2D Transformations: Translate, Rotate, Scale

The coordinate system of the Drawing Window can be altered using several transformations. These transformations are extremely useful in further simplifying drawing of shapes. There are three transformations that are possible:

1. **Translate – displaces/moves the origin by (dx, dy)**

   ```
   translate(dx, dy);
   ```

   The command shown above, translates, or moves the origin by \((dx, dy)\). For example:

   ```
   translate(20, 20);
   ```

   The command above translates the origin 20 pixels to the right and 20 pixels to down. See Figure on the right.

   Knowing that you can translate to any point in the Drawing Window, you can assume any shape that you want to draw is anchored at \((0, 0)\). For example, study the sketch below:

   ```
   void setup() {
     size(500, 500);
     background(255);
   } // setup()

   void draw() {
     translate(250, 250);
     fill(90, 119, 255);
     rect(0, 0, 100, 50);
   } // draw()
   ```

   The above sketch translates the origin \((0, 0)\) to the center of the screen (by moving it 250 pixels to the right and 250 pixels down). Then, when the rectangle is drawn with its top-left corner at \((0, 0)\), it will be drawn at \((250, 250)\). Try it.

   To the sketch above, add the following commands at the end:

   ```
   translate(0, 75);
   rect(0, 0, 100, 50);
   ```

   Above, we further move the origin 75 pixels down and then draw a rectangle. Try it. Next, change the parameters of the translate command to:
translate(-100, -125);

Run the sketch to see the effect. Let us revisit the `drawTruck()` function we write earlier. Our definition of `drawTruck()` had the following parameters:

drawTruck(x, y, w, h);

That is, draw a truck anchored at (x, y) of width, w and height, h. We then defined the `drawTruck()` function to use the values of x, y, w, and h to draw the shapes that made up the truck. If we change the drawing of the truck so that it is always anchored to (0, 0), we can eliminate the first two parameters. Then we can draw a truck anywhere by first translating to that point and then drawing the truck. Here is the implementation:

```java
void setup() {
    size(500, 500);
    background(255);
}

float x = 0;
void draw() {
    translate(50, 250);
    fill(255, 46, 46);
    drawTruck(70, 30);
}

void drawTruck(float w, float h) {
    float x = 0, y = 0;
    float bodyW = w * 2 / 3, bodyH = h;
    float hoodW = w - bodyW, hoodH = h/2;
    float wheelD = bodyW * 0.3;
    noStroke();
    rect(x, y - h, bodyW, bodyH);
    rect(x + bodyW, y - hoodH, hoodW, hoodH);
    fill(0);
    triangle(x + bodyW, y - bodyH, x + bodyW, y - hoodH, x + bodyW +
             hoodW/2, y - hoodH);
    fill(0);
    circle(x + bodyW/2, y, wheelD);
    circle(x + bodyW + hoodW/2, y, wheelD);
}
```

Focus on the lines highlighted above. First, the definition of the truck now only has two parameters.
Next, in `drawTruck()`, we set two local variables `x` and `y` (anchor points) to (0, 0). Everything is then drawn relative to (0, 0). Now look at how we are using the commands to draw a truck at (50, 250) in `draw()`. Study the sketch and run it with different values to see how translating the origin to a different point enables you to draw the same shape.

2. **Rotate** – rotates the entire coordinate system by the specified amount.

   \[ \text{rotate}(\text{angle}); \]

   The angle is to be specified in radians. Thus the command:

   \[ \text{rotate}((\pi)/4); \]

   or the command:

   \[ \text{rotate}(\text{radians}(45)); \]

   Both rotate the coordinate system by 45 degrees in the clockwise direction. Like translate, rotate is also a very useful command as it allows us to take any shape and rotate it by any amount. Here is a sketch that you can try:

   ```java
   void setup() {
     size(500, 500);
     background(255);
   } // setup()

   void draw() {
     translate(width/2, height/2);
     rotate((\pi)/4);
     fill(90, 119, 255);
     rectMode(CENTER);
     rect(0, 0, 100, 50);
   } // draw()
   ```

   Run the sketch with different values of the angle. Use the `radians()` function with different values. Here is something fun you can do with rotation:

   ```java
   void setup() {
     size(500, 500);
     background(255);
   } // setup()
   ```
float degrees = 0.0;
void draw() {
  background(255);
  translate(width/2, height/2);
  rotate(radians(degrees));
  degrees = (degrees + 1) % 360;
  //scale(0.5);
  fill(90, 119, 255);
  rectMode(CENTER);
  rect(0, 0, 100, 50);
} // draw()

Try the sketch to see what happens.

3. **Scale – Change the scale of the coordinate system**

scale(value);

For example,

scale(2.0);

The `scale()` command changes the scale of the coordinate system by the amount specified. By default, the scale of the coordinate system is 1. Above, when we specify the scale to be 2.0, everything that is drawn after that will appear twice as big. For example, see the sketch below:

```java
void setup() {
  size(500, 500);
  background(255);
} // setup()

void draw() {
  translate(250, 250);
  scale(2.0);
  fill(90, 119, 255);
  rect(0, 0, 100, 50);
} // draw()
```

Try the sketch shown above both with and without the `scale()` command. Then, change the scale to (0.5) to see the rectangle shrink to half its size when drawn.
4. **Saving and Restoring Drawing Context**

```plaintext
push();   // Save current drawing attributes and transformations
pop();    // Restores previous drawing attributes and transformations
```

We have now seen several different ways to change the drawing settings in our sketches. We can change the stroke attributes, fill attributes, and use transformations. Each of these attributes takes effect after the command to set them is issued. Often, it becomes necessary to revert to the earlier attributes. For example, in the following sketch:

```plaintext
void draw() {
  // Circle 1
  strokeWeight(2);
  fill(125);
  circle(50, 50, 40);

  // Circle 2
  strokeWeight(0.5);
  fill(255);
  translate(50, 50);
  circle(50, 50, 40);

  // Circle 3
  strokeWeight(2);
  fill(125);
  translate(-50, -50);
  circle(100, 50, 40);
} // draw()
```

Above, we are drawing three circles. We set the attributes of circle 1 to stroke weight 2 and fill 125. Next, we change the attributes of circle 2 to stroke weight 0.5 and fill value 255. Also, to draw circle 2, we translated by (50, 50). For circle 3, we are reverting back to the original coordinate system (by translating by (-50, -50), and also setting the stroke weight to 2 and fill to 125 (i.e. same as circle 1).

Using `push()` and `pop()`, we can rewrite the above as shown below:
void draw() {
    // Circle 1
    strokeWeight(2);
    fill(125);
    circle(50, 50, 40);

    // Circle 2
    push();
    strokeWeight(0.5);
    fill(255);
    translate(50, 50);
    circle(50, 50, 40);
    pop();

    // Circle 3
    circle(100, 50, 40);
} // draw()

By using push(), we saved the current context so all the drawing attributes and the
transformations for circle 1 are saved, and then after drawing circle 2, we used pop() to restore all
those settings before we draw circle 3. Try the above sketch both with and without the push() and
pop() commands to see what happens. push() and pop() are very useful when we have large
sketches where we are drawing several shapes and are using many attributes and transformations.

New Commands

translate(dx, dy) Translates the origin (0, 0) by (dx, dy)
rotate(angle) Rotates the coordinate system at (0, 0) by angle radians
scale(n) Scales the coordinate system by a factor of n
push() Save the drawing attributes and transformations of the current context
pop() Restores the drawing attributes and transformations from the previous context