**Introduction to Processing Workshop – Lesson 2**

**Learning Objectives:**

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

* Get user input from Keyboard and Mouse
* Draw basic shapes in Sketch
* Animate basic motion in Sketch
* Import image in Sketch
* Draw text with different fonts.

**Task 1: Recapping the draw() function**

To see how draw() works, run this example:

*void draw() {*

*// Displays the frame count to the Console*

*println("I’m drawing");*

*println(frameCount);*

*}*

You’ll see the following:

I’m drawing

1

I’m drawing

2

I’m drawing

3

...

The code within the draw() block runs from top to bottom, then repeats until you quit the program by clicking the Stop button or closing the window.

Each trip through draw() is called a frame. (The default frame rate is 60 frames per second, but this can be changed by the frameRate() function)

The frameCount is a special variable in Processing to keep track of the number of times draw() has been called.

**Task 2: Recapping the setup() function**

The setup() function runs just once when the program starts:

*void setup() {*

*println("I’m starting");*

*}*

*void draw() {*

*println("I’m running");*

*}*

When this code is run, the following is written to the Console:

I’m starting

I’m running

I’m running

...

The text “I’m running” continues to write to the Console until the program is stopped.

**Task 3: Understanding Global variables**

We can create a variable outside a function so that we can use it anywhere.

This kind of variable is called global variables, because they can be used “globally” in the program.

Try:

*void setup()*

*{ count = 0;*

*}*

*int count;*

*void draw() {*

*println(count);*

*count++;*

*if(count == 10)*

*{noLoop();*

*}*

*}*

**Task 4: Using setup() and draw() for animation**

Try:

*int x = 280;*

*int y = 140;*

*int diameter = 380;*

*void setup() {*

*size(480, 320);*

*smooth();*

*fill(102);*

*}*

*void draw() {*

*background(204);*

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

*}*

Note that the draw() are being called repeated with the drawing of the same ellipse

We can modify the variable diameter to make some very simple animation.

Try:

*int x = 280;*

*int y = 140;*

*int diameter = 380;*

*void setup() {*

*size(480, 320);*

*smooth();*

*fill(102);*

*}*

*void draw()*

*{*

*background(204);*

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

*diameter--;*

*if(diameter ==0)*

*noLoop();*

*}*

**Task 5: Track the mouse**

The mouseX variable stores the x-coordinate, and the mouseY variable stores the y-coordinate:

Try:

*void setup() {*

*size(480, 120);*

*fill(0, 102);*

*smooth();*

*noStroke();*

*}*

*void draw() {*

*ellipse(mouseX, mouseY, 9, 9);*

*}*

**Task 6: The dot follows you**

In this example, a new circle is added to the window each time the code in draw() is run. To refresh the screen and only display the newest circle, place a background() function at the beginning of draw() before the shape is drawn:

Try:

*void setup() {*

*size(480, 120);*

*fill(0, 102);*

*smooth();*

*noStroke();*

*}*

*void draw() {*

*background(204);*

*ellipse(mouseX, mouseY, 9, 9);*

*}*

The background() function clears the entire window, so be sure to always place it before other functions inside draw(); otherwise, the shapes drawn before it will be erased.

**Task 7: Drawing**

The pmouseX and pmouseY variables store the position of the mouse at the previous frame. Like mouseX and mouseY, these special variables are updated each time draw() runs. When combined, they can be used to draw continuous lines by connecting the current and most recent location:

void setup() {

size(480, 120);

strokeWeight(4);

smooth();

stroke(0, 102);

}

void draw() {

line(mouseX, mouseY, pmouseX, pmouseY);

}

**Task 8: Understanding mouse click**

Processing keeps track if the mouse button is pressed.

The mousePressed variable is a boolean (true/false) data type.

The value of mousePressed is true when a button is pressed.

Try:

*void setup() {*

*size(240, 120);*

*smooth();*

*strokeWeight(30);*

*}*

*void draw() {*

*background(204);*

*stroke(102);*

*line(40, 0, 70, height);*

*if (mousePressed) {*

*stroke(0);//set to black*

*}*

*line(0, 70, width, 50);*

*}*

**Task 9: Tracking mouse click**

Processing tracks which button is pressed. The mouseButton variable can be one of three values: LEFT, CENTER, or RIGHT.

Try:

*void setup() {*

*size(120, 120);*

*smooth();*

*strokeWeight(30);*

*}*

*void draw() {*

*background(204);*

*stroke(102);*

*line(40, 0, 70, height);*

*if (mousePressed)*

*{*

*if (mouseButton == LEFT) {*

*stroke(255);*

*} else {*

*stroke(0);*

*}*

*line(0, 70, width, 50);*

*}*

*}*

**Task 10: Growing circle**

In this example, when the cursor is within the circle it grows.

We use the dist() function to get the distance from the center of the circle to the cursor, then we test to see if that distance of the cursor to the center of the circle is less than the radius of the circle . If it is, the cursor is inside the circle.

*int x = 120;*

*int y = 60;*

*int radius = 12;*

*void setup() {*

*size(240, 120);*

*smooth();*

*}*

*void draw() {*

*background(204);*

*float d = dist(mouseX, mouseY, x, y);*

*if (d < radius) {*

*radius++;*

*fill(0);*

*} else {*

*fill(255);*

*}*

*ellipse(x, y, radius\*2, radius\*2);*

*}*

**Task 11: Detecting key pressed**

In this example, the second line is drawn only when a key is pressed:

*void setup() {*

*size(240, 120);*

*smooth();*

*}*

*void draw() {*

*background(204);*

*line(20, 20, 220, 100);*

*if (keyPressed) {*

*line(220, 20, 20, 100);*

*}*

*print(key);*

*}*

The value of the key that has been pressed is stored in the special variable “key”.

**Task 12: Writing text**

This example introduces the textSize() function to set the size of the letters, the textAlign() function to center the text on its x-coordinate, and the text() function to draw the letter.

*void setup() {*

*size(120, 120);*

*textSize(64);*

*textAlign(CENTER);*

*}*

*void draw() {*

*background(0);*

*text(key, 60, 80);*

*}*

Note that the key is of a data type “char”, which means a “character”.

The char data type is specified by single quotes, unlike a string value, which is distinguished by double quotes

char c = 'A'; // Declares and assigns 'A' to the variable c

And these attempts will cause an error:

char c = "A"; // Error! Can't assign a String to a char

char h = A; // Error! Missing the single quotes from 'A'

**Task 13: Check for specific keys**

In this example, we test for an H or N to be typed.

Try:

*void setup() {*

*size(120, 120);*

*smooth();*

*}*

*void draw() {*

*background(204);*

*if (keyPressed) {*

*if ((key == 'h') || (key == 'H')) {*

*line(30, 60, 90, 60);*

*}*

*else if ((key == 'n') || (key == 'N')) {*

*line(30, 20, 90, 100);*

*}*

*}*

*line(30, 20, 30, 100);*

*line(90, 20, 90, 100);*

*}*

We combine the two tests together with a logical OR, the || symbol.

In plain English, it says, “If the ‘h’ key is pressed OR the ‘H’ key is pressed.”

**Task 14: Moving with Arrow Keys**

The following example shows how to check for the left or right arrow keys to move a rectangle:

*int x = 215;*

*void setup() {*

*size(480, 120);*

*}*

*void draw() {*

*if (keyPressed && (key == CODED)) { // If it’s a coded key*

*if (keyCode == LEFT) { // If it’s the left arrow*

*x--;*

*} else if (keyCode == RIGHT) { // If it’s the right arrow*

*x++;*

*}*

*}*

*rect(x, 45, 50, 50);*

*}*

First, check if the key that’s been pressed is a coded key,

Then check the code with the keyCode variable to see which key it is.

The most frequently used keyCode values are ALT, CONTROL, and SHIFT, as well as the arrow keys, UP, DOWN, LEFT, and RIGHT.

**Task 15: Load an image**

Preparation work:

1. Open a new a Processing file (Sketch).
2. Save the empty Sketch to the default Processing folder with a new name (e.g. LoadImage).
3. Go to Processing->Preferences and note down the Sketch location.
4. From Window Explorer or Finder, go to Sketch location
5. You should be able to see all the past Sketch project and the LoadImage project folder.
6. Copy the media folder to the Sketch location folder.

In the LoadImage Sketch try:

*PImage img; // Create a PImage variable to store the image.*

*void setup() {*

*size(480, 120);*

*//Load the image into the variable with loadImage().*

*img = loadImage("../media/lunar.jpg");*

*}*

*void draw() {*

*image(img, 0, 0); //display the img to the window using image()*

*}*

You should get a window with the image loaded.

image(img, x, y) - **x** and **y** parameters define the location of the image from its upper-left corner.

**Task 16: Load two images**

This example loads two images and places them side-by-side.

*PImage img1;*

*PImage img2;*

*void setup() {*

*size(480, 120);*

*img1 = loadImage("../media/lunar.jpg");*

*img2 = loadImage("../media/capsule.jpg");*

*}*

*void draw() {*

*image(img1, -120, 0);*

*image(img1, 130, 0, 240, 120);*

*image(img2, 300, 0, 240, 120);*

*}*

image(img, x, y, width, height);

The **img** parameter specifies the image to display and the **x** and **y** parameters define the location of the image from its upper-left corner. The image is displayed at its original size unless the **width** and **height** parameters specify a different size.

-0,0

-200,0

130,0

120

240

300,0

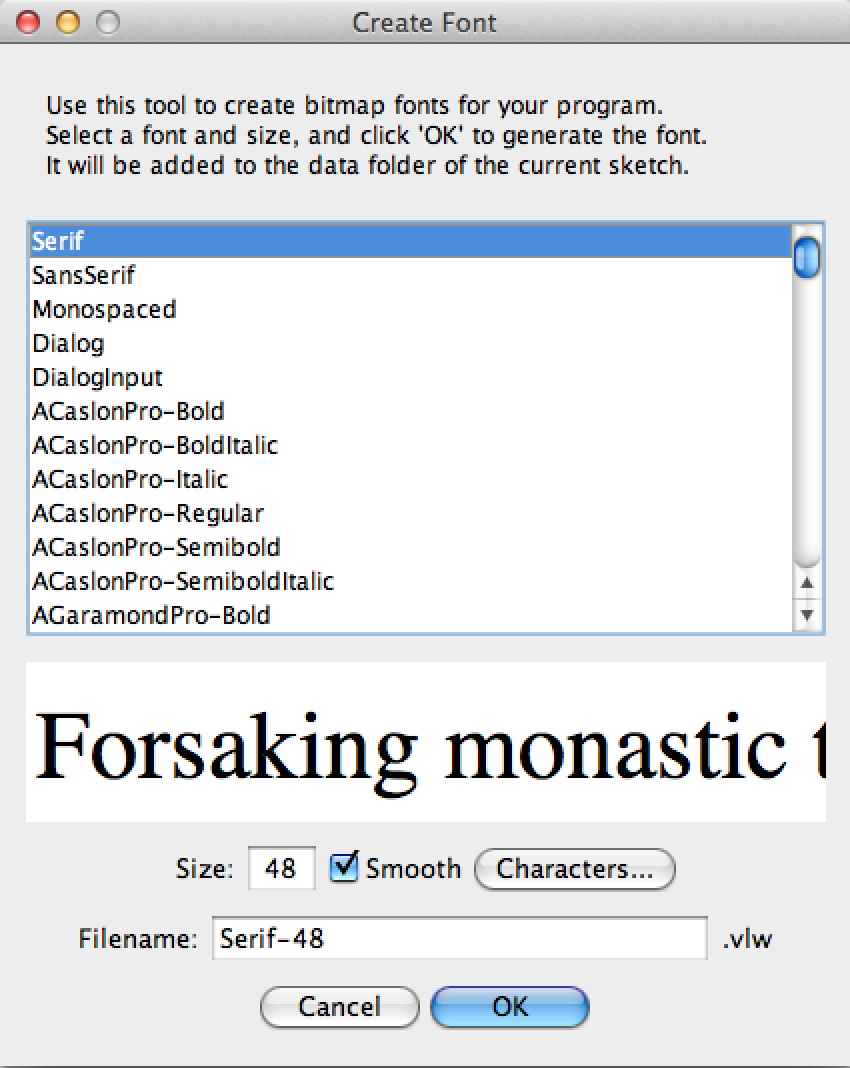
120

240

**Task 17: Drawing with Font**

Before displaying text in a different font, we need to convert one of the fonts on the computer to the VLW format.

To do this, select Create Font from the Tools menu to open the dialog box. Specify the font you want to convert, as well as the size and whether you want it to be smooth (anti-aliased).



For this exercise, create a ComicSansMS font with size 48. Save the font file as ComicSansMS-48.vlw. The file will be saved under the current Sketch directory data folder.

Draw a string using that font.

Try:

*PFont font;*

*void setup() {*

*size(480, 120);*

*smooth();*

*font = loadFont("ComicSansMS-48.vlw");*

*textFont(font);*

*}*

*void draw() {*

*background(102);*

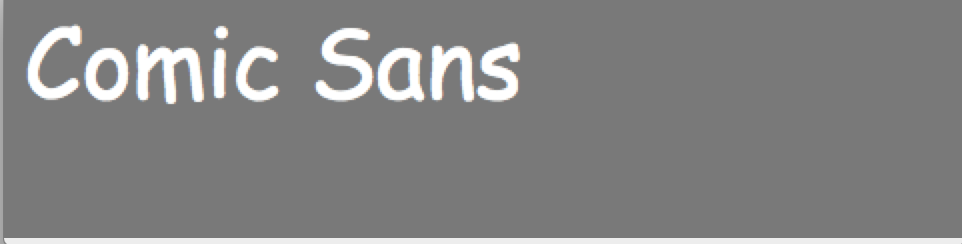
*text("Comic Sans", 10,50);*

*}*

The first parameter to text() is the character(s) to draw to the screen.

(Notice that the characters are enclosed within quotes.) The second and third parameters set the horizontal and vertical location. The location is relative to the baseline of the text.

****

****

10,50

We can use textSize() to change the font size.

Try:

*PFont font;*

*void setup() {*

*size(480, 120);*

*smooth();*

*font = loadFont("ComicSansMS-48.vlw");*

*textFont(font);*

*}*

*void draw() {*

*background(102);*

*textSize(24);*

*text("Comic Sans", 10,50);*

*}*

You can also set text to draw inside a box by adding fourth and fifth parameters that specify the width and height of the box:

Try:

*PFont font;*

*void setup() {*

*size(480, 400);*

*smooth();*

*font = loadFont("ComicSansMS-48.vlw");*

*textFont(font);*

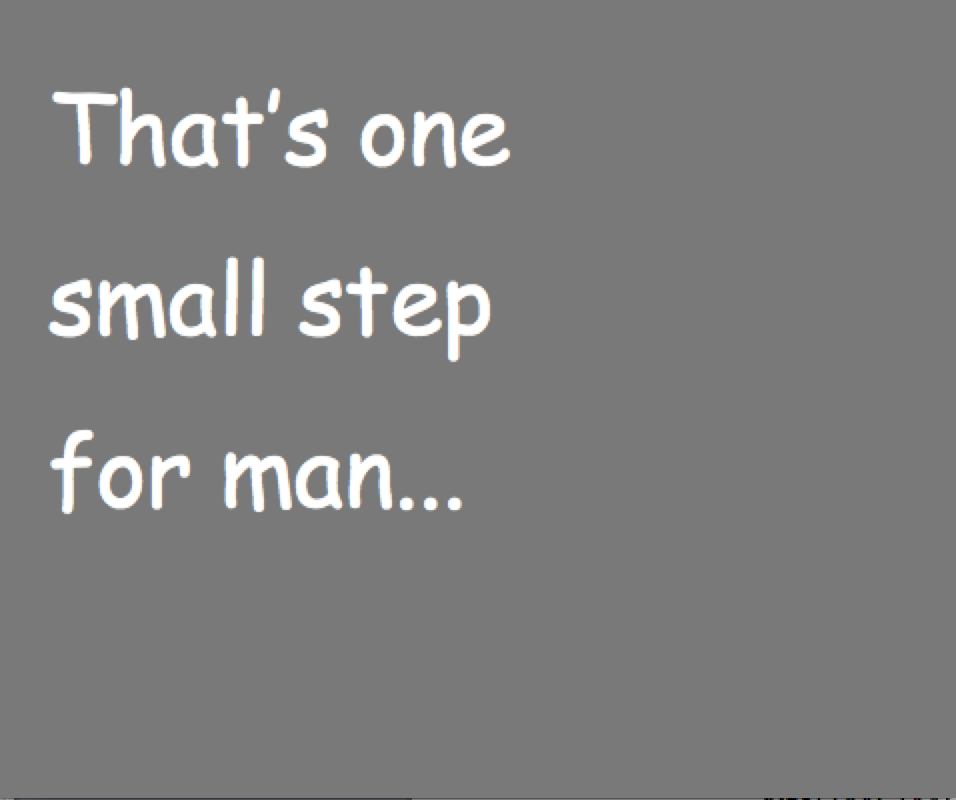
*}*

*void draw() {*

*background(102);*

*text("That’s one small step for man...", 26, 30, 240, 250);*

*}*

****

In the previous example, the words inside the text() function start to make the code difficult to read. We can store these words in a variable to make the code more modular. The String data type is used to store text data.

*PFont font;*

*String text = "That’s one small step for man...";*

*void setup() {*

*size(480, 400);*

*smooth();*

*font = loadFont("ComicSansMS-48.vlw");*

*textFont(font);*

*}*

*void draw() {*

*background(102);*

*text(text, 26, 30, 240, 250);*

*}*

**Task 18: Understanding the frame rate**

To create smooth motion, Processing tries to run the code inside draw() at 60 frames each second. To confirm the frame rate, run this program and watch the values print to the Console. The frameRate variable keeps track of the program’s speed.

Try:

*void draw() {*

*println(frameRate);*

*}*

The frameRate() function changes the speed at which the program runs.

To see the result, uncomment different versions of frameRate() in this example:

*void setup() {*

*frameRate(30); // Thirty frames each second*

*//frameRate(12); // Twelve frames each second*

*//frameRate(2); // Two frames each second*

*//frameRate(0.5); // One frame every two seconds*

*}*

*void draw() {*

*println(frameRate);*

*}*

NOTE: Processing tries to run the code at 60 frames each second, but if it takes longer than 1/60th of a second to run the draw() method, then the frame rate will decrease. The frameRate() function specifies only the maximum frame rate, and the actual frame rate for any program depends on the computer that is running the code.

**Task 19: Move a Shape**

To create fluid motion, we use a data type called float.

Float variable stores numbers with decimal places.

When using ints, the slowest we can move each frame is one pixel at a time (1, 2, 3, 4, . . .), but with floats, you can move as slowly as you want (1.01, 1.01, 1.02, 1.03, . . .).

Try:

*int radius = 40;*

*float x = -radius;*

*float speed = 0.5;*

*void setup() {*

*size(240, 120);*

*smooth();*

*ellipseMode(RADIUS);*

*}*

*void draw() {*

*background(0);*

*x += speed; // Increase the value of x*

*println(x);*

*arc(x, 60, radius, radius, 0.52, 5.76);*

*}*

The shape will moves off the right of the screen when the value of the x variable is greater than the width of the window. The value of x continues to increase, but the shape is no longer visible.

**Task 20: Wrap the moving Shape**

This example shows how to move the shape back to the left edge of the screen after it disappears off the right.

Try:

*int radius = 40;*

*float x = -radius;*

*float speed = 0.5;*

*void setup() {*

*size(240, 120);*

*smooth();*

*ellipseMode(RADIUS);*

*}*

*void draw() {*

*background(0);*

*x += speed; // Increase the value of x*

*if (x > width+radius) { // If the shape is off screen,*

*x = -radius; // move to the left edge*

*}*

*println(x);*

*arc(x, 60, radius, radius, 0.52, 5.76);*

*}*

**Task 21: Bounce off the wall**

This example shows how to bounce back when it hits the left and right edge.

*int radius = 40;*

*float x = 110;*

*float speed = 0.5;*

*int direction = 1; //to keep track of moving direction 1: move to right, -1:move to left*

*void setup() {*

*size(240, 120);*

*smooth();*

*ellipseMode(RADIUS);*

*}*

*void draw() {*

*background(0);*

*x += speed \* direction;*

*if ((x > width-radius) || (x < radius)) {//hit the right or left edge*

*direction = -direction; // Flip direction*

*}*

*if (direction == 1) {*

*arc(x, 60, radius, radius, 0.52, 5.76); // Face right*

*} else {*

*arc(x, 60, radius, radius, 3.67, 8.9); // Face left*

*}*

*}*

**Task 22: Random**

Unlike the smooth, linear motion common to computer graphics, motion in the physical world is usually idiosyncratic. For instance, think of a leaf floating to the ground, or an ant crawling over rough terrain. We can simulate the unpredictable qualities of the world by generating random numbers.

Try:

*void draw() {*

*float r = random(0, 20);*

*println(r);*

*}*

The random(x,y) function will generate a random number between x and y

**Task 23: Random art**

This example uses the values from random() to change the position of lines on screen. When the mouse is at the left of the screen, the change is small; as it moves to the right, the values from random() increase and the movement becomes more exaggerated. Because the random() function is inside the for loop, a new random value is calculated for each point of every line:

Try:

*void setup() {*

*size(240, 120);*

*smooth();*

*}*

*void draw() {*

*background(204);*

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

*float mx = mouseX / 10;*

*float offsetA = random(-mx, mx);*

*float offsetB = random(-mx, mx);*

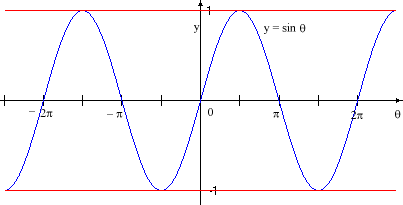
*line(x + offsetA, 20, x - offsetB, 100);*

*}*

*}*

**Task 24: Using Sine function**

Sine function cycles from 0 to 1 to 0 to -1 and 0.



The sinval variable is converted from this range to values from 0 and 255.

This new value is used to set the background color of the window:

float angle = 0.0;

void draw() {

float sinval = sin(angle);//produce value between -1 and 1

float gray = ((sinval+1)\*255.0)/2.0;

println(sinval + " " +gray);

background(gray);

angle += 0.1;

}

This example shows how these values can be converted into movement

*float angle = 0.0;*

*float offset = 60;*

*float scalar = 40;*

*float speed = 0.05;*

*void setup() {*

*size(240, 120);*

*smooth();*

*}*

*void draw() {*

*background(0);*

*float y1 = offset + sin(angle) \* scalar;*

*float y2 = offset + sin(angle + 0.4) \* scalar;*

*float y3 = offset + sin(angle + 0.8) \* scalar;*

*ellipse( 80, y1, 40, 40);*

*ellipse(120, y2, 40, 40);*

*ellipse(160, y3, 40, 40);*

*angle += speed;*

*}*

**Task 25: Circular motion**

Use sin() and cos() together to produce circular motion.

The cos() values provide the x-coordinates, and the sin() values the y-coordinates. Both are multiplied by a variable named scalar to change the radius of the movement and summed with an offset value to set the center of the circular motion:

Try:

*float angle = 0.0;*

*float offset = 60;*

*float scalar = 30;*

*float speed = 0.05;*

*void setup() {*

*size(120, 120);*

*smooth();*

*}*

*void draw() {*

*float x = offset + cos(angle) \* scalar;*

*float y = offset + sin(angle) \* scalar;*

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

*angle += speed;*

*}*

**Task 26: Spiral motion**

This example shows how to use sin() and cos() to animate a spiral motion.

*float angle = 0.0;*

*float offset = 60;*

*float scalar = 2;*

*float speed = 0.05;*

*void setup() {*

*size(120, 120);*

*fill(0);*

*smooth();*

*}*

*void draw() {*

*float x = offset + cos(angle) \* scalar;*

*float y = offset + sin(angle) \* scalar;*

*ellipse( x, y, 2, 2);*

*angle += speed;*

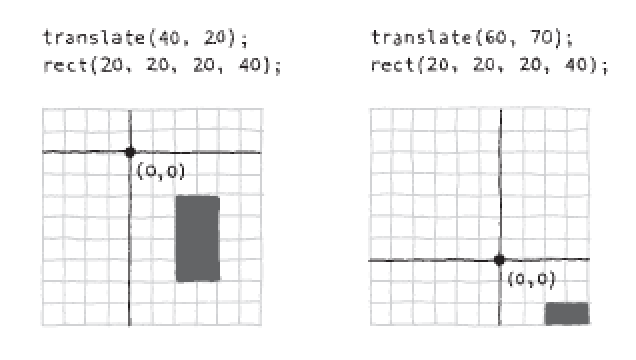
*scalar += speed;*

*}*

**Task 27: Translate**

The default origin(0,0) is at the top left corner.

Translate will move the origin to a new point on the window.

****

Try:

*void setup() {*

*size(120, 120);*

*}*

*void draw() {*

*background(255);*

*translate(50,50);*

*rect(0, 0, 30, 30);*

*rect(10,20,30,30);*

*}*

After a transformation is made, it is applied to all subsequent drawing functions.

The origin is reset back to (0,0) in every draw() call.

Try:

*void setup() {*

*size(120, 120);*

*}*

*void draw() {*

*background(255);*

*translate(mouseX,mouseY);*

*rect(0, 0, 30, 30);*

*}*

**Task 28: Multiple translations**

After a transformation is made, it is applied to all subsequent drawing functions.

Try:

*void setup() {*

*size(120, 120);*

*}*

*void draw() {*

*background(255);*

*translate(mouseX, mouseY);*

*rect(0, 0, 30, 30);*

*translate(35, 10);*

*rect(0, 0, 15, 15);*

*}*

The smaller rectangle was translated the amount of mouseX + 35 and mouseY + 10.

**Task 29: Isolate Translation**

To isolate the effects of a translation so they don’t affect later commands, use the pushMatrix() and popMatrix() functions. When the pushMatrix() function is run, it saves a copy of the current coordinate system and then restores that system after popMatrix():

*Try:*

*void setup() {*

*size(120, 120);*

*}*

*void draw() {*

*background(255);*

*pushMatrix();*

*translate(mouseX, mouseY);*

*rect(0, 0, 30, 30);*

*popMatrix();*

*translate(35, 10);*

*rect(0, 0, 15, 15);*

*}*

**Task 30: Rotation**

The rotate() function rotates the coordinate system.

It has one parameter, which is the angle (in radians) to rotate.

It always rotates relative to the origin (0,0), To spin a shape around its center point, first use translate() to move to the location where you’d like the shape, then call rotate(), and then draw the shape with its center at coordinate (0,0):

Try:

*float angle = 0.0;*

*void setup() {*

*size(120, 120);*

*smooth();*

*}*

*void draw() {*

*background(255);*

*translate(mouseX, mouseY);*

*rotate(angle);*

*rect(-15, -15, 30, 30);*

*angle += 0.1;*

*}*

**Task 31: Combining Translation and Rotation**

We can combine Translation and Rotation to achieve more exciting result:

*float angle = 0.0;*

*void setup() {*

*size(400, 400);*

*smooth();*

*}*

*void draw() {*

*background(255);*

*translate(200, 200);*

*rotate(angle);*

*float x =mouseX/4.0;*

*float y= mouseY/4.0;*

*translate(x, y);*

*rect(-15, -15, 30, 30);*

*angle += 0.1;*

*}*

**Final Task**

Using the SIM building front view drawing that you have completed.

Add in a few cars with animation coming from left to right and right to left.