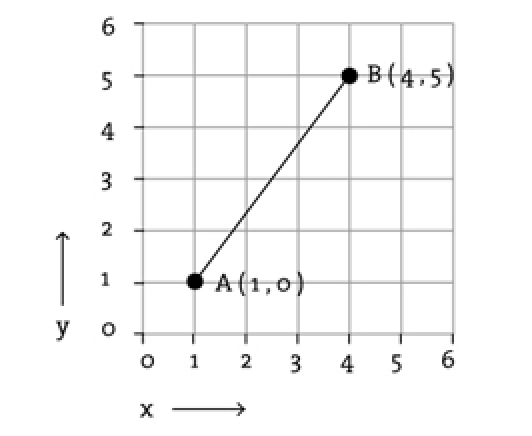
To draw a line between coordinate (20, 50) and (420,110), try:

*size(480, 120);*

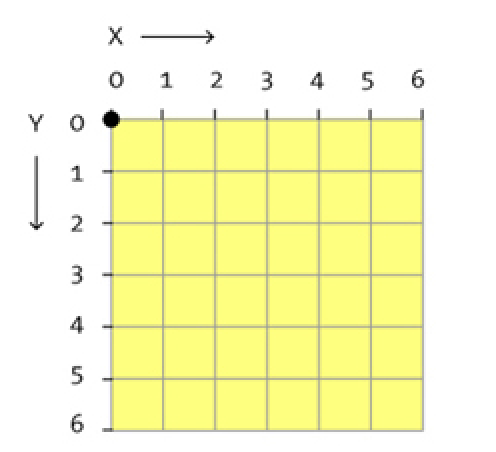
*line(20, 50, 420, 110);*

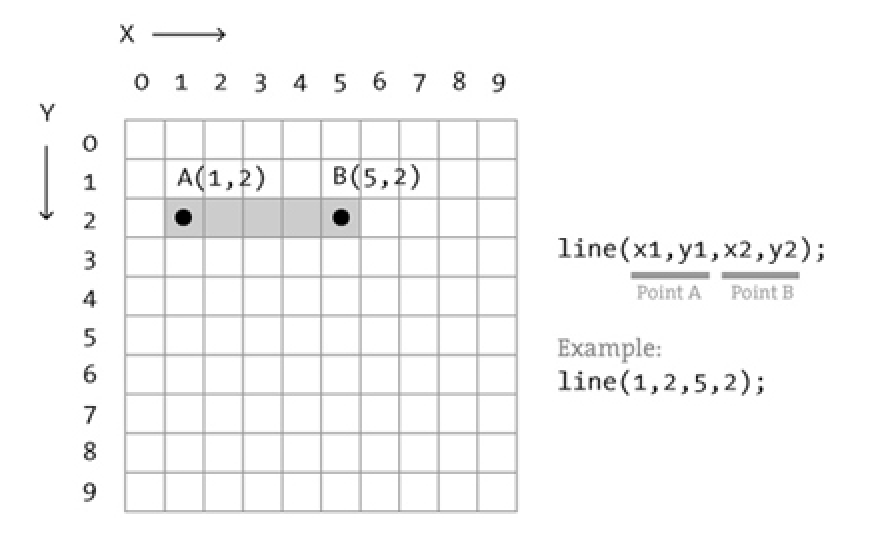
**Task 10: Understanding the Coordinate System in Processing**

In Math, the Cartesian coordinate system placed (0,0) in the bottom left with the y-axis pointing up and the x-axis pointing to the right (in the positive direction, negative down and to the left).



The coordinate system for pixels in a computer window, however, is reversed along the y-axis. (0,0) can be found at the top left with the positive direction to the right horizontally and down vertically.



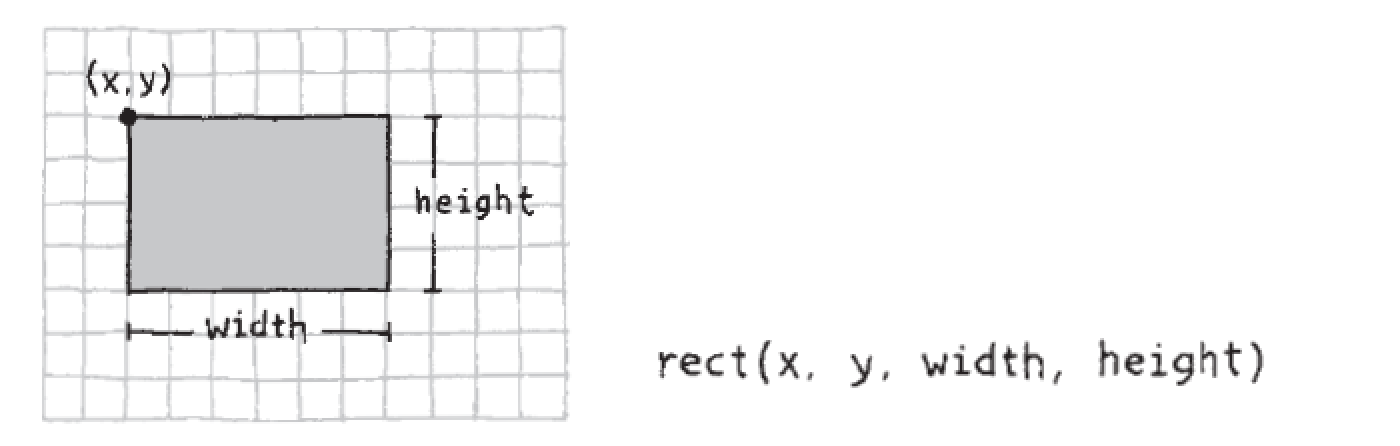


Note that the index starts from 0 and not 1.

* Screen size is 512x512
* Pixel range is 0-511

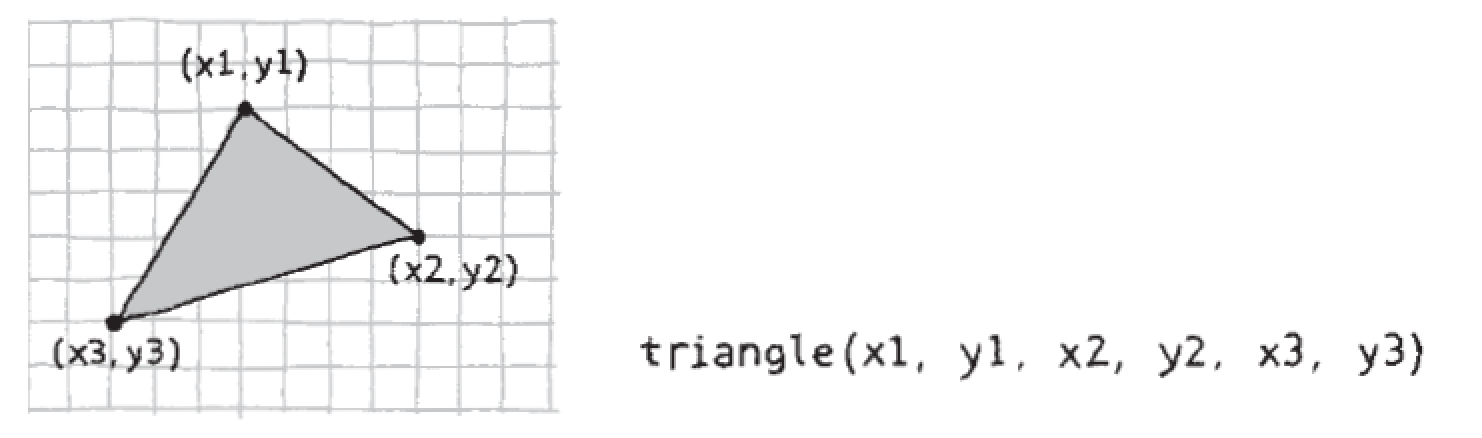
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **………………** | **511** |
| **1** |  |  |  | **………………** |  |
| **2** |  |  |  | **………………** |  |
| **3** |  |  |  | **………………** |  |
| **…..** |  |  |  | **………………** |  |
| **511** |  |  |  | **………………** |  |

**Task 11: Draw a rectangle**



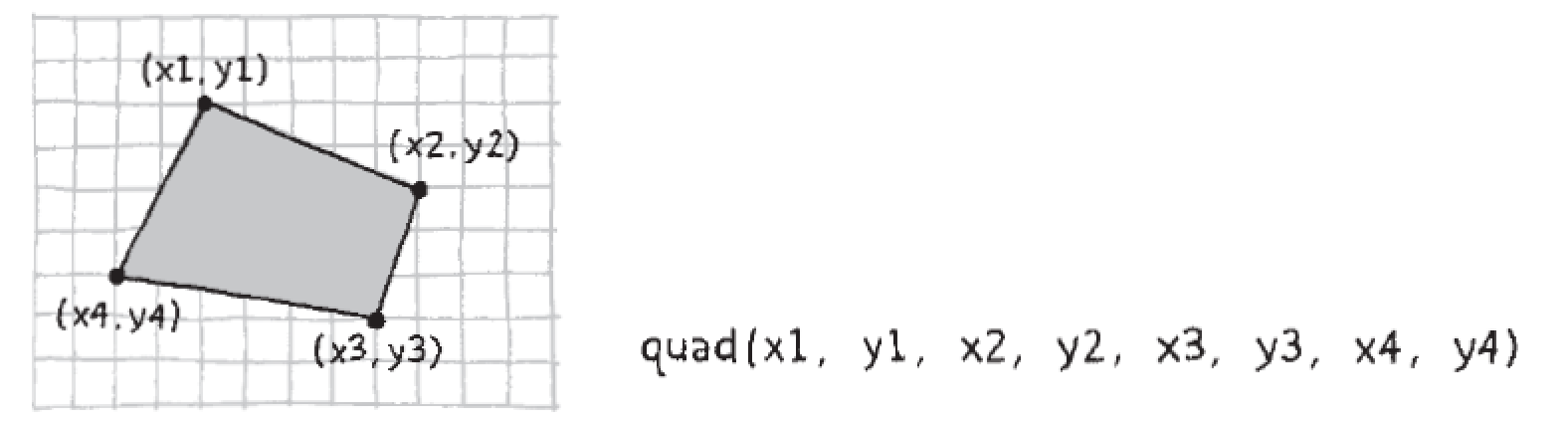
Try: rect(30, 20, 55, 55);

**Task 12: Draw a Triangle**



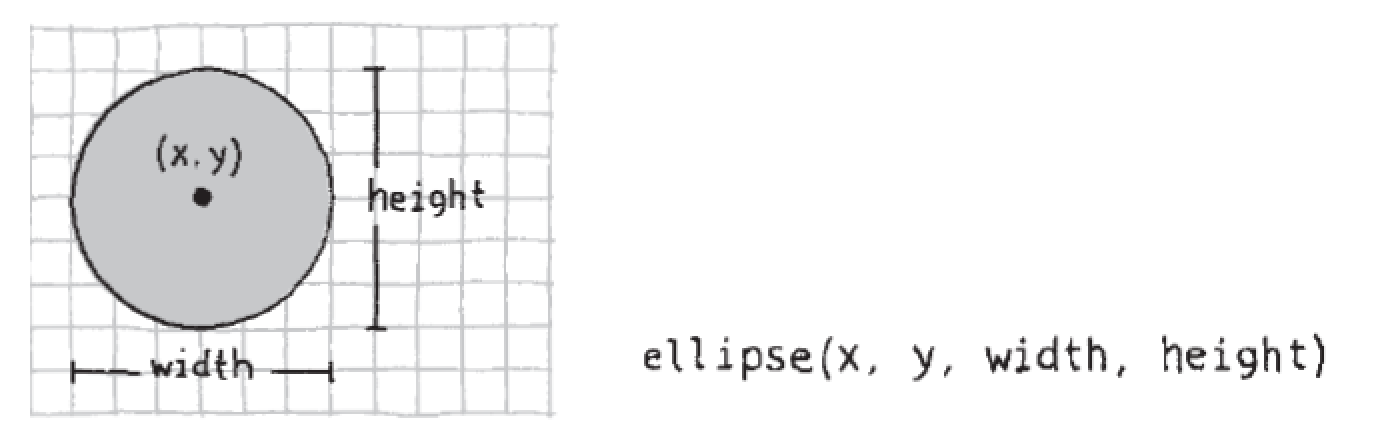
Try: triangle(30, 75, 58, 20, 86, 75);

**Task 13: Draw a Quadrilateral**



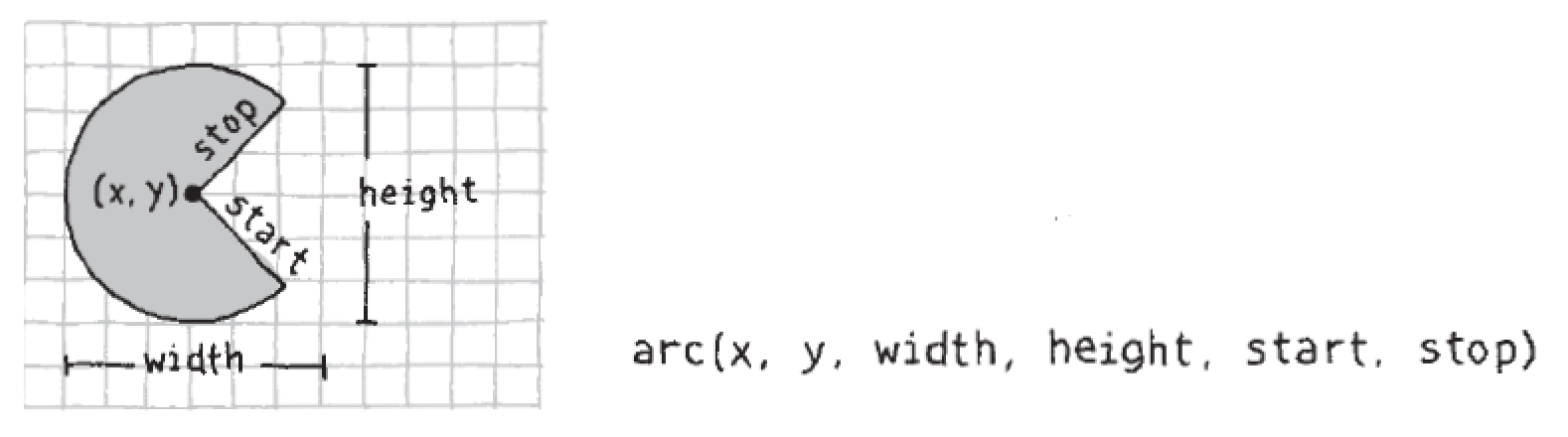
Try: quad(38, 31, 86, 20, 69, 63, 30, 76);

**Task 14: Draw an Ellipse**



Try: ellipse(56, 46, 55, 55);

**Task 15: Draw an Arc.**



Try:

arc(50, 55, 50, 50, 0, PI/2);

arc(50, 55, 60, 60, PI/2, PI);

arc(50, 55, 70, 70, PI, TWO\_PI-PI/2);

arc(50, 55, 80, 80, TWO\_PI-PI/2, TWO\_PI);

Note: The start and stop are in radian.

One PI is 180o,

Two PI is 360o

(If you are unsure of radian, please read up <http://en.wikipedia.org/wiki/Radian>)

You may also convert the degree to radian by using radian() function.

Try:

size(480, 120);

arc(90, 60, 80, 80, 0, radians(90));

arc(190, 60, 80, 80, 0, radians(270));

arc(290, 60, 80, 80, radians(180), radians(450));

arc(390, 60, 80, 80, radians(45), radians(225));

But note that most of the examples we see use radian instead of degree.

**Task 16: Drawing using basic shape**

The things around us are formed by basic shape like rectangles, circles and squares.

Try:

size(480, 120);

quad(158, 55, 199, 14, 392, 66, 351, 107);

triangle(347, 54, 392, 9, 392, 66);

triangle(158, 55, 290, 91, 290, 112);

**Task 17: Drawing Order**

When a program runs, the computer starts at the top and reads each line of code until it reaches the last line and then stops. If you want a shape to be drawn on top of all other shapes, it needs to follow the others in the code.

Try:

*size(480, 120);*

*ellipse(140, 0, 190, 190);*

*// The rectangle draws on top of the ellipse*

*// because it comes after in the code*

*rect(160, 30, 260, 20);*

Reversing the order of rect() and ellipse() to see the circle on top of the rectangle:

Try:

*size(480, 120);*

*rect(160, 30, 260, 20);*

*// The ellipse draws on top of the rectangle*

*// because it comes after in the code*

*ellipse(140, 0, 190, 190);*

**Task 18: Draw smooth line**

The smooth() function smooths the edges of lines drawn to the screen.

If smoothing is already turned on, use noSmooth() function to turn it off.

Try:

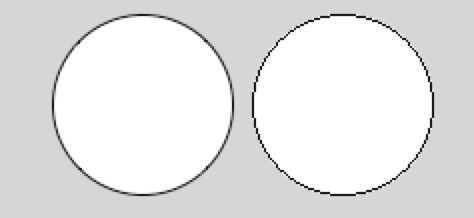
*size(480, 120);*

*smooth(); // Turns on smoothing*

*ellipse(140, 60, 90, 90);*

*noSmooth(); // Turns off smoothing*

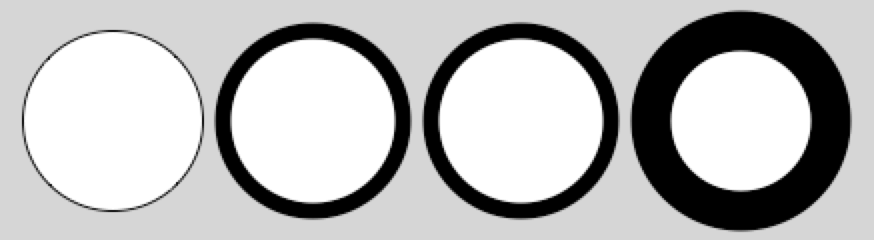
*ellipse(240, 60, 90, 90);*



**Task 19: Set stroke weight**

The default stroke weight is a 1 pixel.

Use strokeWeight() to set the stroke weight.



Try:

*size(480, 120);*

*smooth();*

*ellipse(75, 60, 90, 90);*

*strokeWeight(8); // Stroke weight to 8 pixels*

*ellipse(175, 60, 90, 90);*

*ellipse(279, 60, 90, 90);*

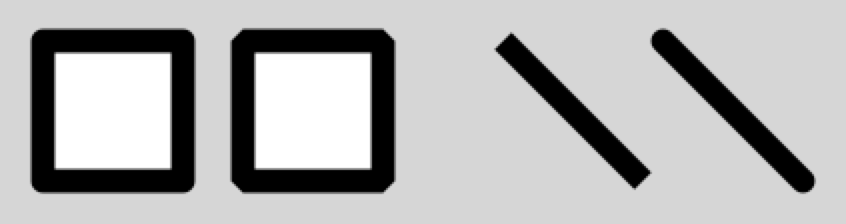
*strokeWeight(20); // Stroke weight to 20 pixels*

*ellipse(389, 60, 90, 90);*

**Task 20: Set stroke attributes**

Use strokeJoin() to change the way lines are joined (how the corners look).

The strokeCap() to change how lines are drawn at the beginning and end:



Try:

*size(480, 120);*

*smooth();*

*strokeWeight(12);*

*strokeJoin(ROUND); // Round the stroke corners*

*rect(40, 25, 70, 70);*

*strokeJoin(BEVEL); // Bevel the stroke corners*

*rect(140, 25, 70, 70);*

*strokeCap(SQUARE); // Square the line endings*

*line(270, 25, 340, 95);*

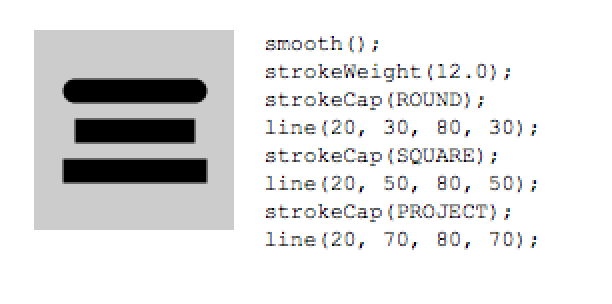
*strokeCap(ROUND); // Round the line endings*

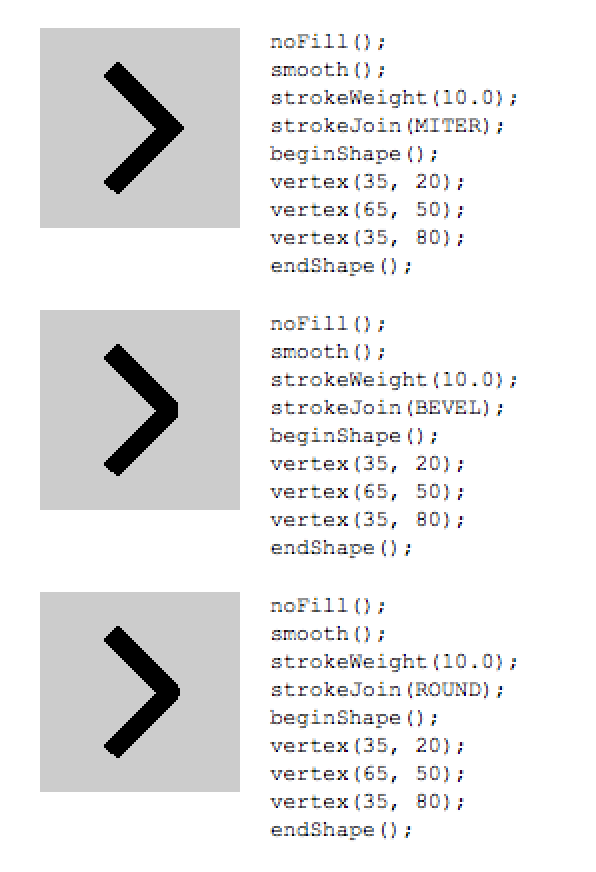
*line(350, 25, 420, 95);*

When any of these attributes are set, all shapes drawn afterward are affected.

For example when stroke weight is set to 12, all the following rect and line has a stroke weight of 12*.*

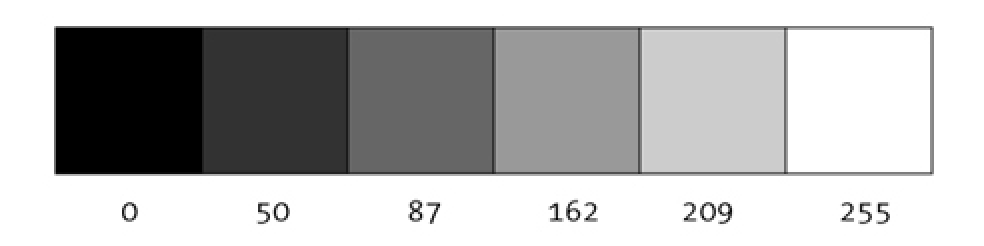
[*http://www.processing.org/reference/strokeCap\_.html*](http://www.processing.org/reference/strokeCap_.html)

**

*http://www.processing.org/reference/strokeJoin\_.html*

**Task 21: Understanding gray scale**

Figure below shows how the values from 0 to 255 map to different gray levels.



Source: <http://processing.org/learning/color/>

The values of the parameters are in the range of 0 to 255, where 255 is white, 128 is medium gray, and 0 is black.

For more info: <http://en.wikipedia.org/wiki/Grayscale>

Use the background(), fill(), and stroke() to change the colour to different shade of grey.

Try:

*void setup() {*

*size(480, 120);*

*frameRate(20);*

*}*

*int colourCode=255;*

*void draw() {*

*background(colourCode);*

*colourCode--;*

*print ("Gray Scale Code:"+colourCode+"\n");*

*if(colourCode==0)*

*{*

*noLoop();*

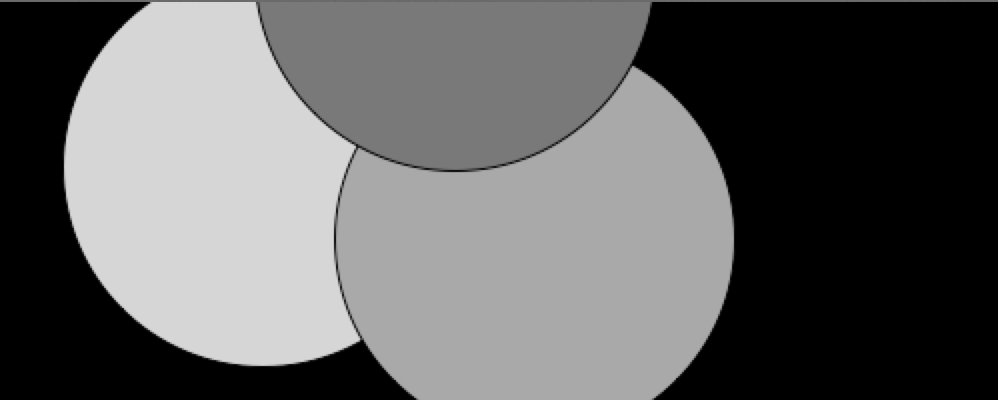
*print ("Stop");*

*}*

*}*

The above Sketch is to keep changing the background colour with different gray scale colour code every 20 frames/sec. It will stop when the colour code reaches 0 (i.e. pure black). Ignore how the setup() and draw() function works for now. We will go into that later.

**Task 22: Painting with Gray scale**



Try:

*size(500, 200);*

*smooth();*

*background(0); // Black*

*fill(204); // Light gray*

*ellipse(132, 82, 200, 200); // Light gray circle*

*fill(153); // Medium gray*

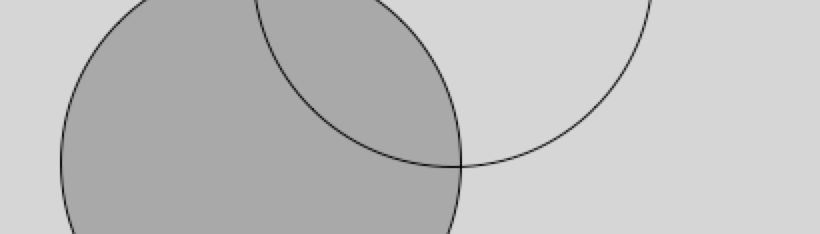
*ellipse(268, 118, 200, 200); // Medium gray circle*

*fill(102); // Dark gray*

*ellipse(228, -16, 200, 200); // Dark gray circle*

**Task 23: Control Fill and Stroke**

You can disable the stroke so that there’s no outline with noStroke() and you can disable the fill of a shape with noFill()



Try:

*size(480, 120);*

*smooth();*

*fill(153); // Medium gray*

*ellipse(132, 82, 200, 200); // Gray circle*

*noFill(); // Turn off fill*

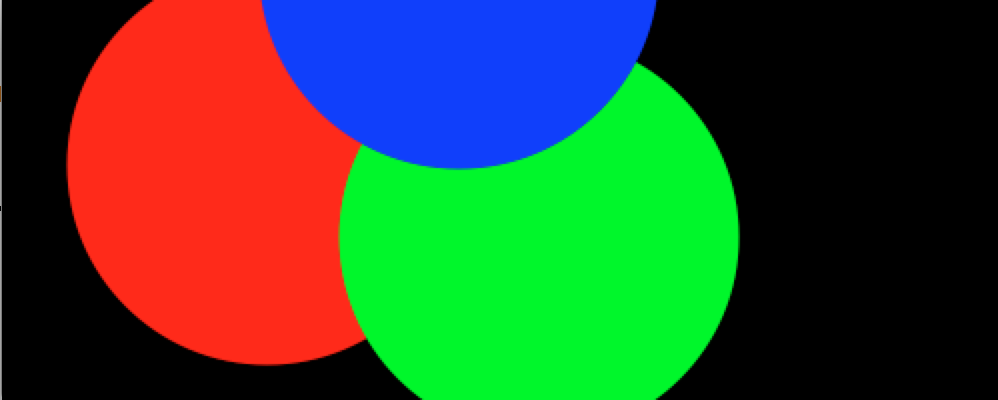
*ellipse(228, -16, 200, 200); // Outline circle*

*noStroke(); // Turn off stroke*

*ellipse(268, 118, 200, 200); // Doesn’t draw!*

**Task 24: Draw with Colour**

To move beyond grayscale values, you use three parameters to specify the red, green, and blue components of a color.



Try:

*size(500, 200);*

*noStroke();*

*smooth();*

*background(0);*

*fill(255, 0, 0); // Red color*

*ellipse(132, 82, 200, 200);*

*fill(0, 255, 0); // Green color*

*ellipse(268, 118, 200, 200);*

*fill(0, 0, 255); // Blue color*

*ellipse(228, -16, 200, 200);*

The colour mode used here is referred to as RGB color.

It comes from how computer define colors on the screen.

The three numbers stand for the values of red, green, and blue, and they range from 0 to 255.

For more info: <http://en.wikipedia.org/wiki/RGB_color_model>

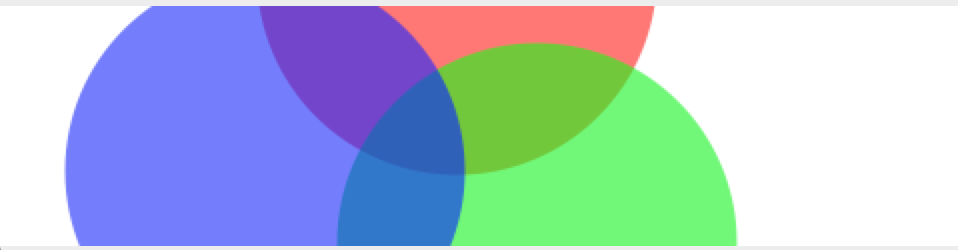
**Task 25: Set Transparency**

To control the colour transparency, add a fourth parameter (alpha value - 0 to 255) to fill() or stroke().

Value 0 - Total transparent (it won’t display),

Value 255 - Total opaque

Values between these extremes cause the colors to mix on screen.



Try

*size(480, 120);*

*noStroke();*

*smooth();*

*background(255);*

*// Red color with transparency of 160*

*fill(255, 0, 0, 160);*

*ellipse(228, -16, 200, 200);*

*// Green color with transparency of 160*

*fill(0, 255, 0, 160);*

*ellipse(268, 118, 200, 200);*

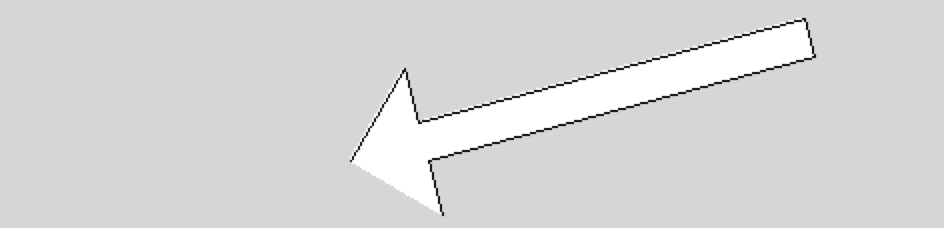
*// Blue color with transparency of 160*

*fill(0, 0, 255, 160);*

*ellipse(132, 82, 200, 200);*

**Task 26: Draw an arrow**

The beginShape() function signals the start of a new shape. The vertex() function is used to define each pair of x- and y-coordinates for the shape. Finally, endShape() is called to signal that the shape is finished.



Try:

*size(480, 120);*

*beginShape();*

*vertex(180, 82);*

*vertex(207, 36);*

*vertex(214, 63);*

*vertex(407, 11);*

*vertex(412, 30);*

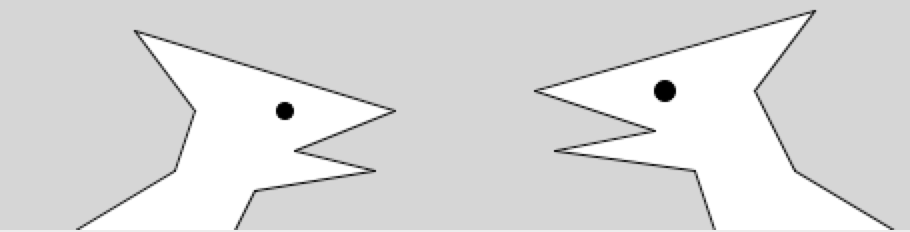
*vertex(219, 82);*

*vertex(226, 109);*

*endShape();*

**Task 27: Draw creatures**

Use vertex() to make shapes with complex outlines.



*size(480, 120);*

*smooth();*

*// Left creature*

*beginShape();*

*vertex(50, 120);*

*vertex(100, 90);*

*vertex(110, 60);*

*vertex(80, 20);*

*vertex(210, 60);*

*vertex(160, 80);*

*vertex(200, 90);*

*vertex(140, 100);*

*vertex(130, 120);*

*endShape();*

*fill(0);*

*ellipse(155, 60, 8, 8);*

*// Right creature*

*//Draw the right creature here*

**Task 28: Understanding Variables**

A variable stores a value in memory so that it can be reused.

Variable can be used many times within a single program, and the value is easily changed when the program is running.

For instance, when you make the y-coordinate and diameter for the two circles in this example into variables, the same values are used for each ellipse:

Try:

*size(480, 120);*

*smooth();*

*int y = 60; //declare a variable integer y*

*int d = 80; //declare a variable integerdy*

*ellipse(75, y, d, d); // Left*

*ellipse(175, y, d, d); // Middle*

*ellipse(275, y, d, d); // Right*

Change the y and d variables will change all three ellipses position:

Try

*size(480, 120);*

*smooth();*

*int y = 100; //only change this*

*int d = 130;//and this*

*ellipse(75, y, d, d); // Left*

*ellipse(175, y, d, d); // Middle*

*ellipse(275, y, d, d); // Right*

If no variables are used, we need to change the y-coordinate used in the code three times and the diameter six times.

**Task 29: Declaring Variables**

To use a variable, it must first be declared.

When declaring a variable, we specify its data type (such as int) and name.

After the data type and name are set, a value can be assigned to the variable:

*int x; // Declare x as an int variable*

*x = 12; // Assign a value to x*

This code does the same thing, but is shorter:

*int x = 12; // Declare x as an int variable and assign a value*

All variable names in a program must be unique.

*int x; // Declare x as an int variable*

*int x = 12; // ERROR! Can’t have two variables called x here*

**Task 30: Special variables in Processing (width and height)**

Processing stores special system variables (e.g. width and height) while it runs.

The width and height of the window are stored in variables called width and height. These values are set by the size() function. They can be used to draw elements relative to the size of the window, even if the size() line changes.

*size(480, 120);//* Change the values in size() function and see how the program react

*smooth();*

*line(0, 0, width, height); // Line from (0,0) to (480, 120)*

*line(width, 0, 0, height); // Line from (480, 0) to (0, 120)*

*ellipse(width/2, height/2, 60, 60);*

**Task 31: Understanding basic mathematics arithmetic in Processing**

Symbols like +, –, and \* are called operators.

The operators for the basic math operations are:

+ Addition

– Subtraction

\* Multiplication

/ Division

= Assignment

Try:

*size(480, 120);*

*int x = 25;*

*int h = 20;*

*int y = 25;*

*rect(x, y, 300, h); // Top*

*x = x + 100;*

*rect(x, y + h, 300, h); // Middle*

*x = x - 250;*

*rect(x, y + h\*2, 300, h); // Bottom*

Frequent calculations in programming have been developed into “shortcuts”.

Try:

int x =0;

x+= 10; // This is the same as x = x + 10

int y =0;

y -= 15; // This is the same as y = y – 15

print (x);

print (y);

Try:

int x=0;

x++; // This is the same as x = x + 1

int y=0;

y--; // This is the same as y = y – 1

print (x);

print (y);

**Task 32: Understanding Repetition**

A for loop code makes it possible to run a line of code more than once.

This example has the type of pattern that can be simplified with a for loop:



|  |  |
| --- | --- |
| No Repetition | With Repetition |
| *size(480, 120);*  *smooth();*  *strokeWeight(8);*  *line(20, 40, 80, 80);*  *line(80, 40, 140, 80);*  *line(140, 40, 200, 80);*  *line(200, 40, 260, 80);*  *line(260, 40, 320, 80);*  *line(320, 40, 380, 80);*  *line(380, 40, 440, 80);* | *size(480, 120);*  *smooth();*  *strokeWeight(8);*  *for (int i = 20; i < 400; i += 60) {*  *line(i, 40, i + 60, 80);*  *}* |

The syntax of a for loop is

|  |
| --- |
| for (<initialization>;<guard>;<final expression>)  {  <body>  } |

|  |
| --- |
| //print \* five time using a for loop  for(int i=0;i<5;i=i+1)  {  print ("\*");  } |

When the program enters a “for” loop, the first thing Processing does is to perform “initialization”.

In the above example it creates a variable “i” and assigns the value 0 to it (int i = 0)

The initialization is only performed once.

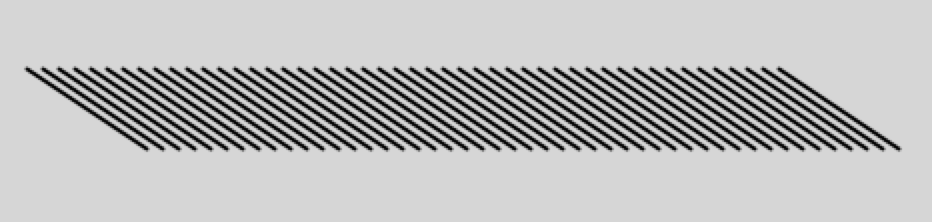
After the initialization, the next statement Processing will look at is “guard”. If the “guard” is true then it will enter the “body”. If the “guard” is false, then it will exit the “for” loop.

Assume that the “guard” is true, Processing will enter the “body” and executes all the codes in the “body”. When Processing finishes executing all the codes, it will go back up to the “for” loop and execute the “final expression”.

After the “final expression” has been executed, Processing will move to “guard” again to check if the “guard” is still true. If it is still true, it will enter the “body” again and the process continues until the “guard” becomes false.



The reason for working with a for loop is the ability to make quick changes to the code. Because the code inside the block is typically run multiple times, a change to the block is magnified when the code is run.



Try:

*size(480, 120);*

*smooth();*

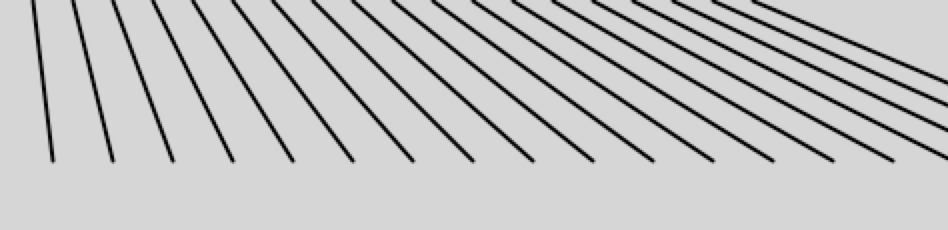
*strokeWeight(2);*

*for (int i = 20; i < 400; i += 8) {*

*line(i, 40, i + 60, 80);*

*}*

**Task 33: Fanning out lines**



Try:

*size(480, 120);*

*smooth();*

*strokeWeight(2);*

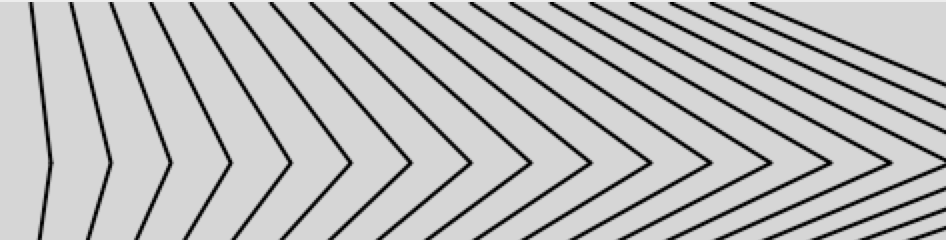
*for (int i = 20; i < 400; i += 20) {*

*line(i, 0, i + i/2, 80);*

*}*

**Make changes to make different grayscale colour for each line.**

**Task 34: Kinking the lines**



size(480, 120);

smooth();

strokeWeight(2);

for (int i = 20; i < 400; i += 20) {

line(i, 0, i + i/2, 80);

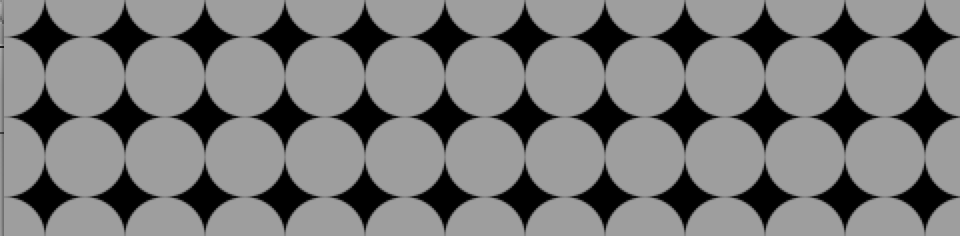
line(i + i/2, 80, i\*1.2, 120);

}

**Make changes so that the colour for each line is different.**

**Task 35: Nested Repetition**

When one for loop is embedded inside another, the number of repetitions is multiplied.



Try:

*size(480, 120);*

*background(0);*

*smooth();*

*noStroke();*

*for (int y = 0; y <= height; y += 40) {*

*for (int x = 0; x <= width; x += 40) {*

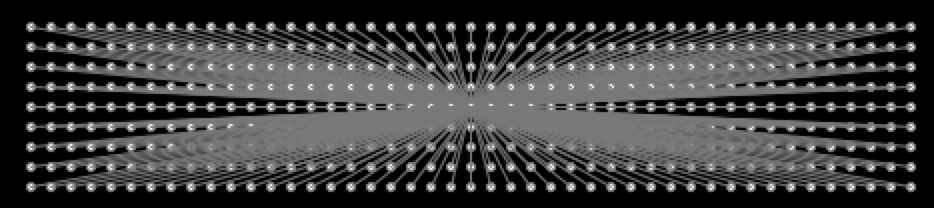
*fill(255, 140);*

*ellipse(x, y, 40, 40);*

*}*

*}*

**Task 36: Pins and Lines**



Try:

*size(480, 120);*

*background(0);*

*smooth();*

*fill(255);*

*stroke(102);*

*for (int y = 20; y <= height-20; y += 10) {*

*for (int x = 20; x <= width-20; x += 10) {*

*ellipse(x, y, 4, 4);*

*// Draw a line to the center of the display*

*line(x, y, 240, 60);*

*}*

*}*

**Task 37: Understanding Static and Active Mode in Processing**

**Static Mode:**

We have been using the “static” mode of Processing.

**Active Mode:**

In “active” mode we can create more interactive program by having Processing keep calling a function draw() until program exits.

There are two main functions that we need to code in active mode; setup() and draw().

The setup() function is called during the initial execution of the program. It is only called once.

After which Processing will call draw() repeatedly until the program terminates.

The rate at which draw() is being called is determined by the frameRate.

Try:

*void setup() //this function is only called once*

*{*

*size(480, 120);*

*frameRate(30); //set the frequency draw() is being called to 30 calls/sec or frame/sec*

*}*

*int colourCode=255;*

*void draw() //this function will be called repeated after setup() is done at 30 calls/sec*

*{*

*background(colourCode);*

*colourCode--;*

*print ("Gray Scale Code:"+colourCode+"\n");*

*if(colourCode==0)*

*{*

*noLoop(); //this will terminate the calling of draw()*

*print ("Stop");*

*}*

*}*

The setup() block runs once, and the draw() block runs repeatedly.

The setup() function is used for any initialization; in this case, setting the screen size and setting the frame rate.

The size() function must always be the first line inside setup().

The draw() block is used to handle animation.

The animation or the repeated calling of draw() can be terminated by noLoop().

**Task 38: Understanding function**

A function is a block of code that is reusable.

A function can be executed from different points in a Program.

As there are no square() function in Processing, we can define our own function to draw a square.

Try:

*void setup(){*

*size(300,200);*

*background(174);*

*}*

*void draw(){*

*square(4,6,56,56,43,44);*

*square(45,43,56,45,43,33);*

*square(145,43,100,45,243,134);*

*}*

*void square(int x,int y, int len, int R, int G, int B)*

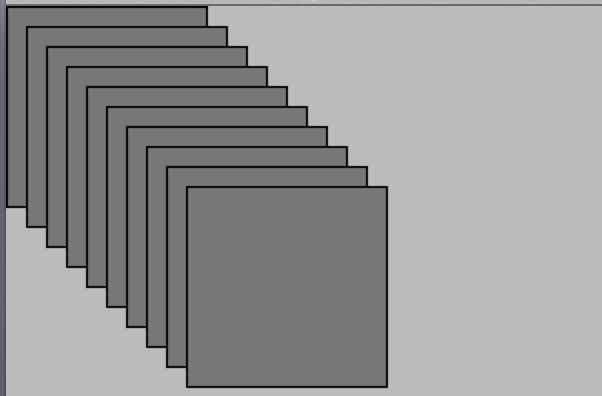
*{*

*fill(R,G,B);*

*rect(x,y,len,len);*

*}*

Replace the code in draw() with a for loop and draw 10 squares such as below:



**Task 39: Circle function**

Write a function for drawing circle and test the function.

**Final Task:**

Draw the outline of SIM building from the front view.

****