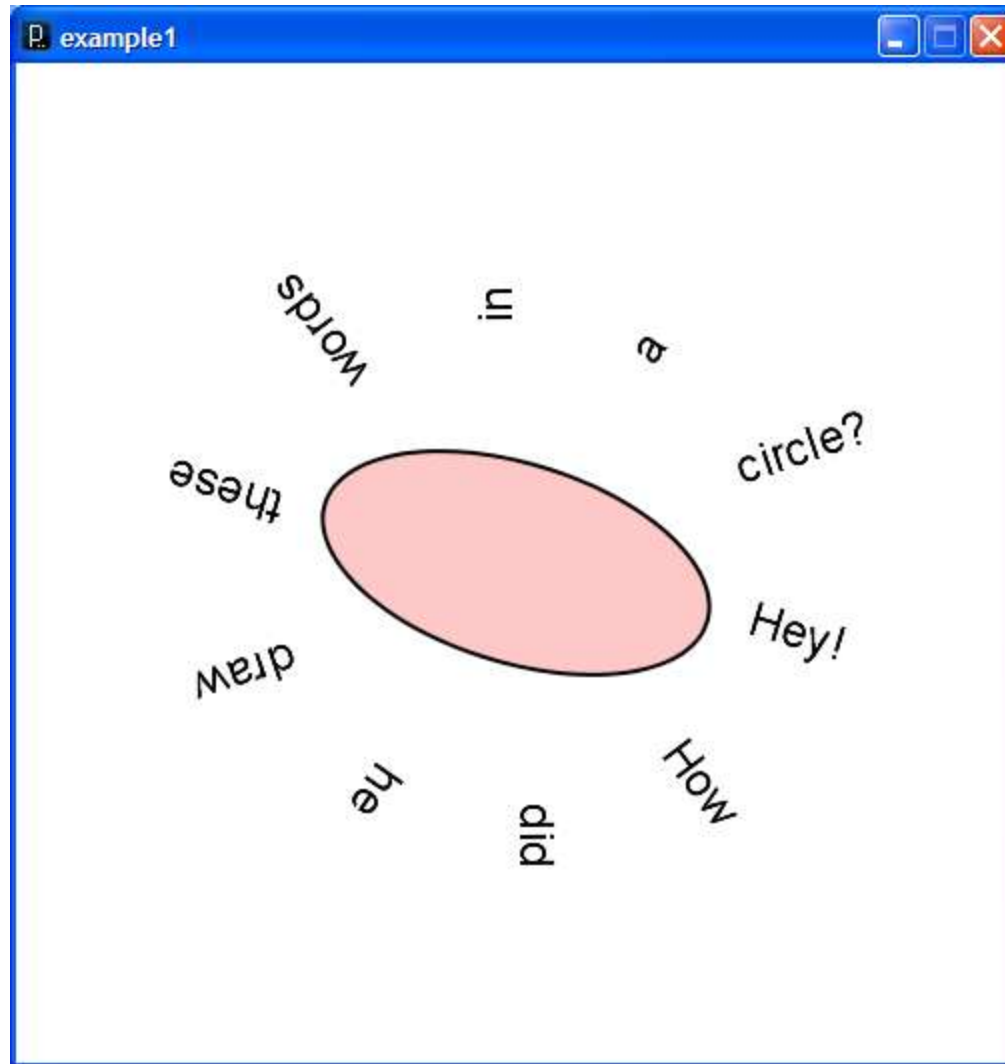


Review

- Cellular Automata
- The Game of Life
 - 2D arrays, 3D arrays
- Review Array Problems
- Challenge

example1.pde

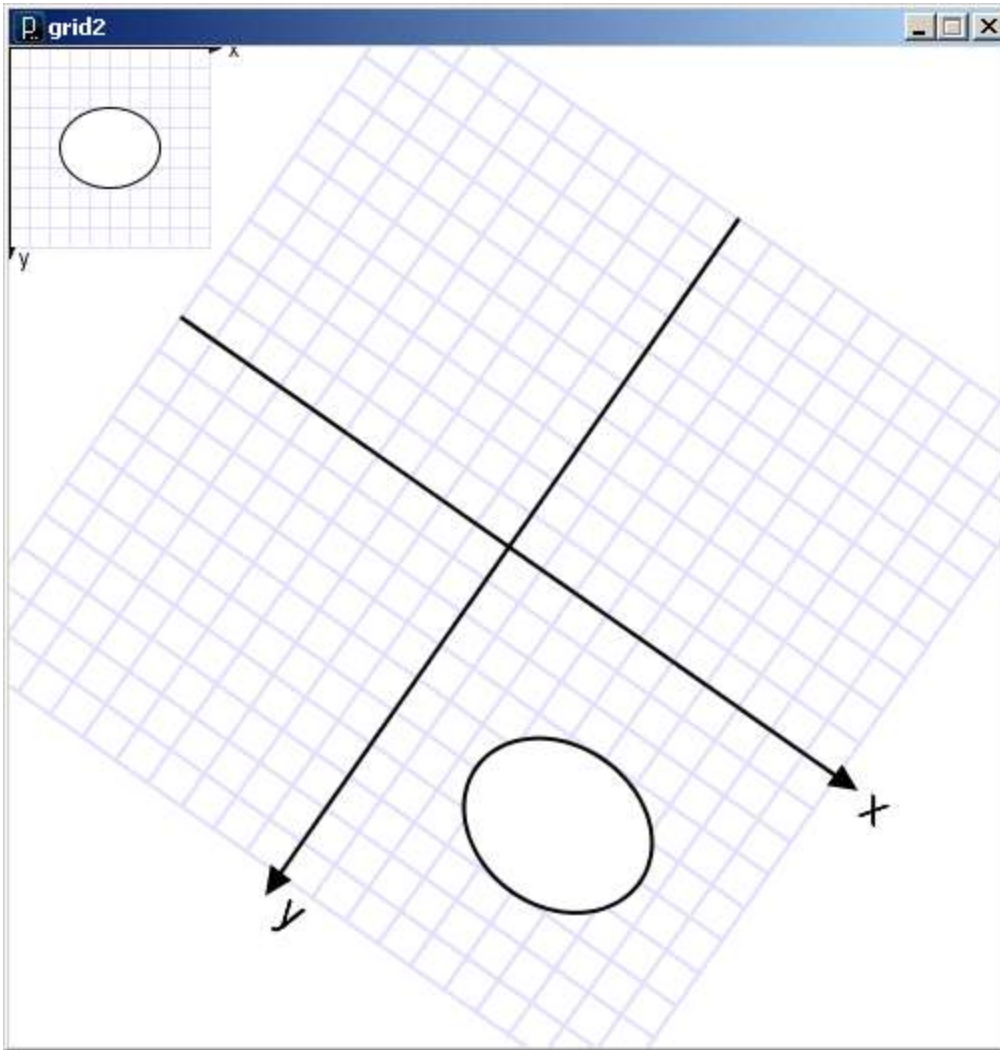


Up until now ...

- *All movement and sizing of graphical objects have been accomplished by **modifying object coordinate values (x, y)** and drawing in the **default coordinate system**.*

There is another option...

- *We can leave coordinate values unchanged, and **modify the coordinate system** in which we draw.*



The commands that draw these two ellipses are identical.

What has changed is the coordinate system in which they are drawn.

Three ways to transform the coordinate system:

1. Translate

- Move axes left, right, up, down ...

2. Scale

- Magnify, zoom in, zoom out ...

3. Rotate

- Tilt clockwise, tilt counter-clockwise ...

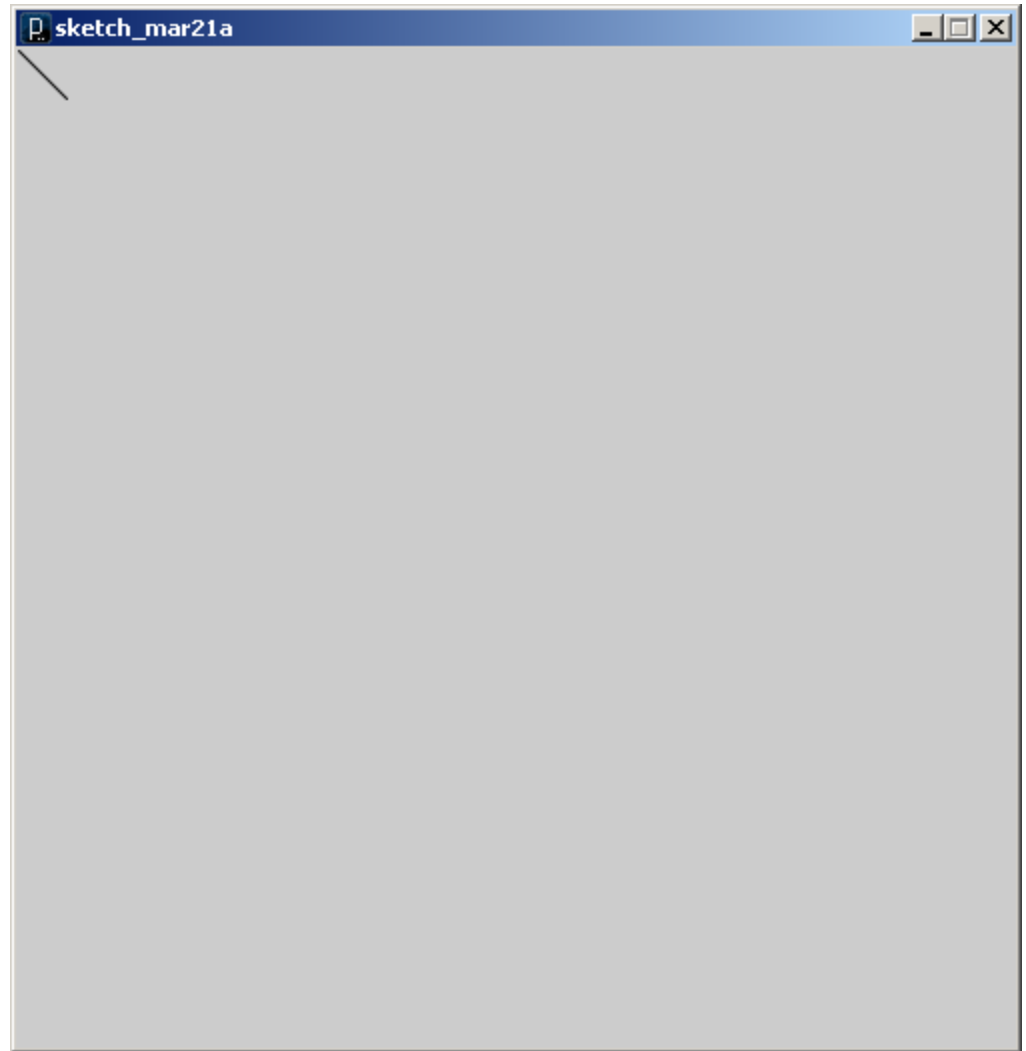
Scale

- All coordinates are multiplied by an x-scale-factor and a y-scale-factor.
- The size of everything is magnified about the origin (0,0)
- Stroke thickness is also scaled.

```
scale( factor );
```

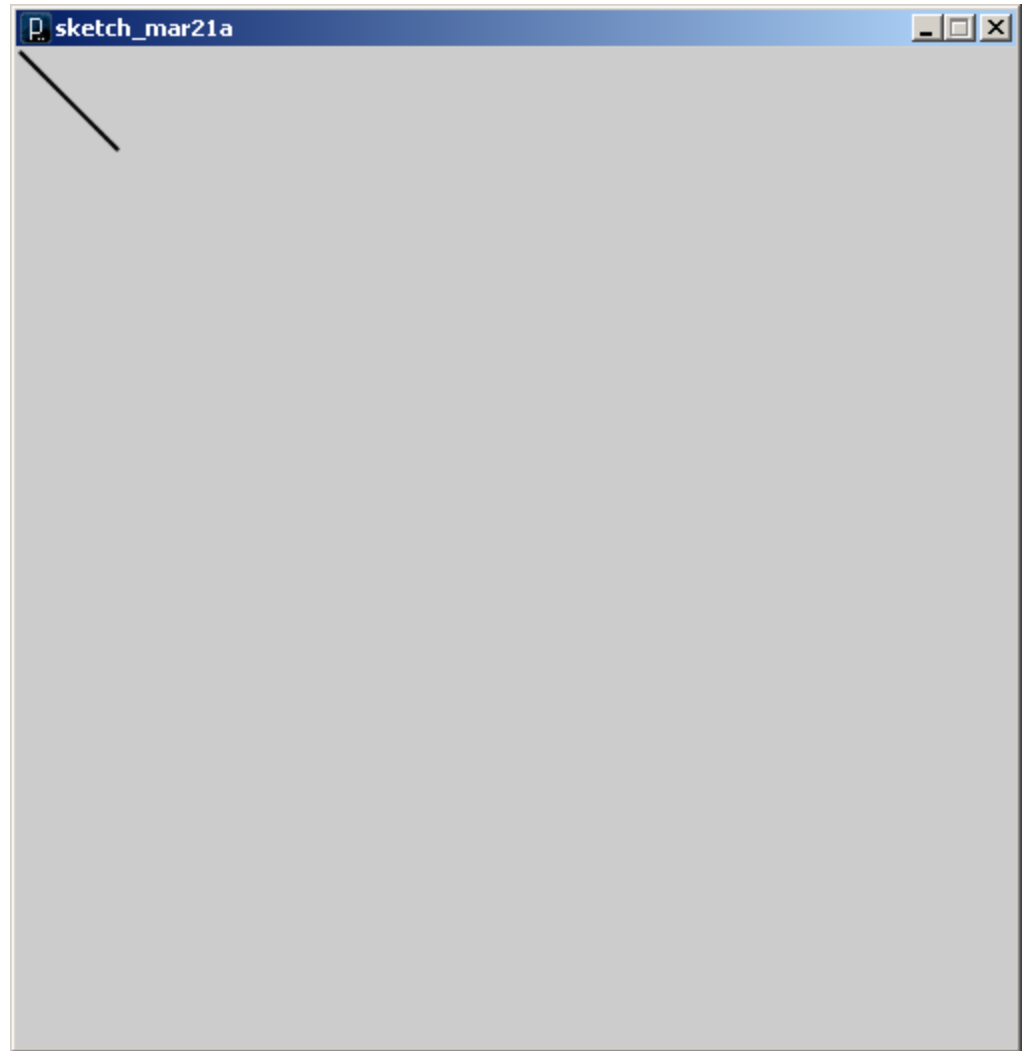
```
scale( x-factor, y-factor );
```

```
void setup() {  
  size(500, 500);  
  smooth();  
  noLoop();  
  
  line(1, 1, 25, 25);  
}
```



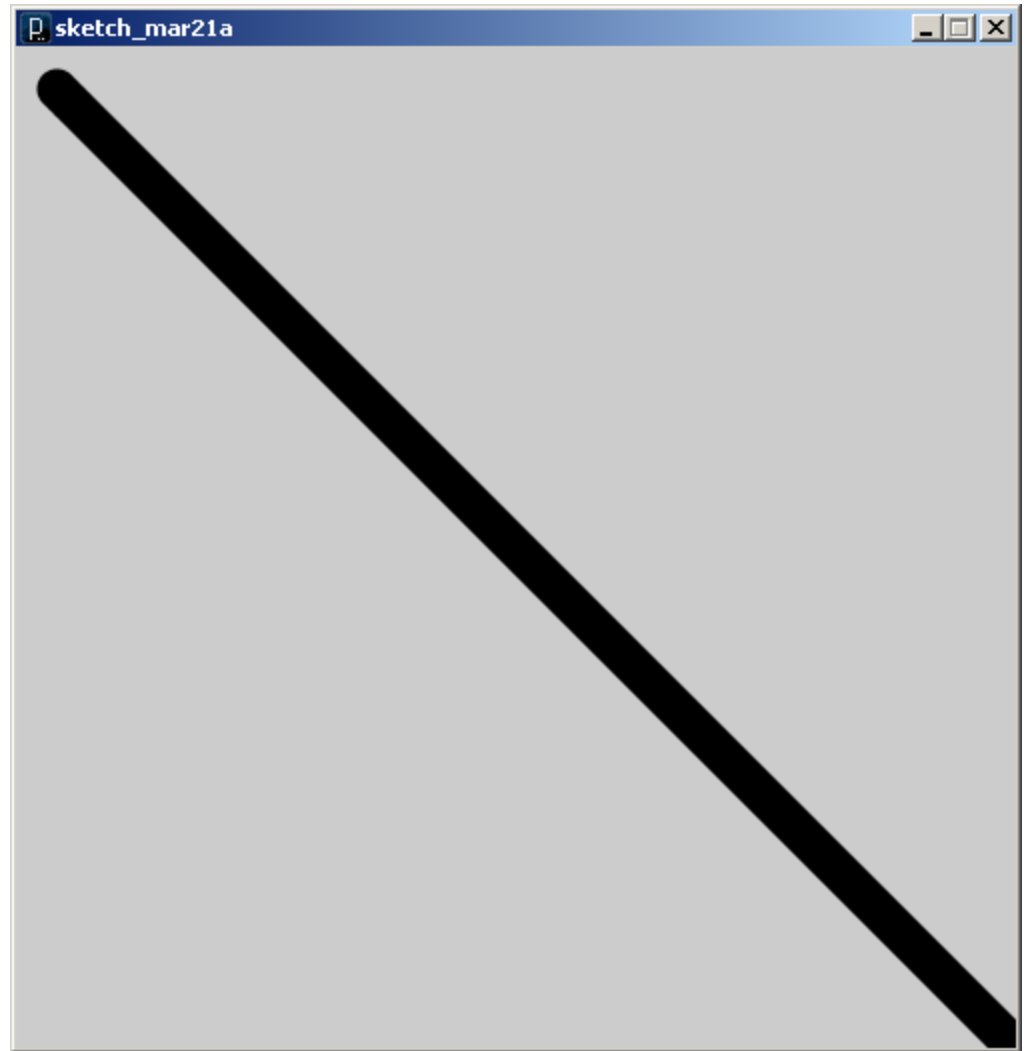
example2.pde

```
void setup() {  
  size(500, 500);  
  smooth();  
  noLoop();  
  
  scale(2,2);  
  line(1, 1, 25, 25);  
}
```



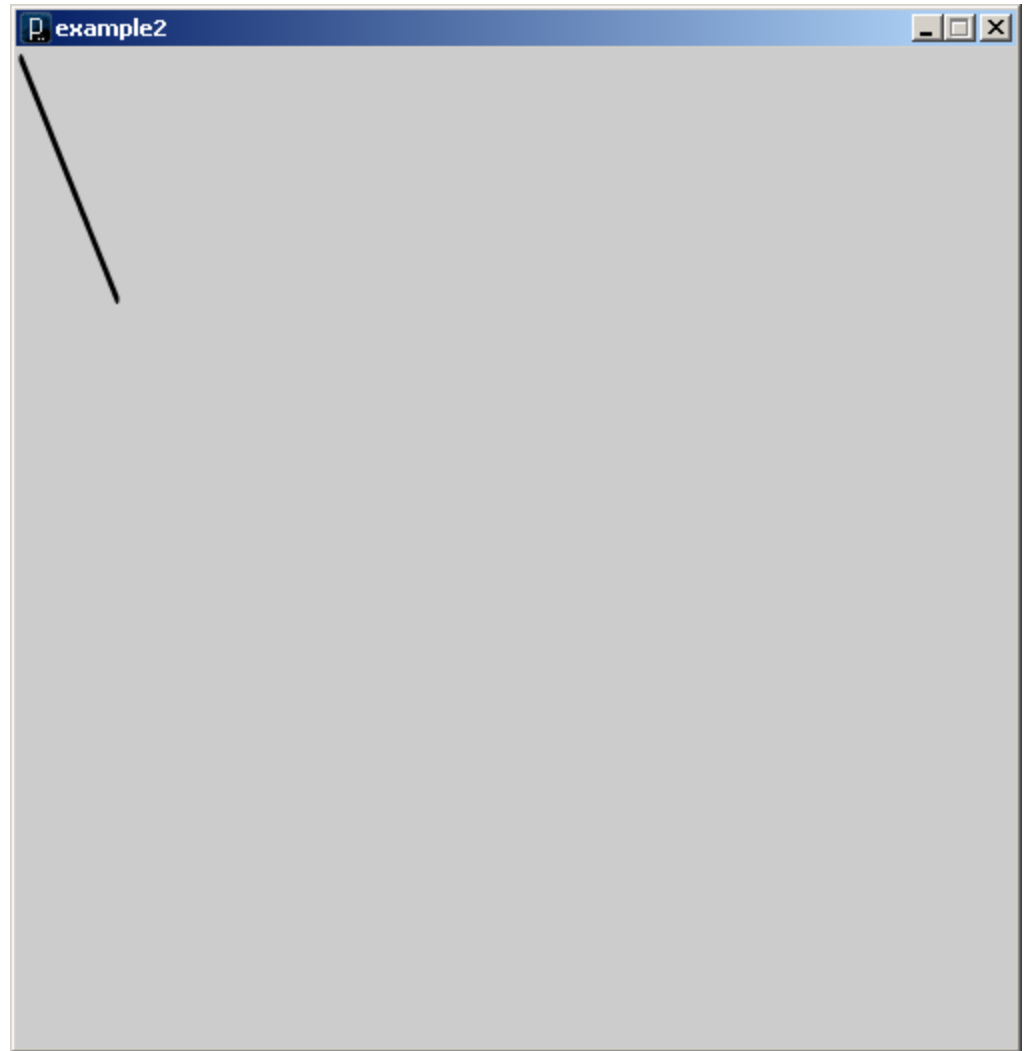
example2.pde


```
void setup() {  
  size(500, 500);  
  smooth();  
  noLoop();  
  
  scale(20,20);  
  line(1, 1, 25, 25);  
}
```



example2.pde

```
void setup() {  
  size(500, 500);  
  smooth();  
  noLoop();  
  
  scale(2, 5);  
  line(1, 1, 25, 25);  
}
```



example2.pde

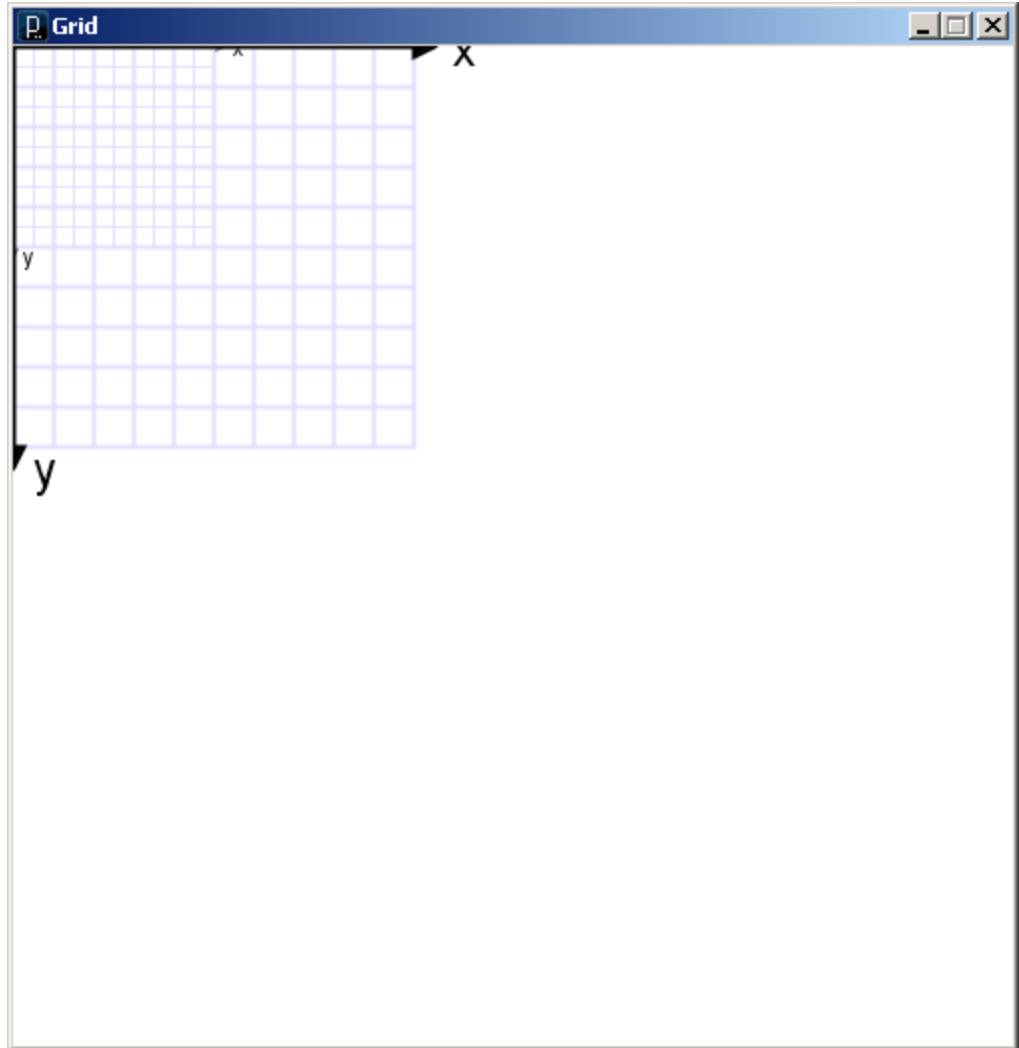
The best way to see what is happening, is to look at a grid drawn in the coordinate system.

```
void grid() {
    grid(-100, 100, 10, -100, 100, 10);
}

void grid(float x1, float x2, float dx,
          float y1, float y2, float dy) {
    // Draw grid
    stroke(225,225,255);
    for (float x=x1; x<=x2; x+=dx) line(x,y1,x,y2);
    for (float y=y1; y<=y2; y+=dy) line(x1,y,x2,y);

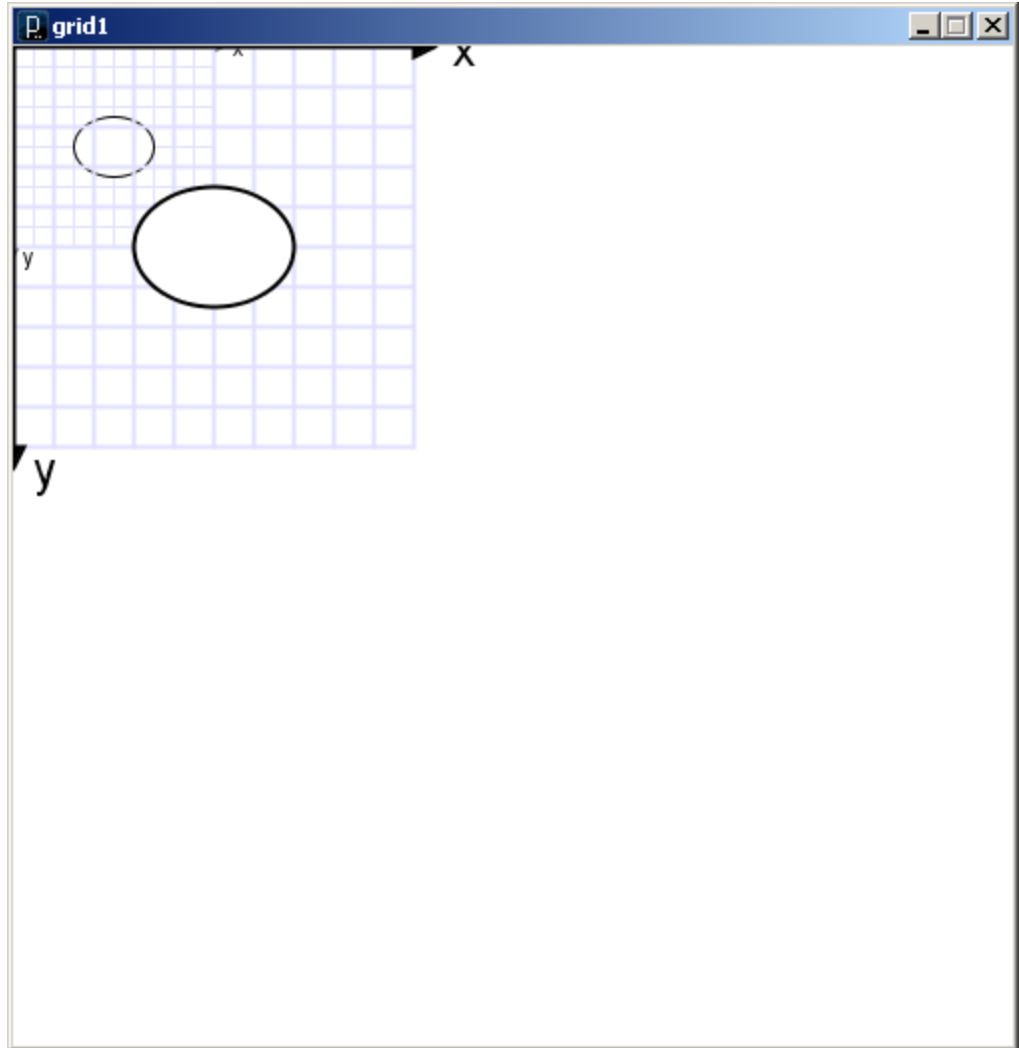
    // Draw axes
    float inc = 0.005*width;
    float inc2 = 2.0*inc;
    stroke(0);
    fill(0);
    line(x1,0,x2,0);
    triangle(x2+inc2,0,x2,inc,x2,-inc);
    text("x",x2+2*inc2,inc2);
    line(0,y1,0,y2);
    triangle(0,y2+inc2,inc,y2,-inc,y2);
    text("y",inc2,y2+2*inc2);
}
```

```
void setup() {  
  size(500, 500);  
  background(255);  
  smooth();  
  noLoop();  
}  
  
void draw() {  
  grid();  
  scale(2,2);  
  grid();  
}
```



grid1.pde

```
void draw() {  
  grid();  
  fill(255);  
  ellipse(50,50,40,30);  
  
  scale(2,2);  
  grid();  
  fill(255);  
  ellipse(50,50,40,30);  
}
```



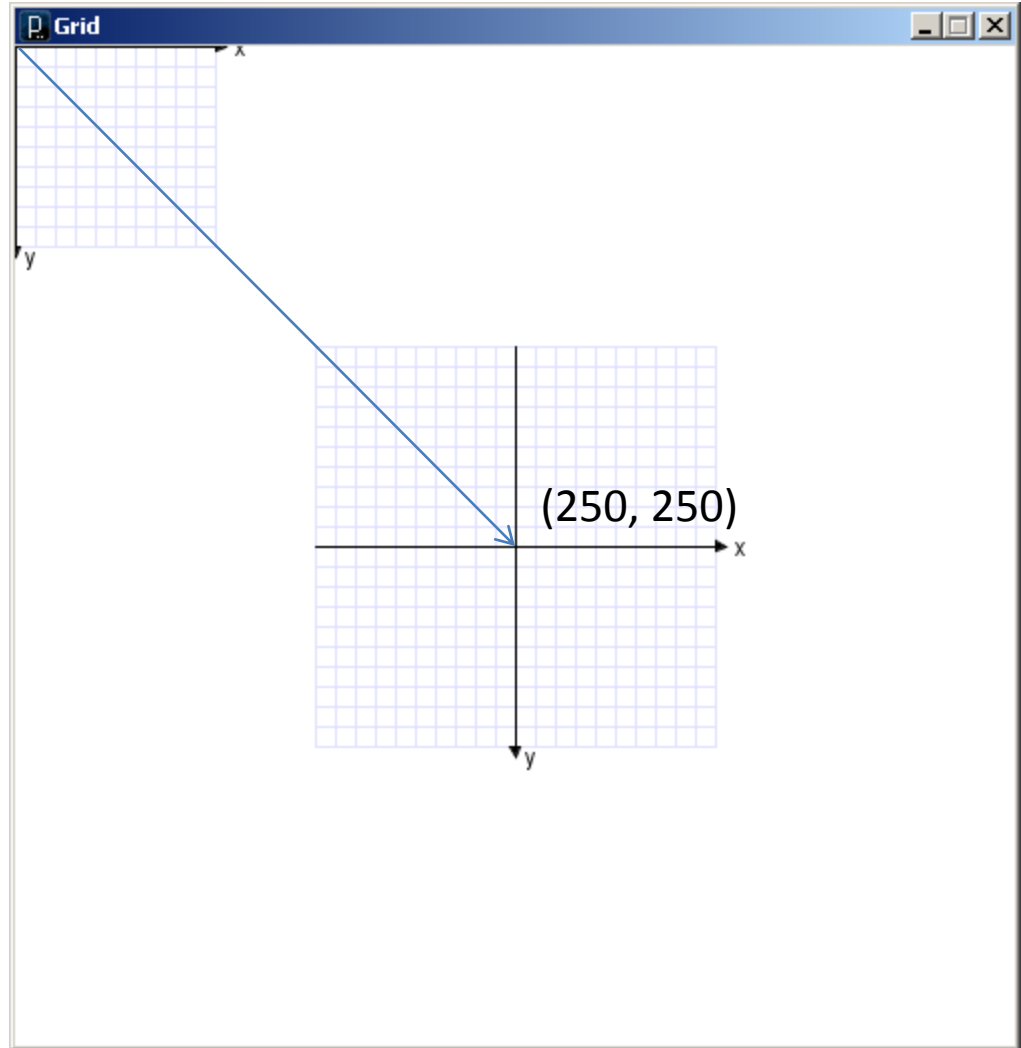
grid1.pde

Translate

- The origin of the coordinate system (0,0) is shifted by the given amount in the x and y directions.

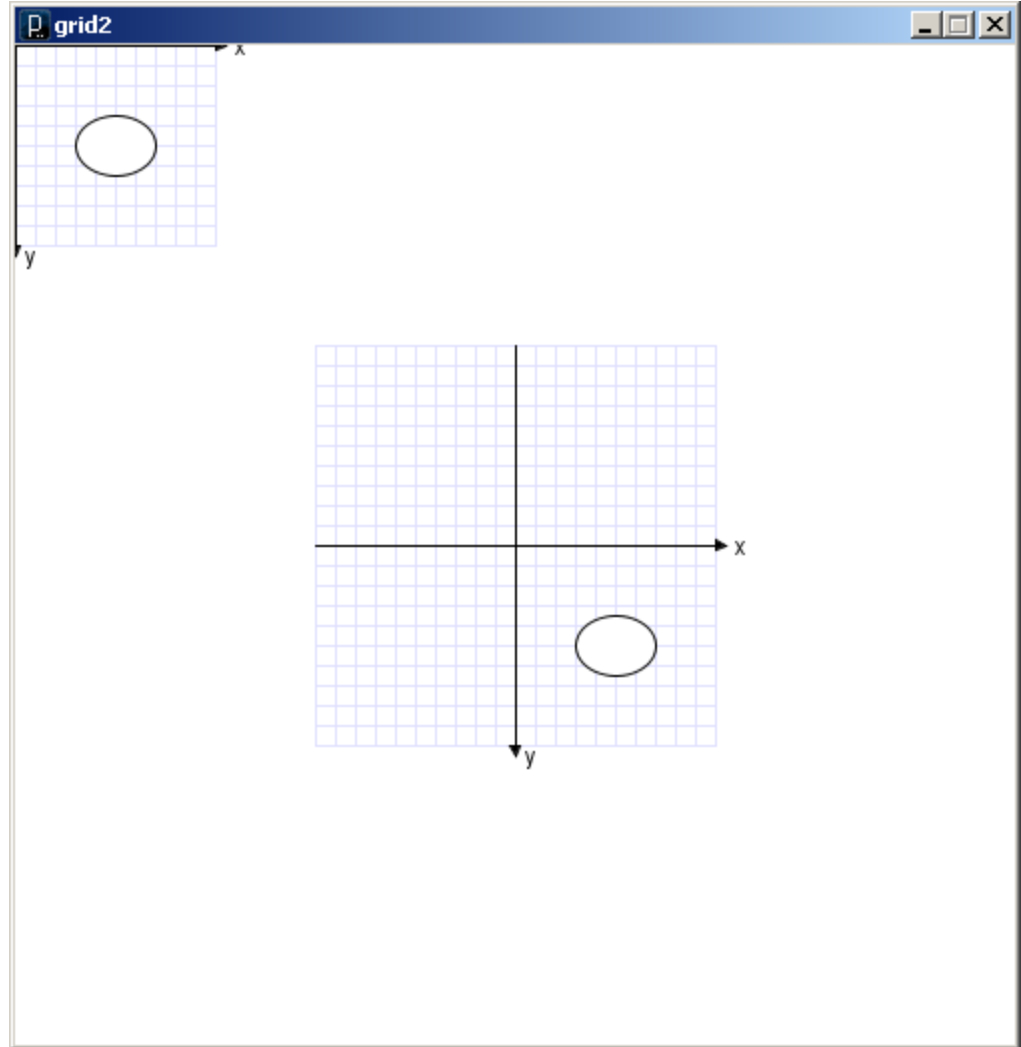
```
translate( x-shift, y-shift);
```

```
void draw() {  
  grid();  
  translate(250,250);  
  grid();  
}
```



grid2.pde

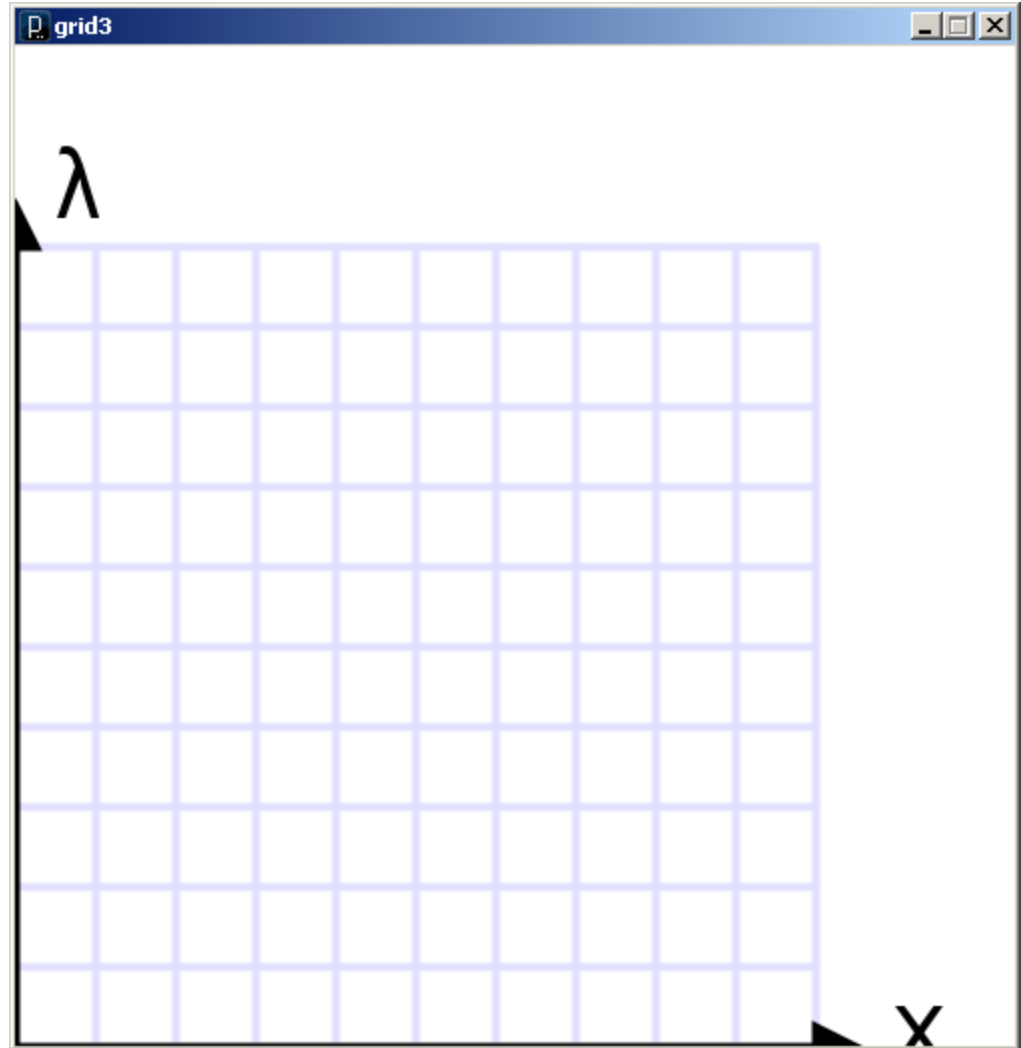
```
void draw() {  
    grid();  
    fill(255);  
    ellipse(50, 50, 40, 30);  
  
    translate(250, 250);  
    grid();  
    fill(255);  
    ellipse(50, 50, 40, 30);  
}
```



Transformations can be combined

- Combine Scale and Translate to create a coordinate system with the y -axis that increases in the upward direction
- Axes can be flipped using negative scale factors
- Order in which transforms are applied matters!

```
void draw() {  
  translate(0,height);  
  scale(4,-4);  
  grid();  
}
```



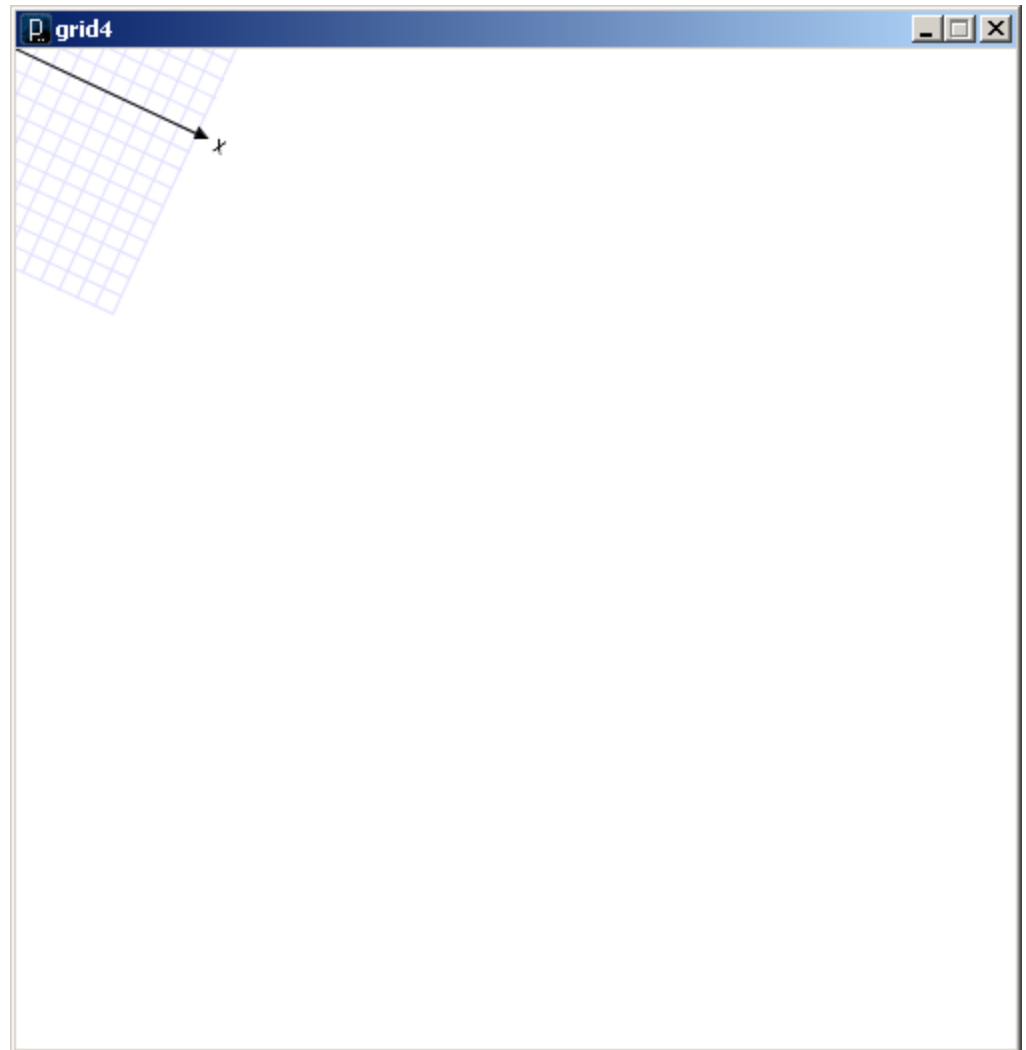
grid3.pde

Rotate

- The coordinate system is rotated around the origin by the given angle (in radians).

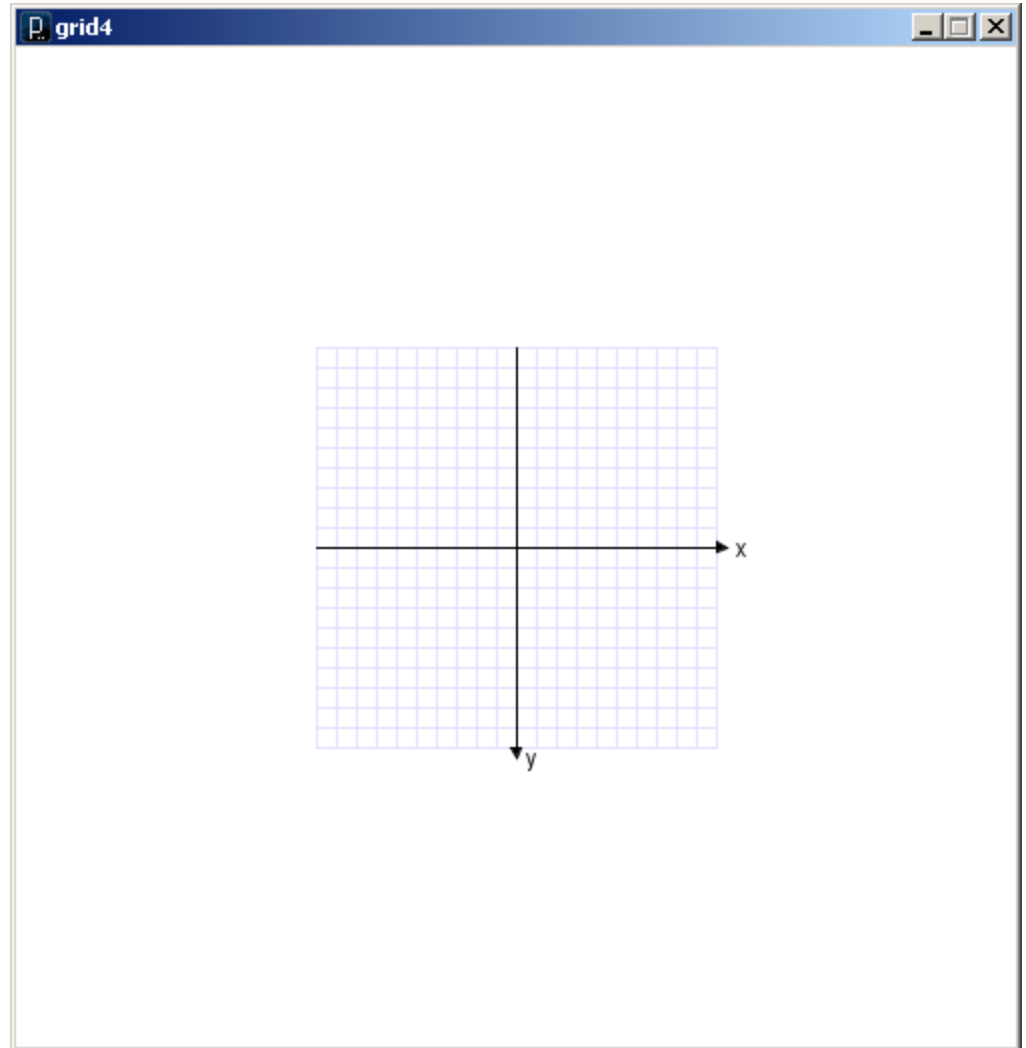
```
rotate( radians );
```

```
void draw() {  
  rotate( 25.0 * (PI/180.0) );  
  grid();  
}
```



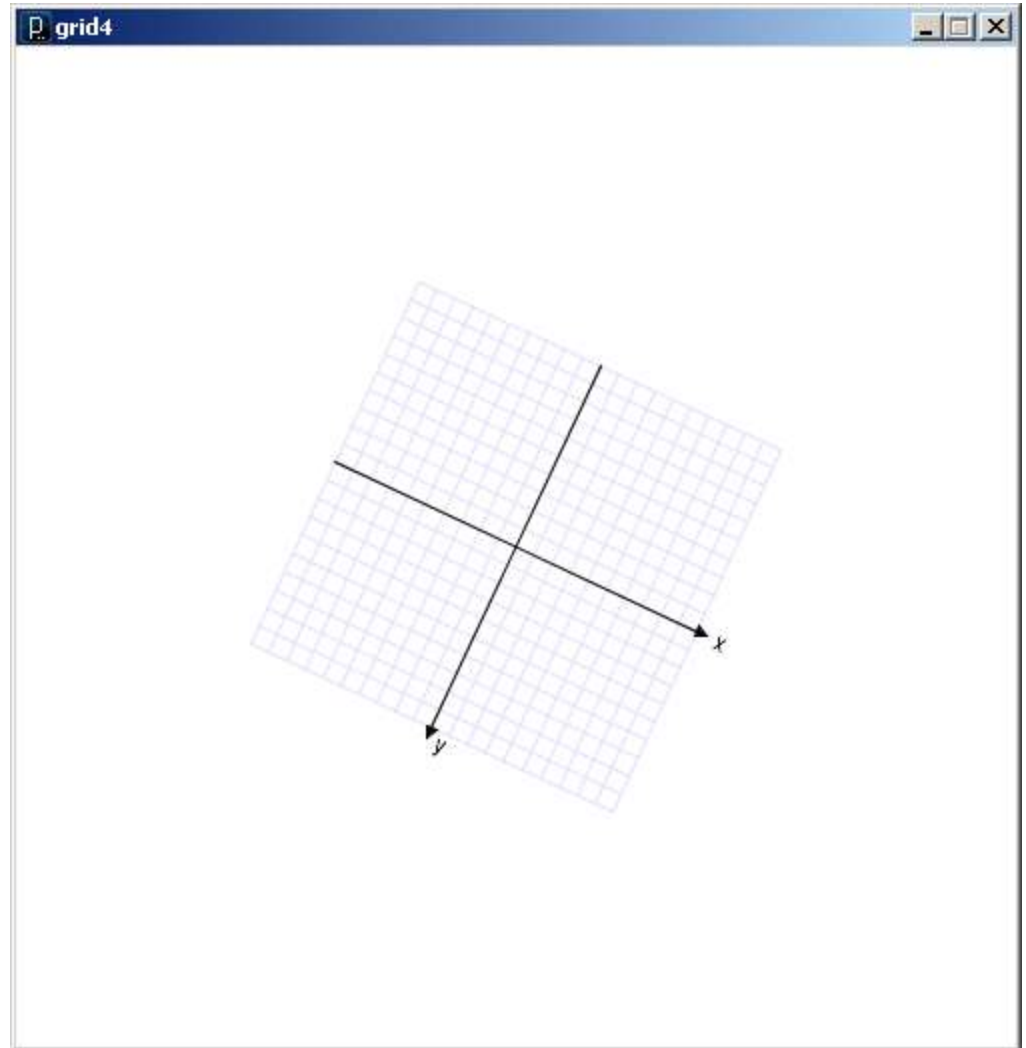
grid4.pde

```
void draw() {  
  translate(250.0, 250.0);  
  //rotate( 25.0 * (PI/180.0) );  
  //scale( 2 );  
  grid();  
}
```



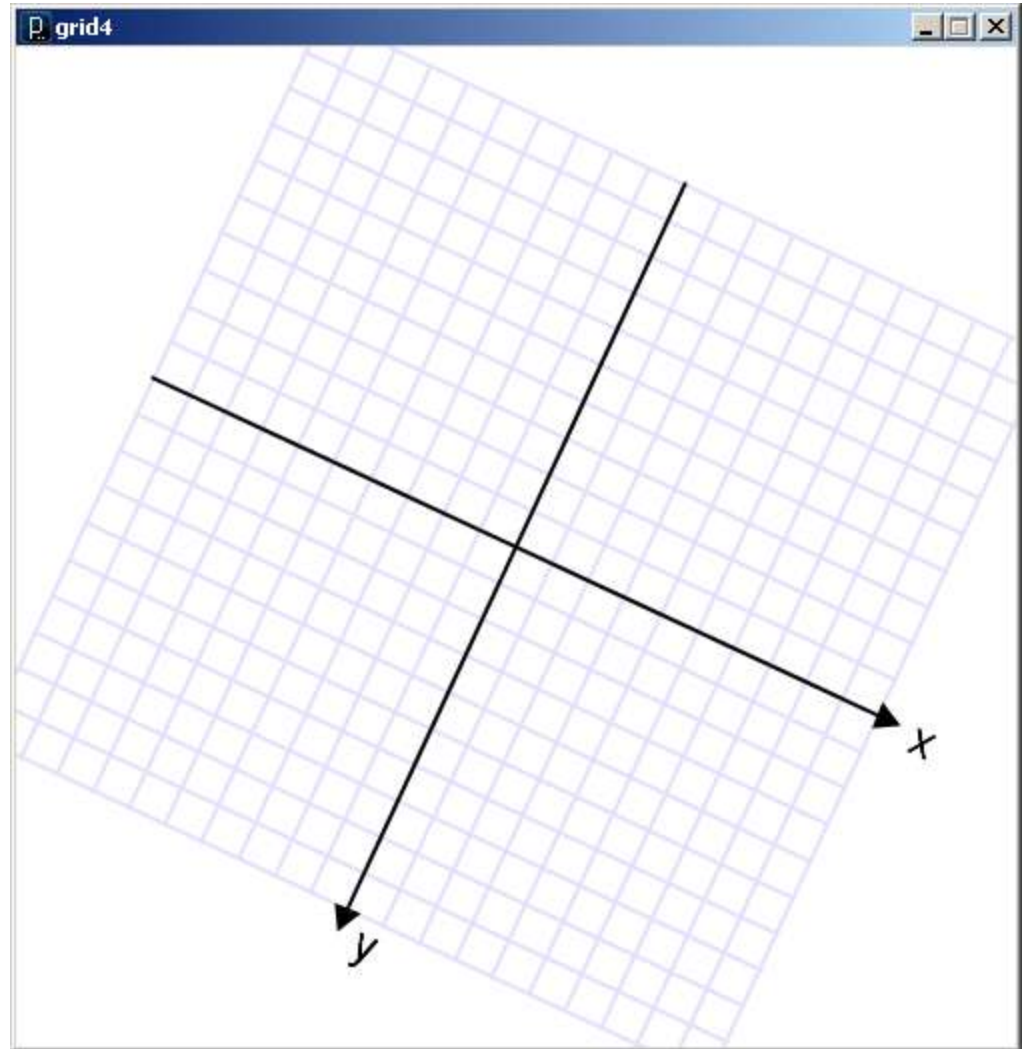
grid4.pde

```
void draw() {  
  translate(250.0, 250.0);  
  rotate( 25.0 * (PI/180.0) );  
  //scale( 2 );  
  grid();  
}
```



grid4.pde

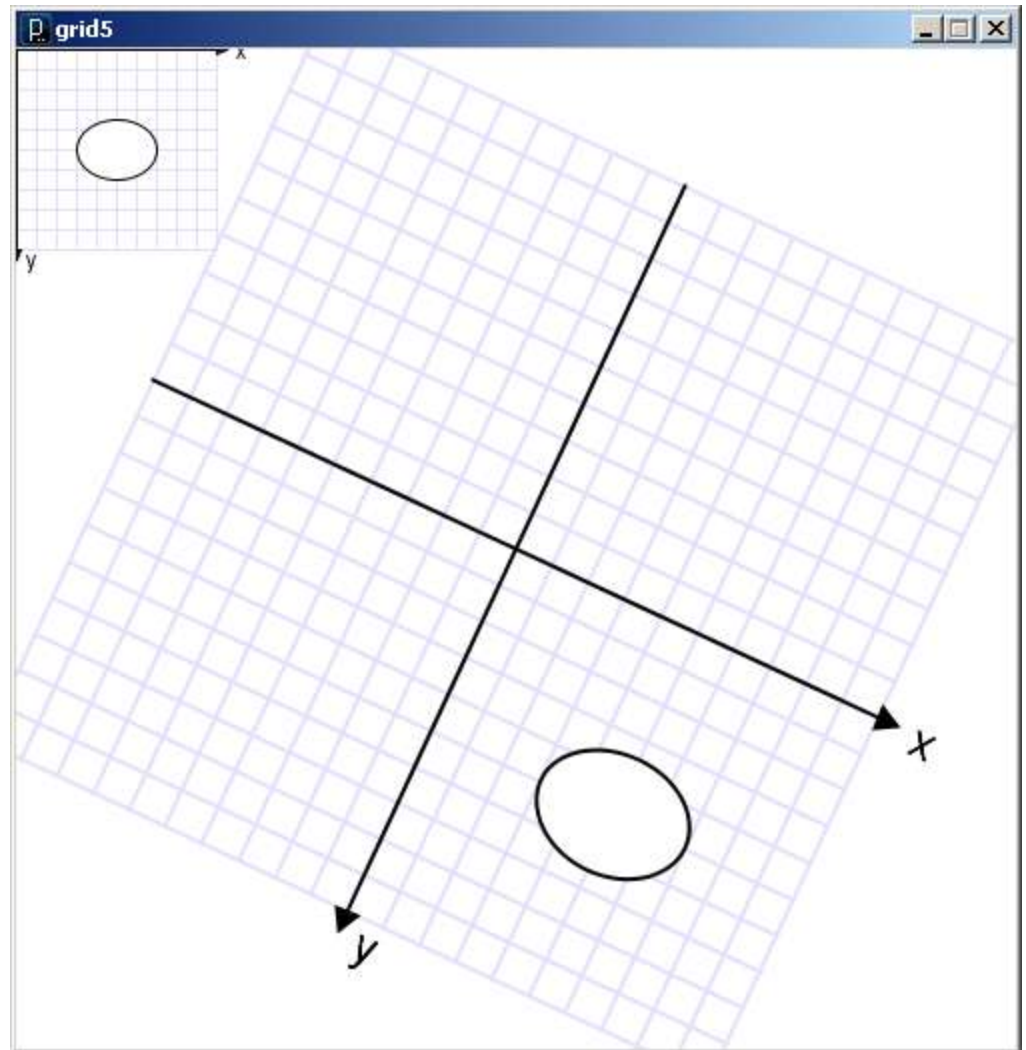
```
void draw() {  
  translate(250.0, 250.0);  
  rotate( 25.0 * (PI/180.0) );  
  scale( 2 );  
  grid();  
}
```



grid4.pde

```
void draw() {  
  grid();  
  fill(255);  
  ellipse(50, 50, 40, 30);  
  
  translate(250.0, 250.0);  
  rotate( 25.0 * (PI/180.0) );  
  scale(2);  
  grid();  
  fill(255);  
  ellipse(50, 50, 40, 30);  
}
```

grid5.pde



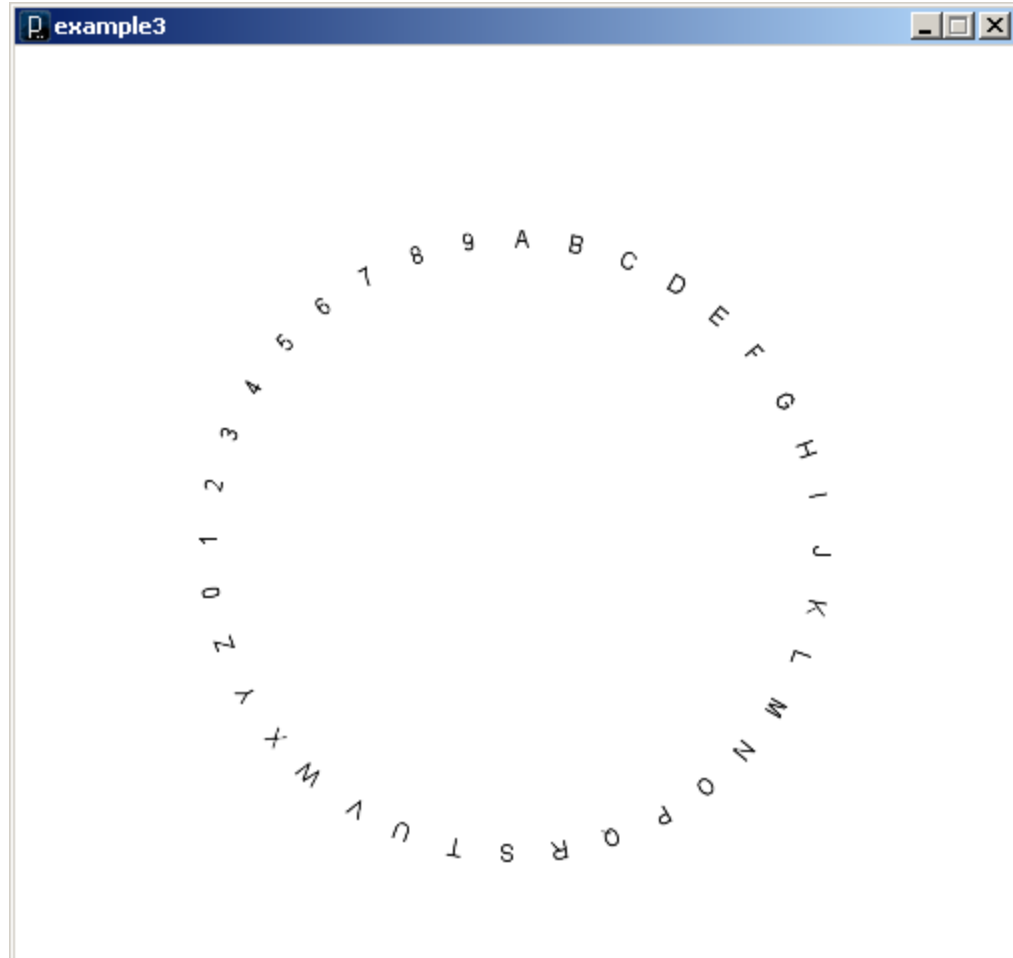
Some things to remember:

1. Transformations are cumulative.
2. All transformations are cancelled each time `draw()` exits.
 - They must be reset each time at the beginning of `draw()` before any drawing.
3. Rotation angles are measured in radians
 - π radians = 180°
 - radians = $(\text{PI}/180.0) * \text{degrees}$
4. Order matters

```
String[] word = new String[]  
{ "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S",  
  "T", "U", "V", "W", "X", "Y", "Z", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};
```

```
void setup() {  
  size(500, 500);  
  smooth();  
  noLoop();  
}
```

```
void draw() {  
  background(255);  
  translate(250,250);  
  
  fill(0);  
  for (int i=0; i<word.length; i++) {  
    text( word[i], 0.0, -150.0 );  
    rotate(10.0 * (PI/180.0));  
  }  
}
```



Each time through the loop an additional 10 degrees is added to the rotation angle.

Total rotation accumulates.

```
String[] word = new String[]  
{ "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T",  
"U", "V", "W", "X", "Y", "Z", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" };
```

```
float start = 0.0;
```

```
void setup() {  
  size(500, 500);  
  smooth();  
}
```

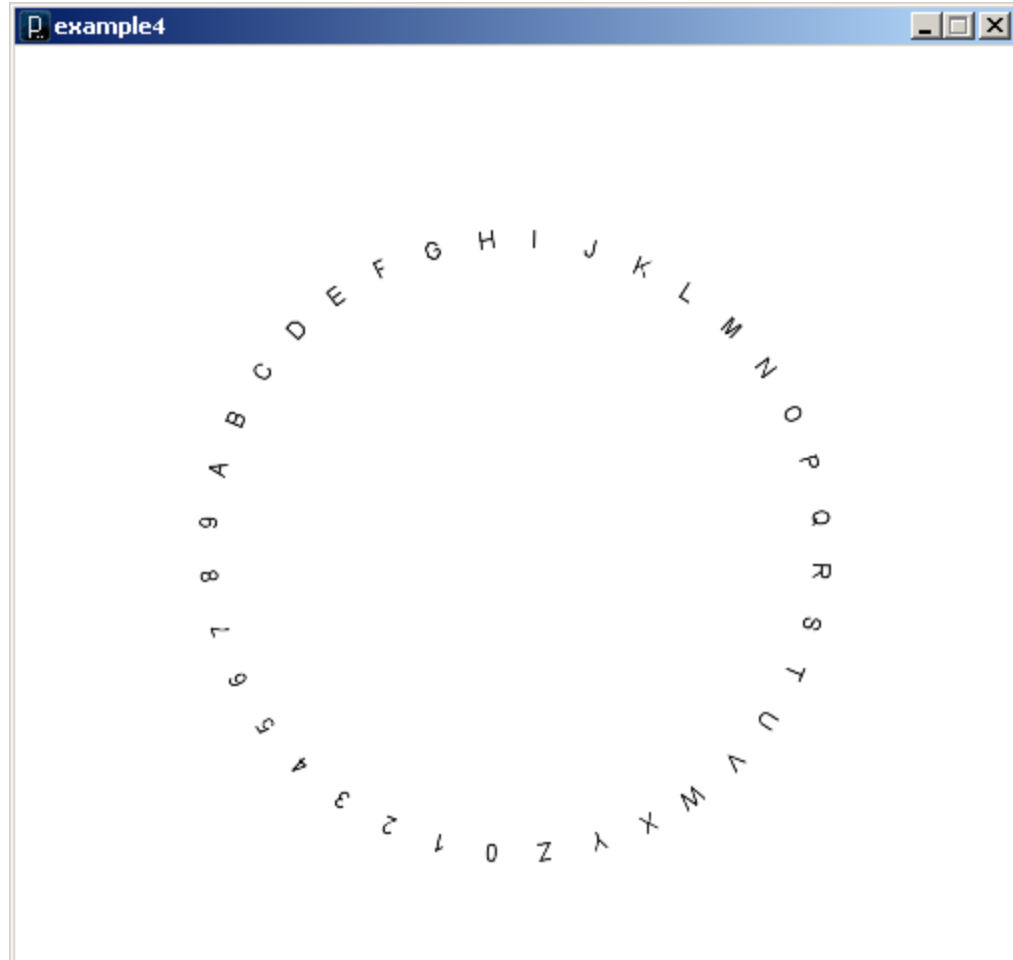
```
void draw() {  
  background(255);  
  translate(250, 250);
```

```
  fill(0);  
  rotate(start);
```

```
  for (int i=0; i<word.length; i++) {  
    text( word[i], 0.0, -150.0 );  
    rotate(10.0 * (PI/180.0));  
  }
```

```
  start += 1.0*(PI/180.0) % TWO_PI;
```

```
}
```



Each time through the loop an initial rotation angle is set, incremented, and saved in a global.

Transformations reset each time draw() is called.

example4.pde