Math 3: Transformation

► Syntax / Function Introduced

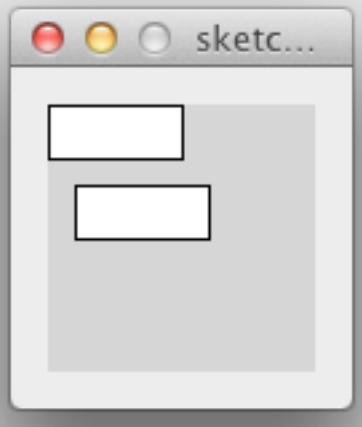
```
translate(), pushMatrix(), popMatrix()
rotate(), scale()
```

This unit introduces coordinate system transformations.

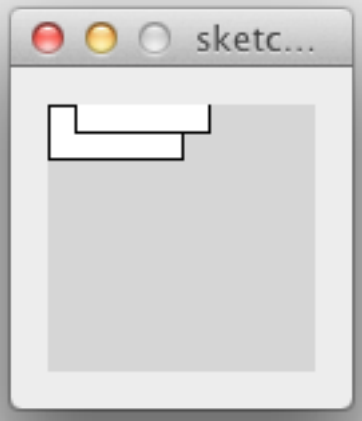
► Translation

The **translate()** function moves the origin from the upper-left corner of the display window to another location, such that you can draw primitives at different locations of the window with same input coordinates.

```
translate(x, y);
```

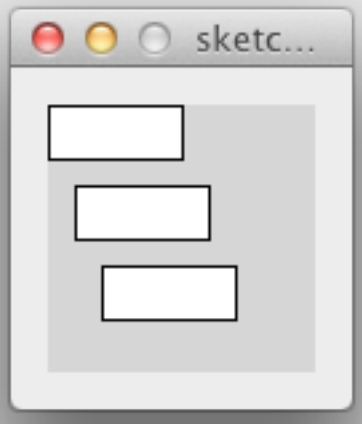



```
rect(0, 0, 50, 20);
// transform origin to 10 pixels right
and 30 pixels down translate(10, 30);
rect(0, 0, 50, 20);
```




```
rect(0, 0, 50, 20);
// transform origin to 10 pixels right
and 10 pixels up
translate(10, -10);
rect(0, 0, 50, 20);
```

Note that **translate()** function is additive, which means if **translate(10, 20)** is run twice, the origin will move to 20 pixels right and 40 pixels down.

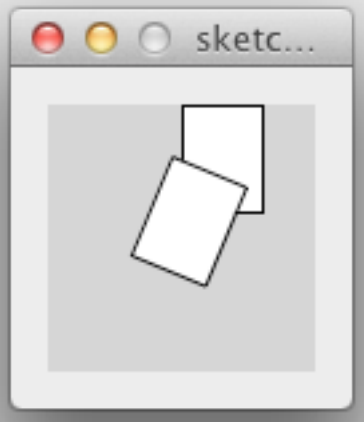



```
rect(0, 0, 50, 20);
translate(10, 30);
rect(0, 0, 50, 20);
```

► Rotation

The **rotate()** function rotates the coordinate system so shapes can be drawn to the screen at an angle. Note that the input angles of rotate()function are in radians.

```
rotate(angle);
```



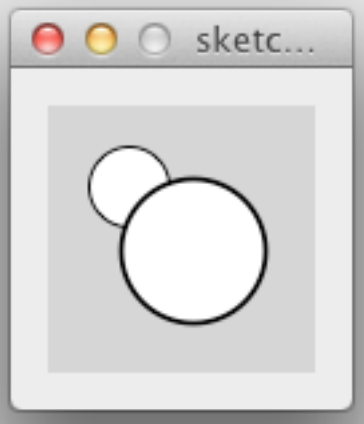

```
smooth();
rect(50, 0, 30, 40);
rotate(PI/8);
rect(50, 0, 30, 40);
```


```
smooth();
rect(50, 0, 40, 20);
rotate(PI/16);
rect(50, 0, 40, 20);
rotate(PI/8);
rect(50, 0, 40, 20);
rotate(PI/4);
rect(50, 0, 40, 20);
```

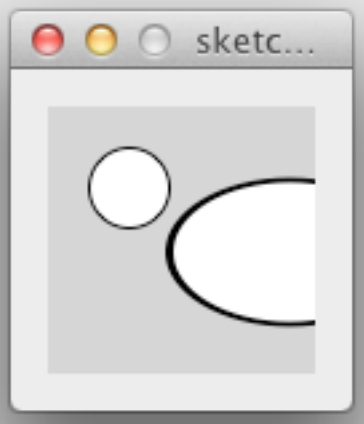
► Scaling

The **scale()** function magnifies the coordinate system so the shapes are drawn larger or smaller. There are 2 versions of the **scale()** function.

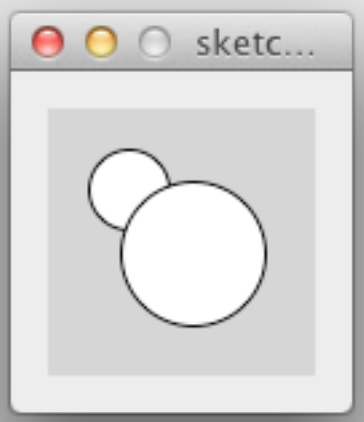
```
scale(size);
scale(xsize, ysize);
```



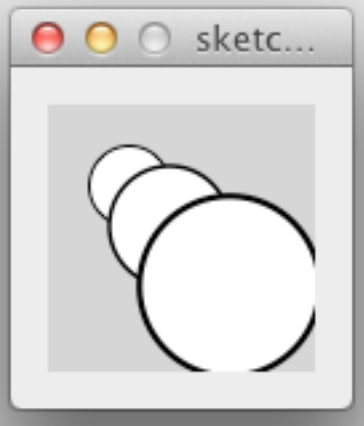

```
smooth();
ellipse(30, 30, 30, 30);
scale(1.8);
ellipse(30, 30, 30, 30);
```




```
smooth();
ellipse(30, 30, 30, 30);
scale(3, 1.8);
ellipse(30, 30, 30, 30);
```



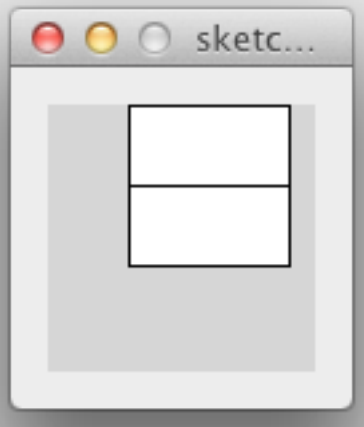

```
smooth();
ellipse(30, 30, 30, 30);
scale(1.8);

// make same stroke weight
strokeWeight(1.0 / 1.8);
ellipse(30, 30, 30, 30);
```



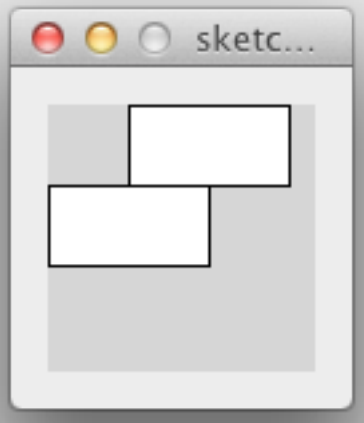

```
smooth();
ellipse(30, 30, 30, 30);
scale(1.5);
ellipse(30, 30, 30, 30);
scale(1.5);
ellipse(30, 30, 30, 30);
```

► Controlling Transformation

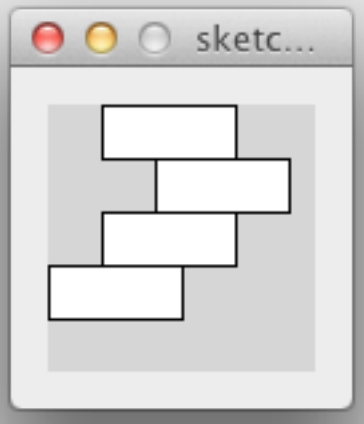
We can use **pushMatrix()** and **popMatrix()** functions to save and restore transformations. In processing, each transformation is represented as a matrix. **pushMatrix()** will store the current matrix into a last-in-first-out stack, and a **popMatrix()** will remove the **LAST matrix** in stack and use it as the current transformation.




```
translate(30, 0);
rect(0, 0, 60, 30);
rect(0, 30, 60, 30);
```



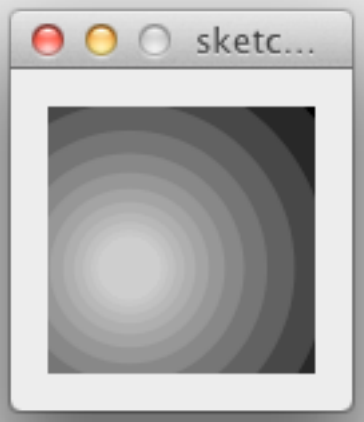

```
// store current transformation
pushMatrix();

translate(30, 0);
rect(0, 0, 60, 30);

// restore transformation
popMatrix();

rect(0, 30, 60, 30);
```



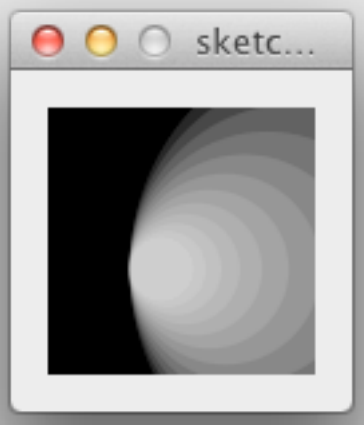

```
// store current transformation A
pushMatrix();

translate(20, 0);
rect(0, 0, 50, 20);

// store current transformation B
pushMatrix();

translate(20, 0);
rect(0, 20, 50, 20);

popMatrix(); // restore transformation B

rect(0, 40, 50, 20);

popMatrix(); // restore transformation A
rect(0, 60, 50, 20);
```

► Combining Transformations


```
background(0);
smooth();
stroke(255, 100);

translate(60, 30);
for (int i=0; i<18; i++) {
    strokeWeight(i);
    rotate(PI/12);
    line(0, 0, 60, 0);
}
```




```
background(0);
smooth();
noStroke();
fill(255, 30);

translate(30, 60);
for (int i=0; i<12; i++) {
  scale(1.2);
  ellipse(0, 0, 20, 20);
}
```




```
background(0);
smooth();
noStroke();
fill(255, 30);

translate(30, 60);
for (int i=0; i<12; i++) {
  scale(1.2);
  ellipse(10, 0, 20, 20);
}
```