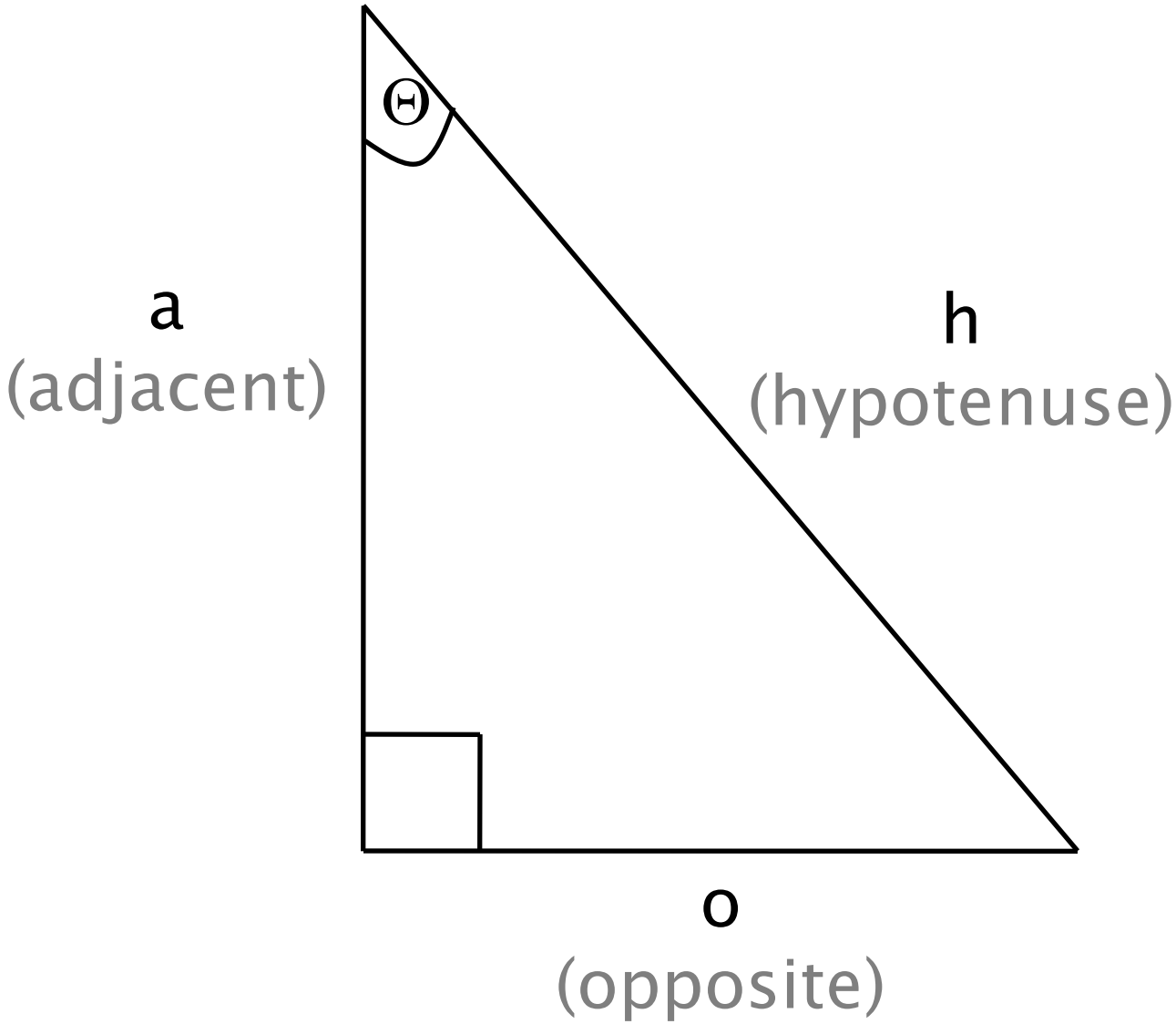
Review

- Lists
- len(), del
- List element access
- List slicing using ‘:’
- List methods – including append()
- Sequence types – List, String, Tuple
- for-in loops
- range() function
- Examples

Basics of Trigonometry



Definition

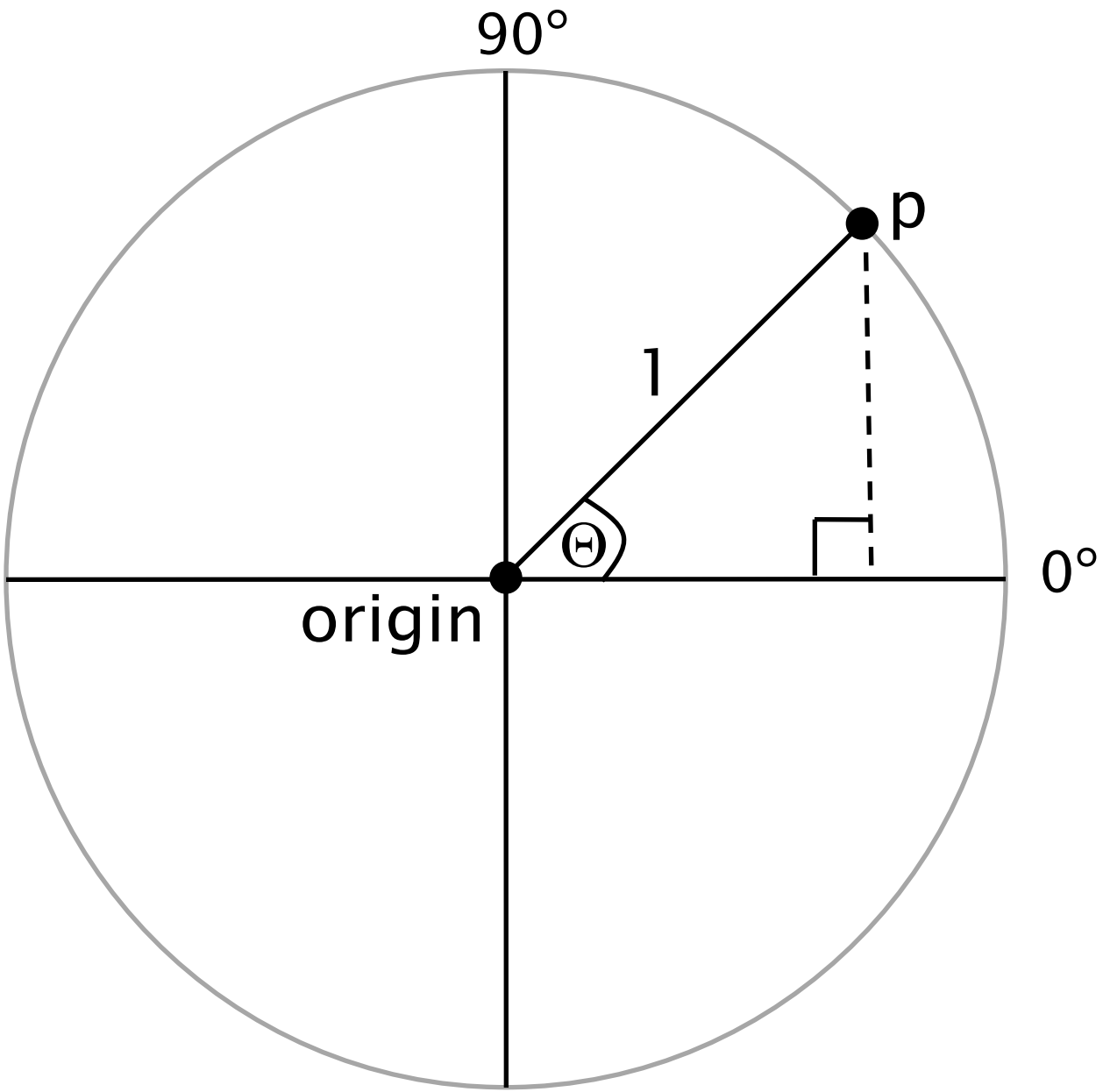
- $\sin(\Theta) = o/h$
- $o = h*\sin(\Theta)$

- $\cos(\Theta) = a/h$
- $a = h*\cos(\Theta)$

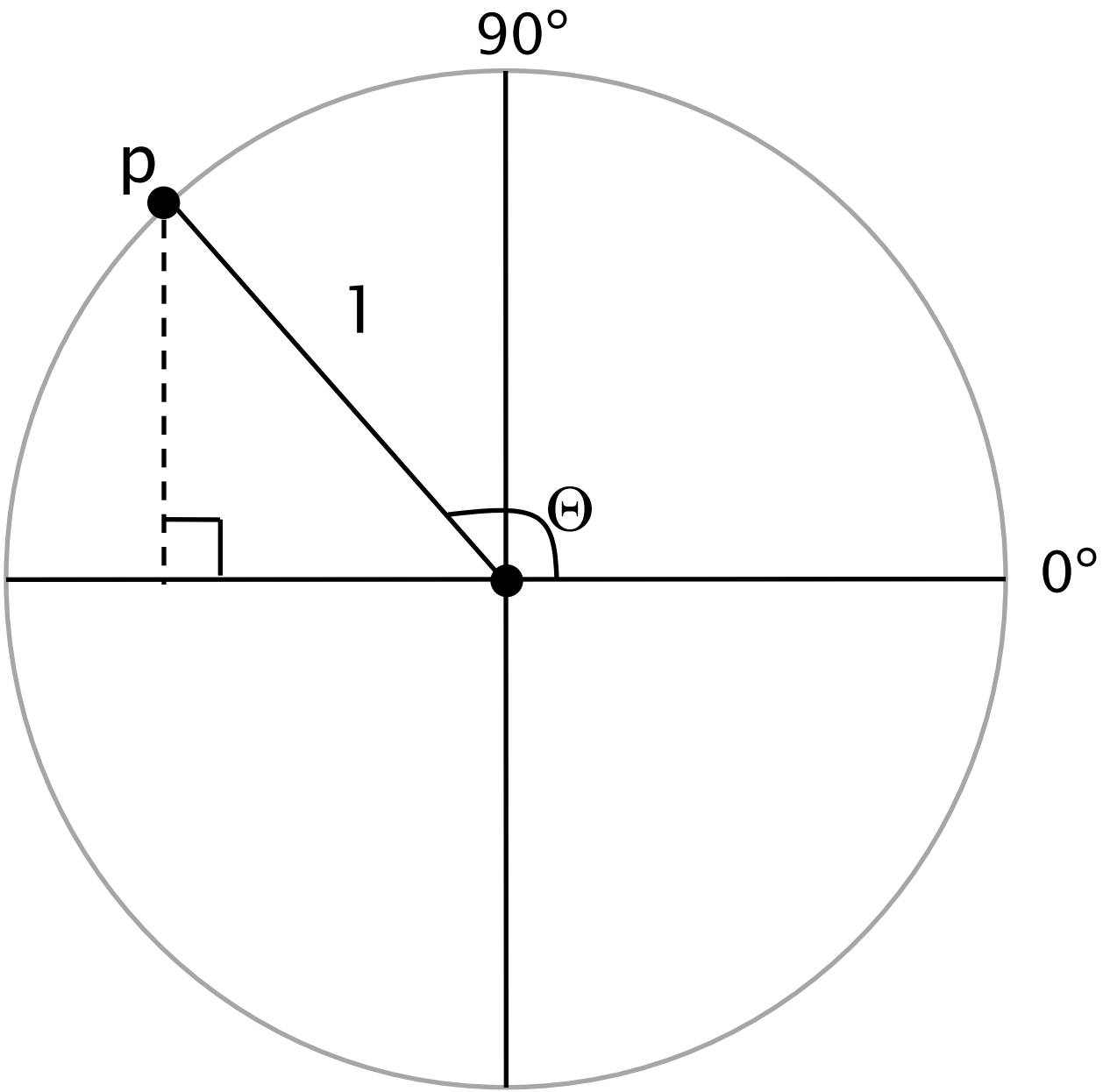
- $\text{tangent}(\Theta) = o/a = \sin(\Theta)/\cos(\Theta)$

sohcahtoa

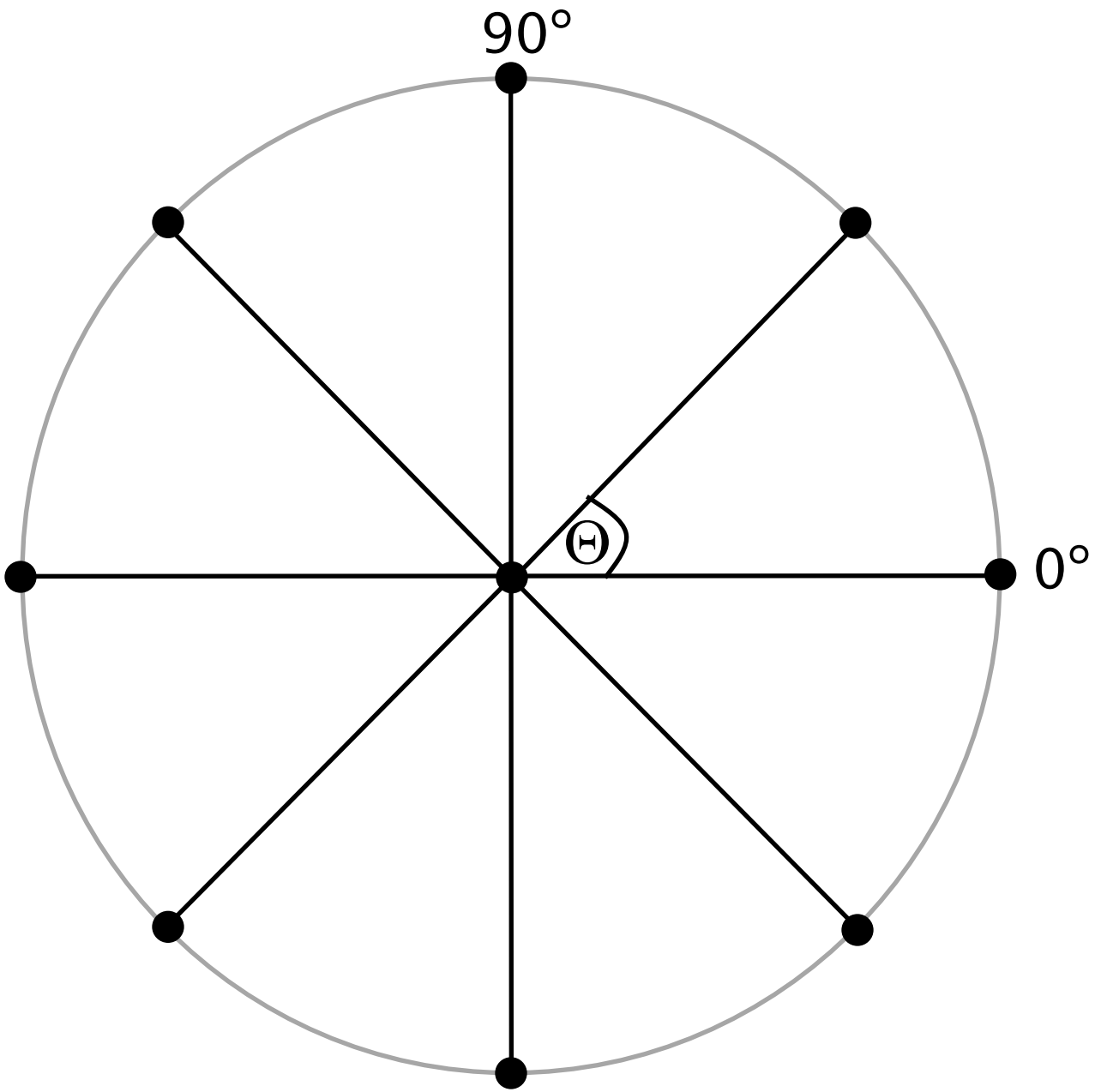
Trigonometry on a unit circle



Trigonometry on a unit circle



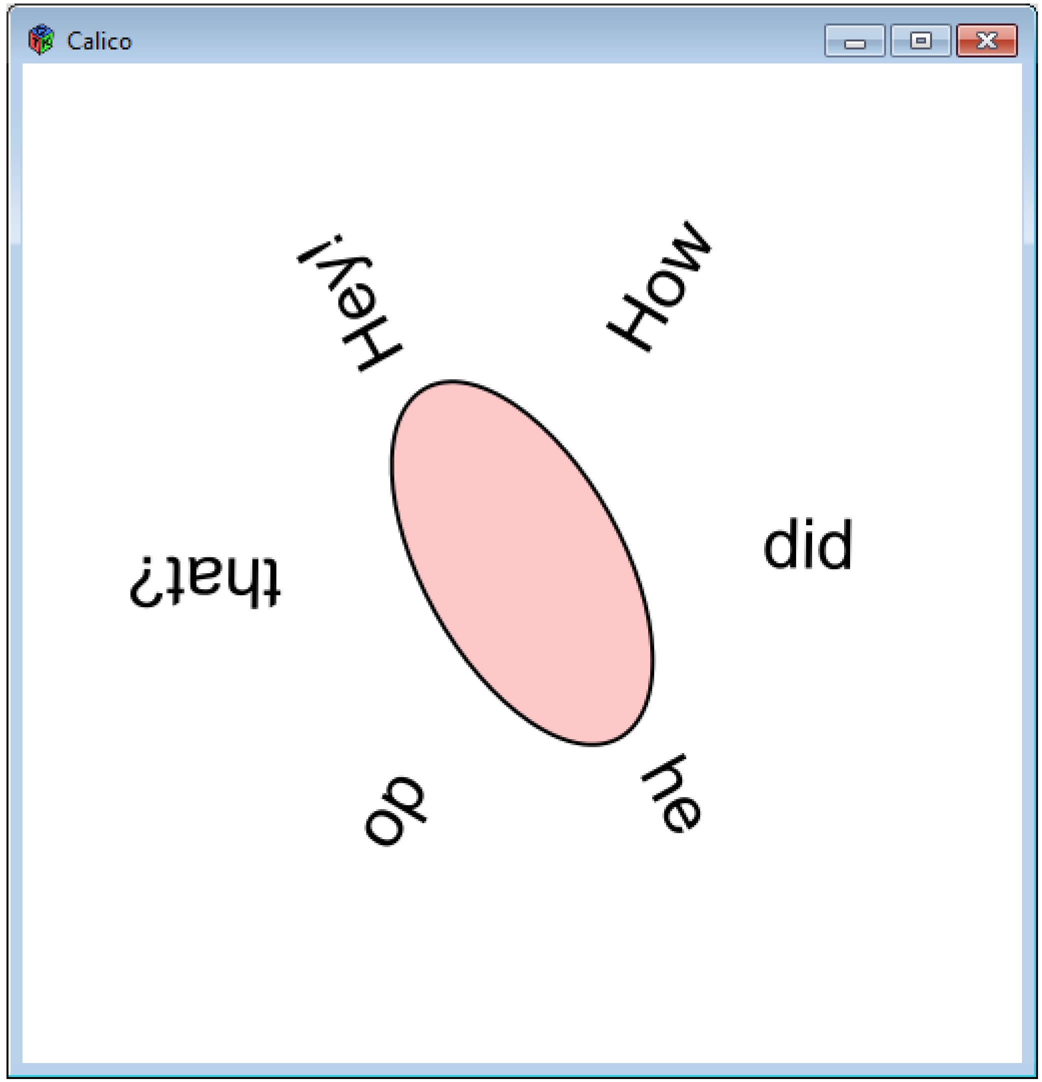
Trigonometry on a unit circle



Drawing points along a circle

```
from Processing import *
from math import sin, cos

window(500, 500)

steps = 80
radius = 200
angle = 2*PI/steps

for i in range(steps):
    x = sin(angle*i)*radius
    y = cos(angle*i)*radius
    # draw a point every 1/8th of a circle
    ellipse(250+x, 250+y, 10, 10)
```

circle.py



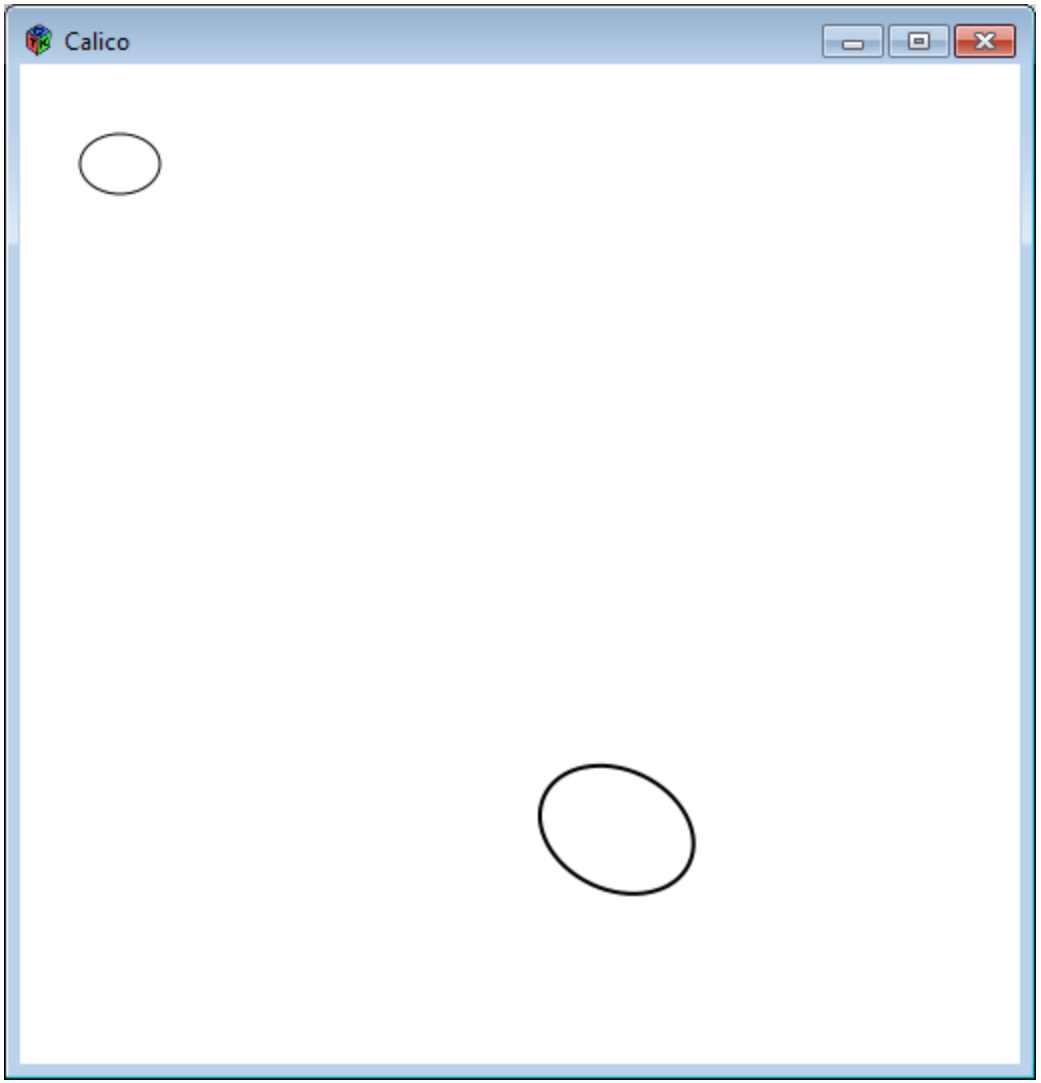
example1.py

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…

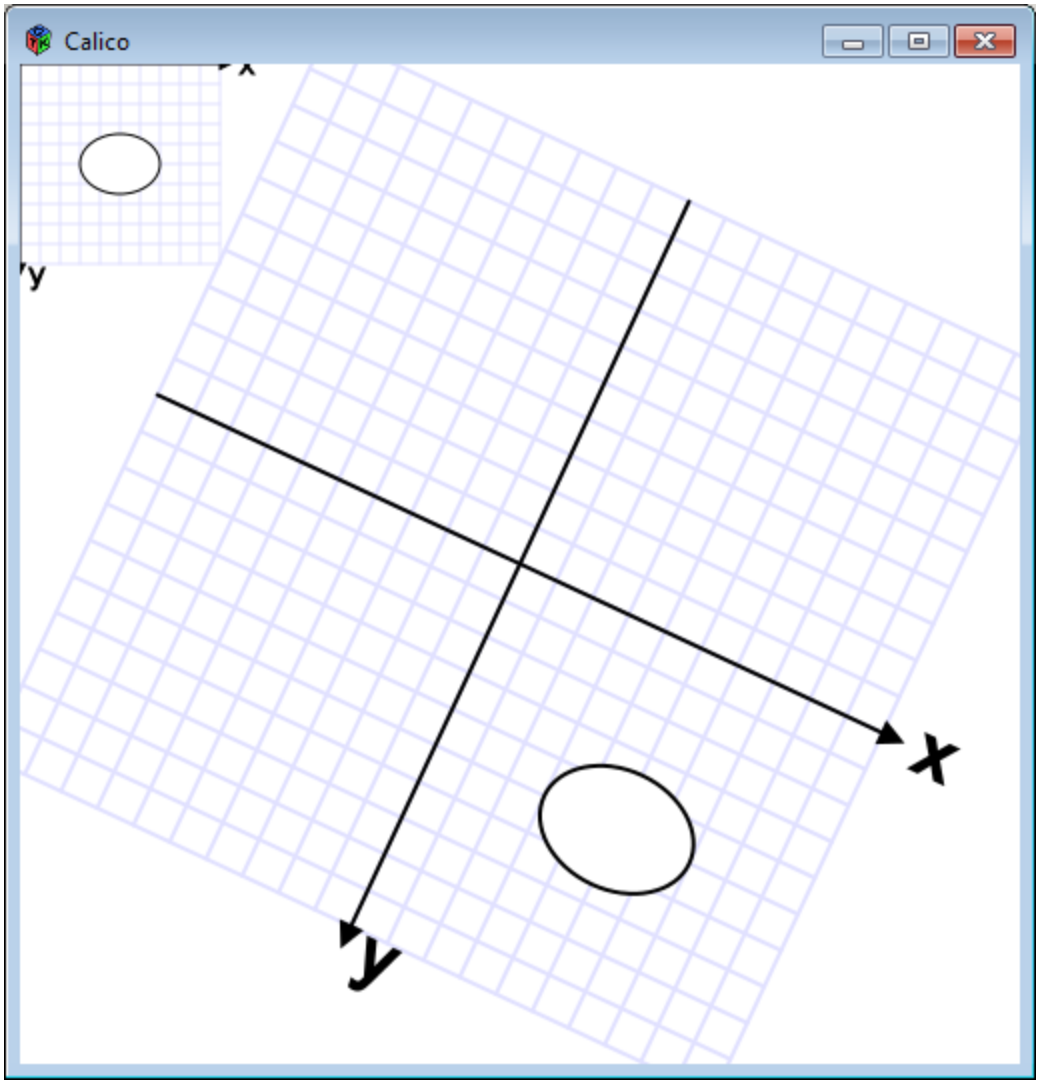
- *We can leave coordinate values unchanged, and* ***modify the coordinate system*** *in which we draw.*



The commands that draw these two ellipses are identical.

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

rotate2.py



The commands that draw these two ellipses are identical.

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

rotate2.py

Three ways to transform the coordinate system:

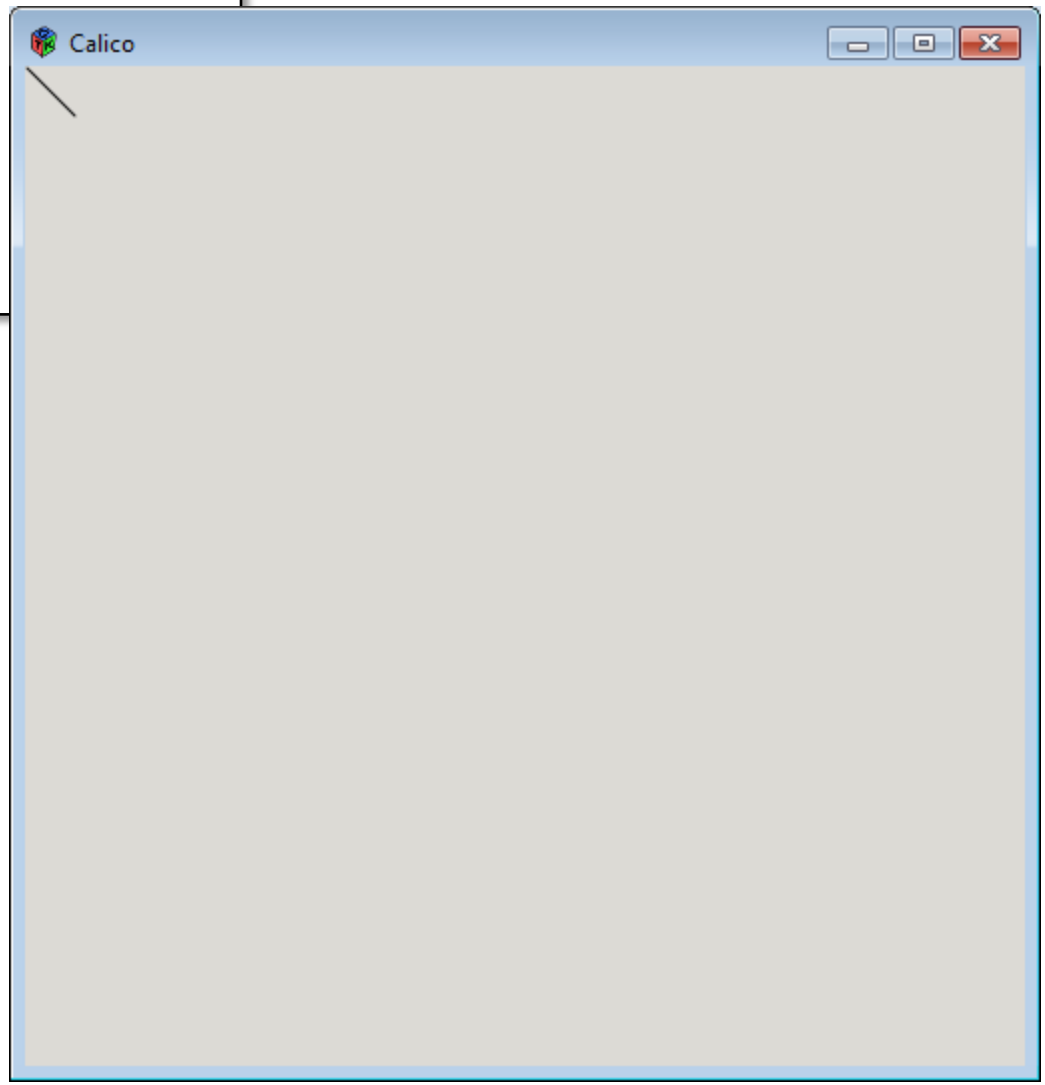
1. **Translate**
   - Move axes left, right, up, down …
2. **Scale**
   - Magnify, zoom in, zoom out, about the origin …
3. **Rotate**
   - Tilt clockwise, counter-clockwise, about the origin …

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.

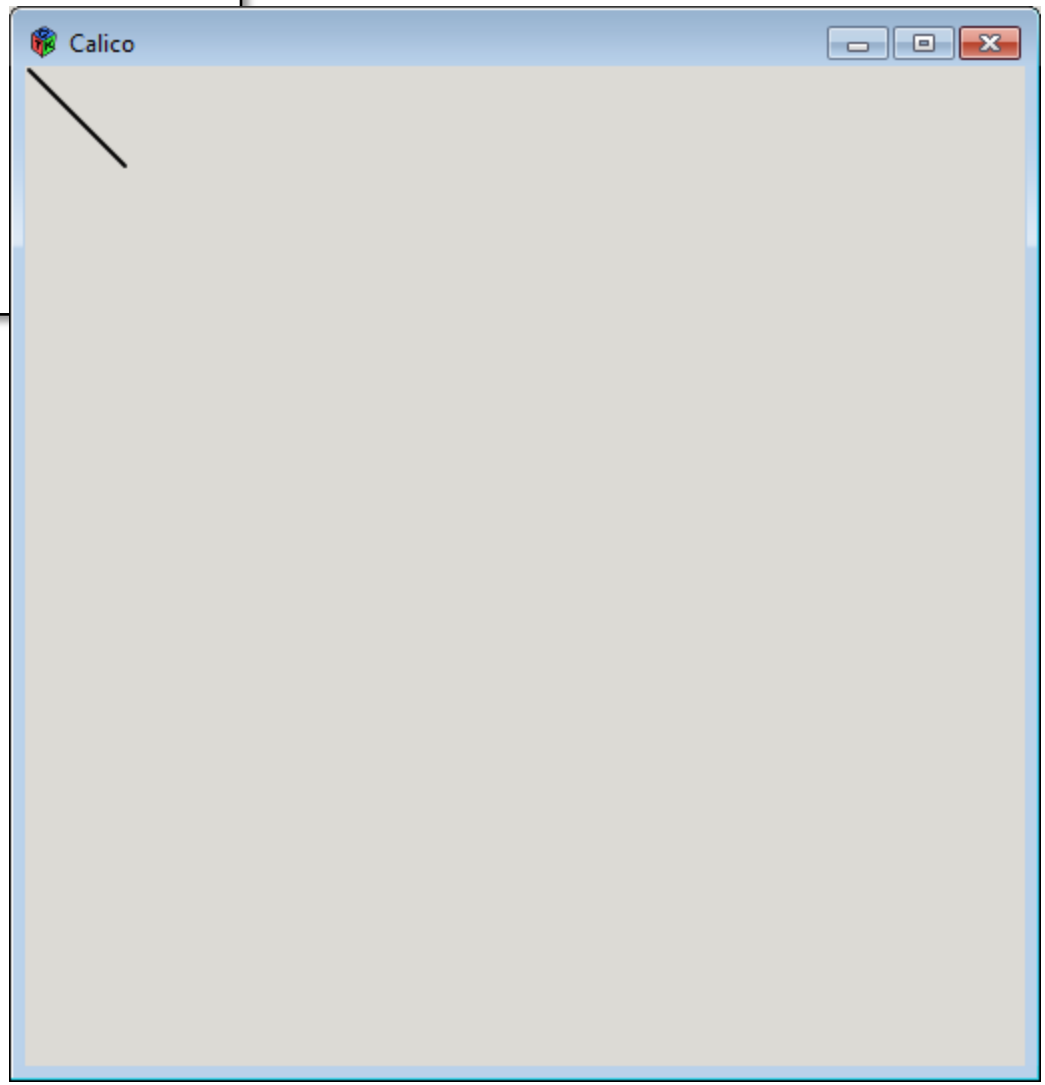
```
scale( factor )
scale( x-factor, y-factor )
```

```
from Processing import *

window(500, 500)

line(1, 1, 25, 25)
```



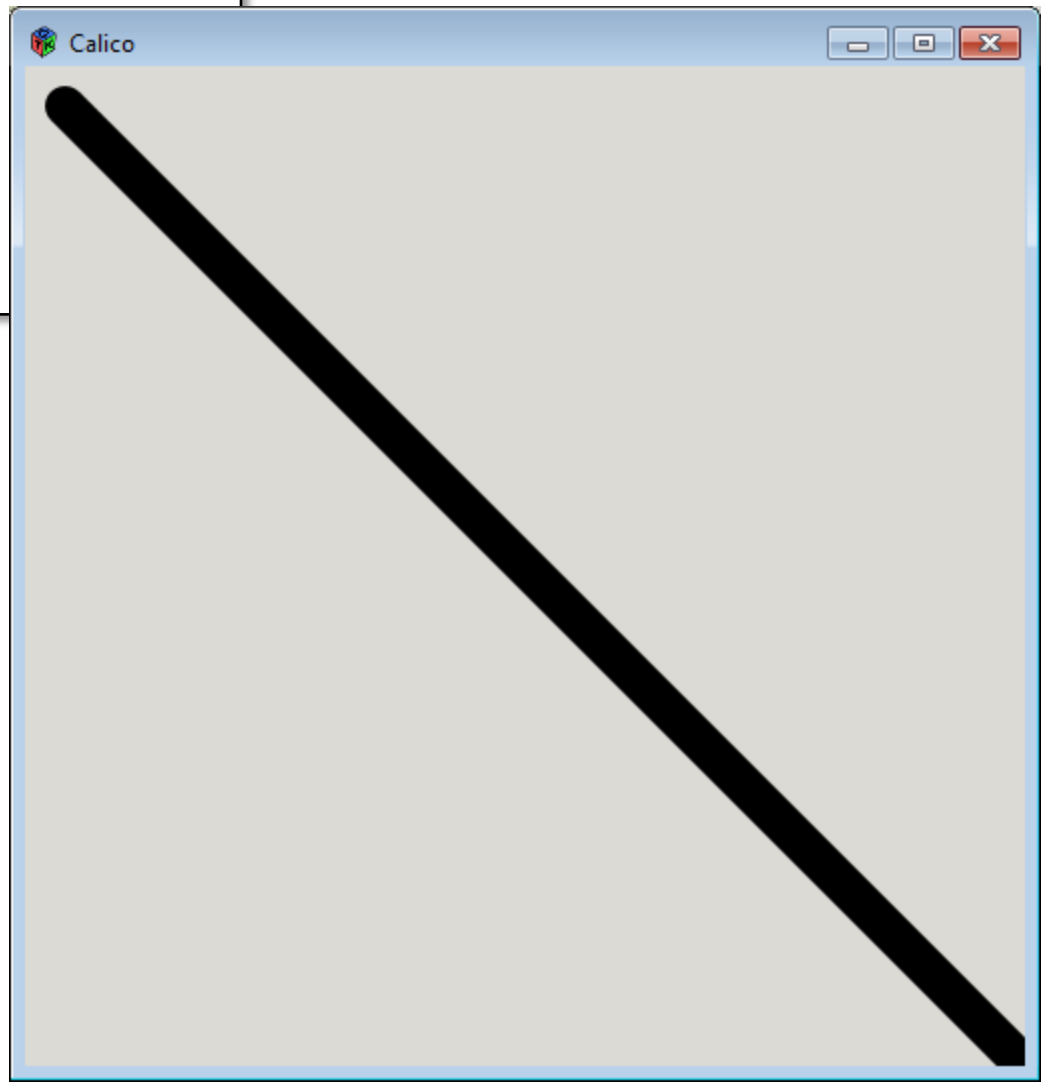
example2.pde

```
from Processing import *

window(500, 500)

scale(2,2)
line(1, 1, 25, 25)
```



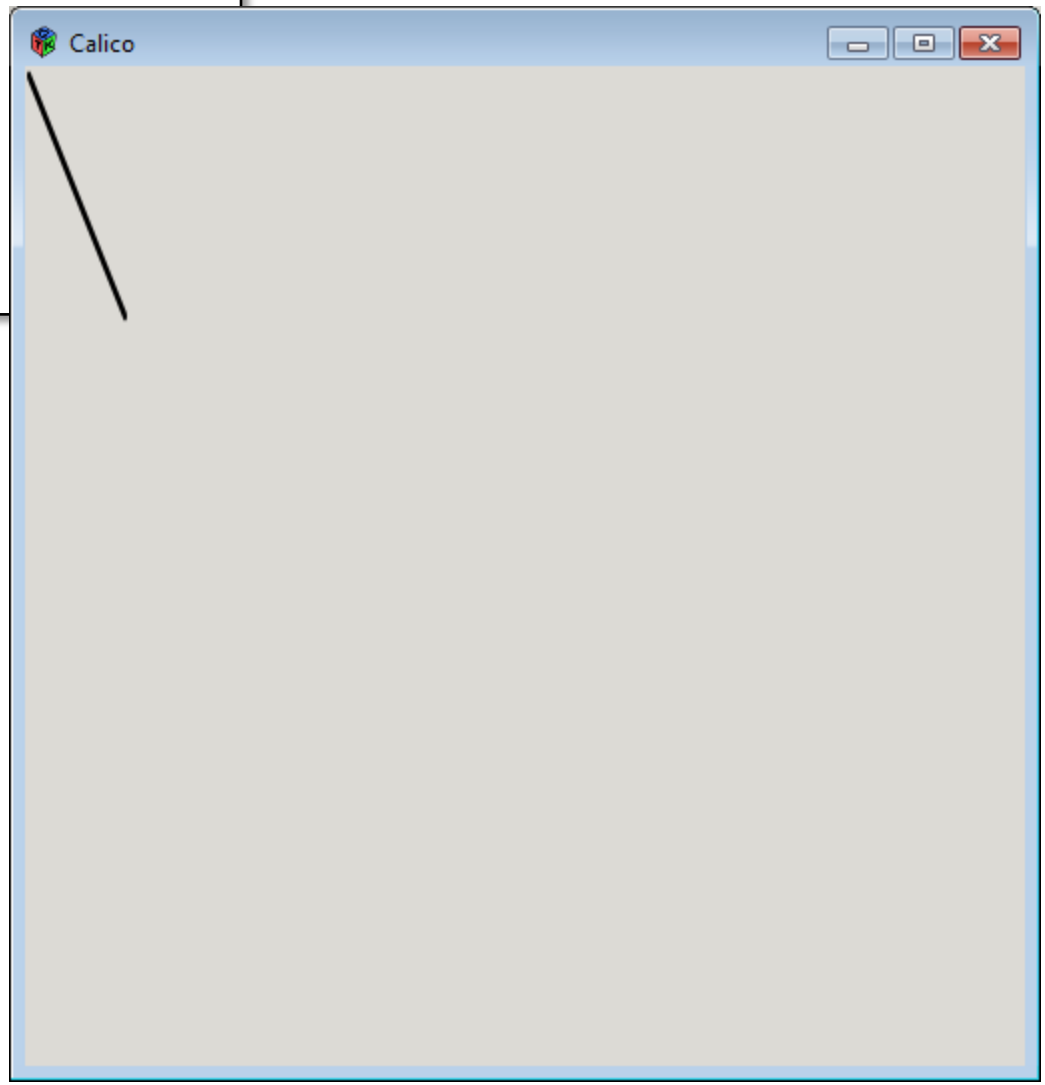
example2.pde

```
from Processing import *

window(500, 500)

scale(20,20)
line(1, 1, 25, 25)
```



example2.pde

```
from Processing import *

window(500, 500)

scale(2,5)
line(1, 1, 25, 25)
```

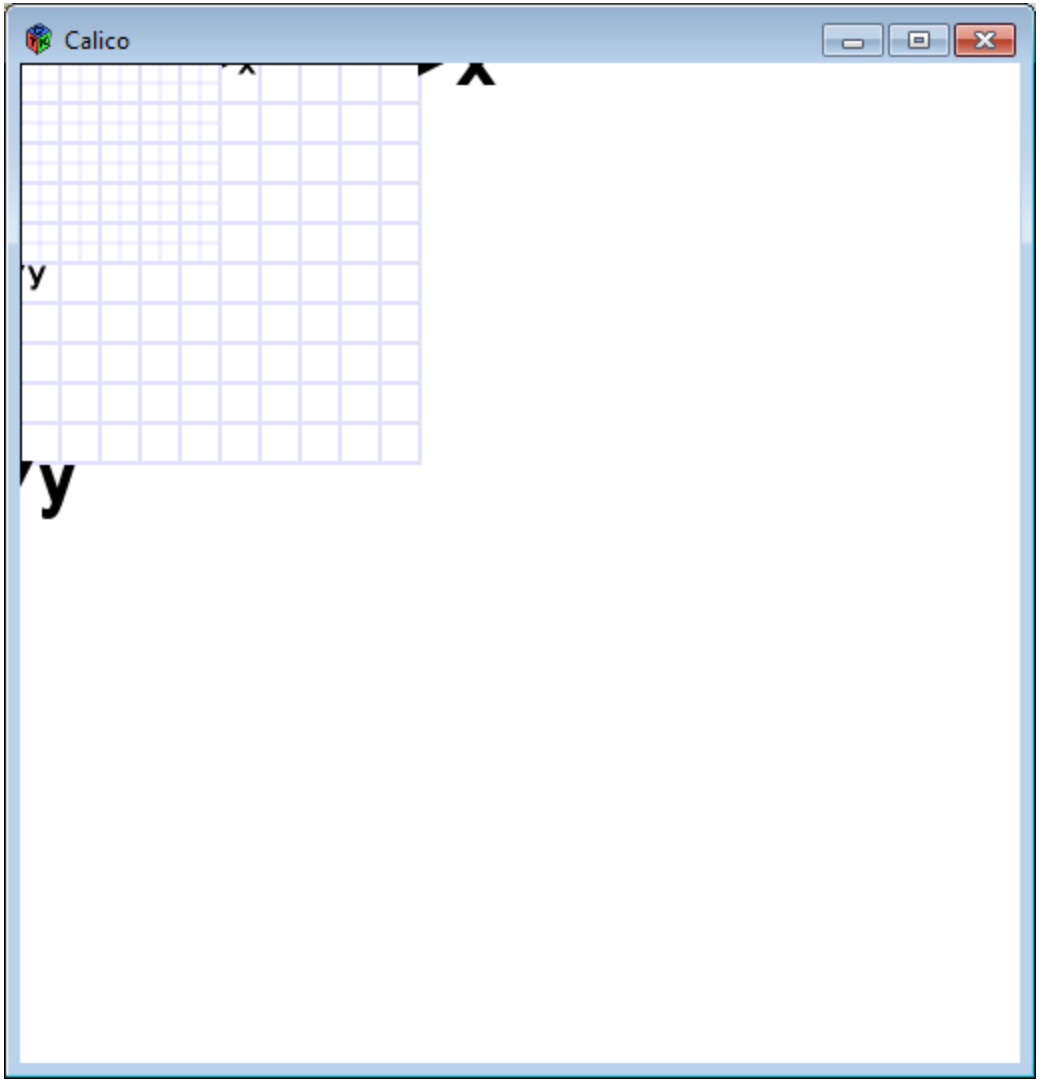


example2.pde

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

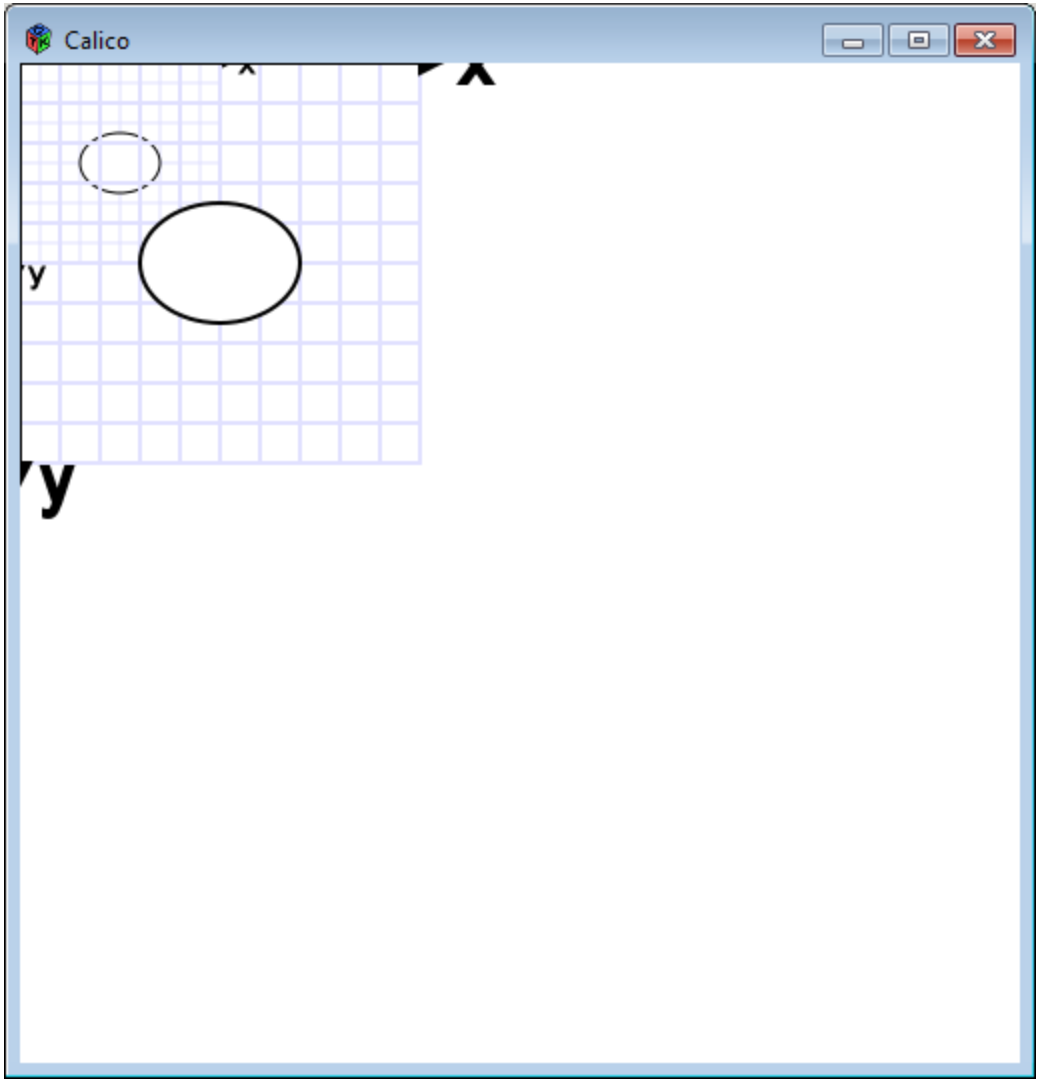
```
# Draw a grid
def grid():
    x1, x2, dx = -100.0, 100.0, 10.0
    y1, y2, dy = -100.0, 100.0, 10.0

    # Draw grid
    stroke(225,225,255)
    x = x1
    while x <= x2:
        line(x,y1,x,y2)
        x+=dx
    y = y1
    while y <= y2:
        line(x1,y,x2,y)
        y+=dy

    # Draw axes
    inc = 0.005*width()
    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)
```

```
from Processing import *
window(500, 500)

background(255)
grid()
scale(2,2)
grid()
```



grid.pde

```
from Processing import *
window(500, 500)

background(255)
grid()
fill(255)
ellipse(50,50,40,30)
scale(2,2)
grid()
fill(255)
ellipse(50,50,40,30)
```



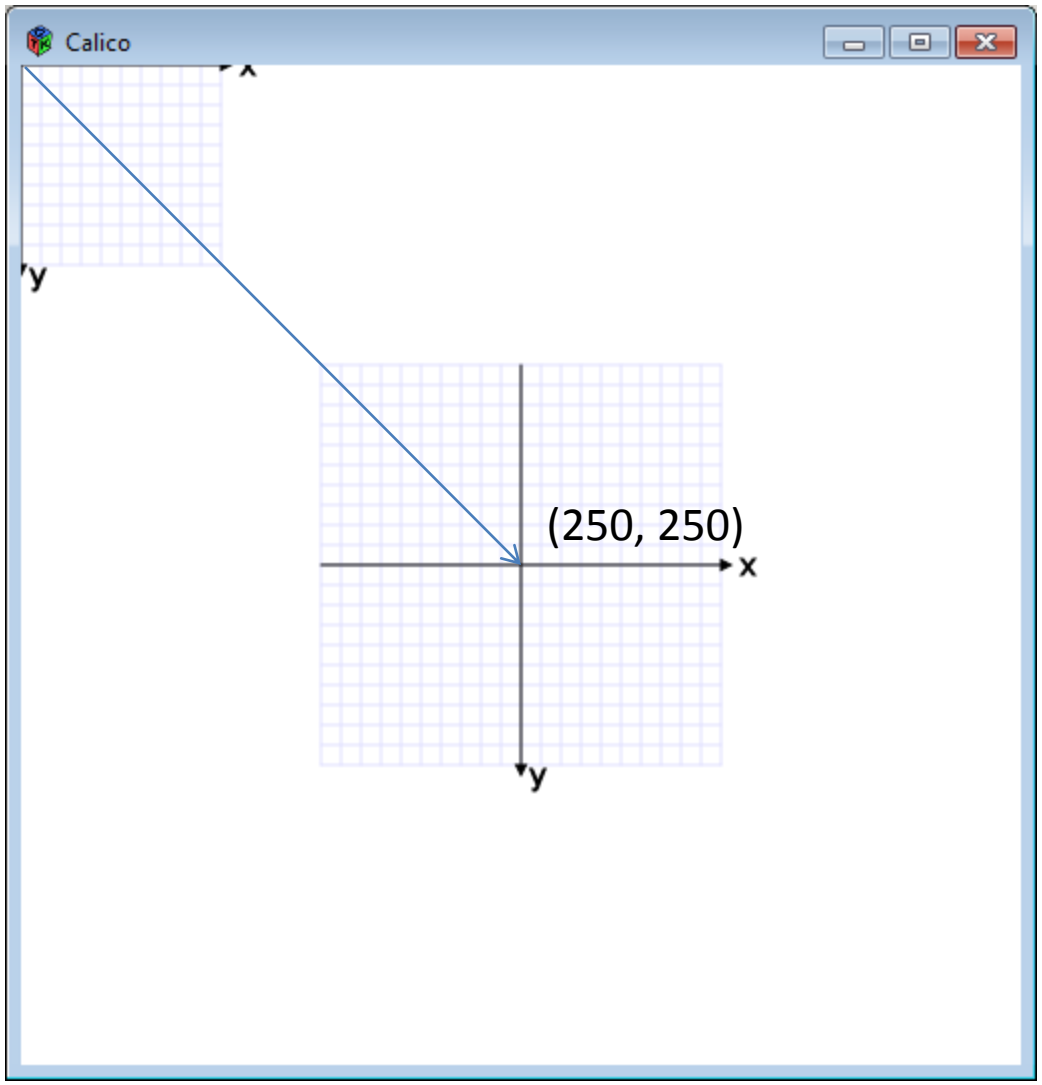
grid.pde

Translate

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

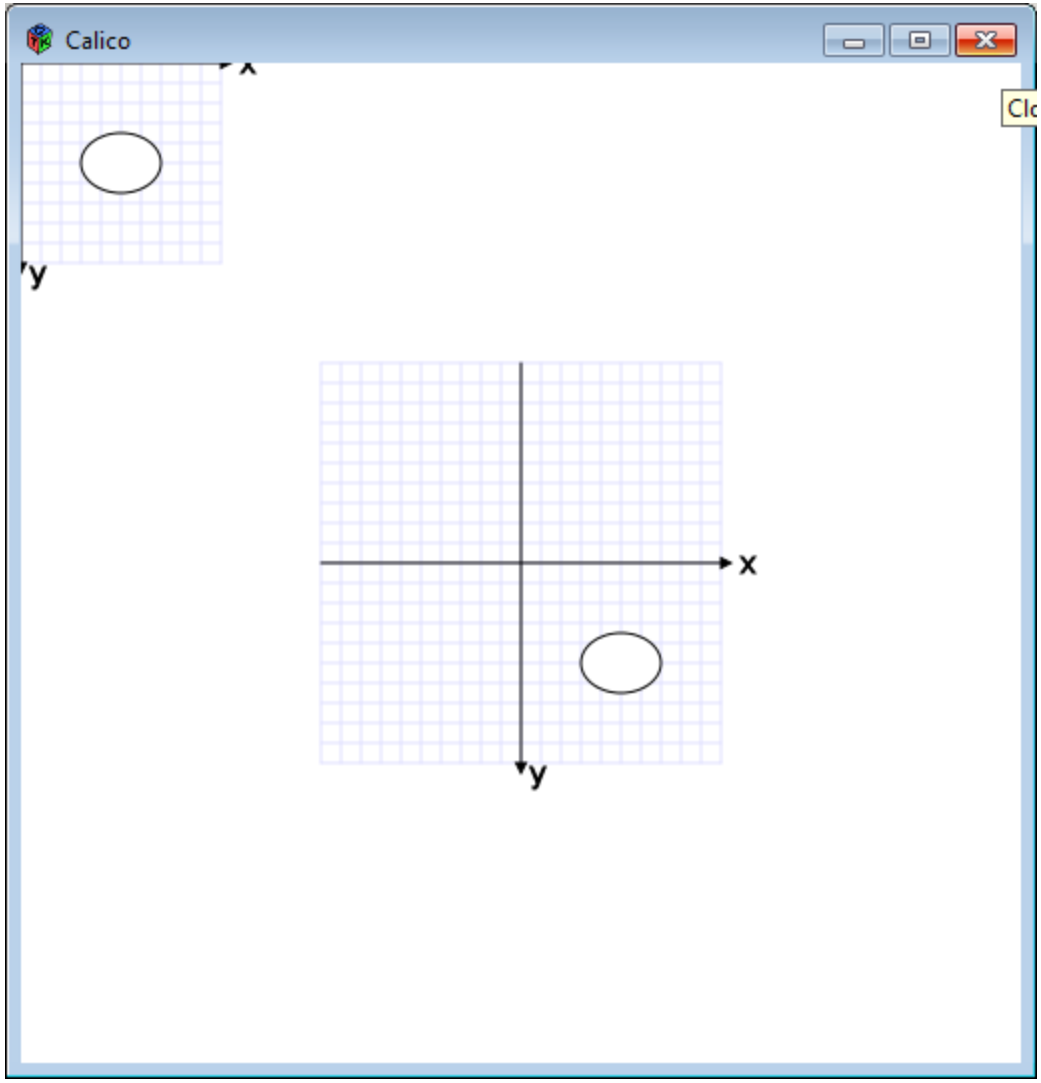
```
translate( x-shift, y-shift)
```

```
from Processing import *
window(500, 500)

background(255)
grid()
translate(250, 250)
grid()
```



grid2.pde

```
from Processing import *
window(500, 500)

background(255)
grid()
fill(255)
ellipse(50, 50, 40, 30)

translate(250, 250)
grid()
fill(255)
ellipse(50, 50, 40, 30)
```

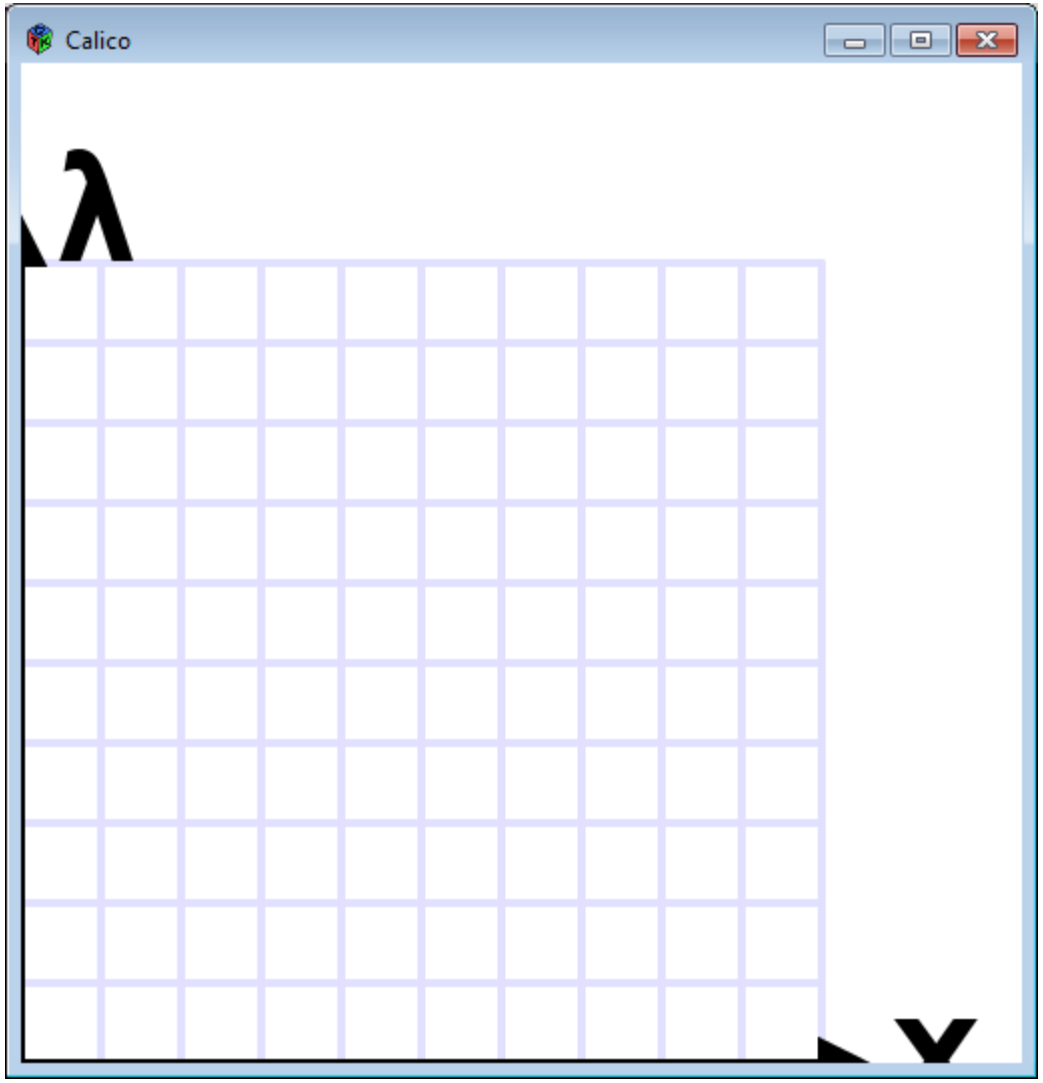


Transformations can be combined

- 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!

```
from Processing import *
window(500, 500)

background(255)
#grid()

translate(0, height())
scale(4, -4)
grid()
```



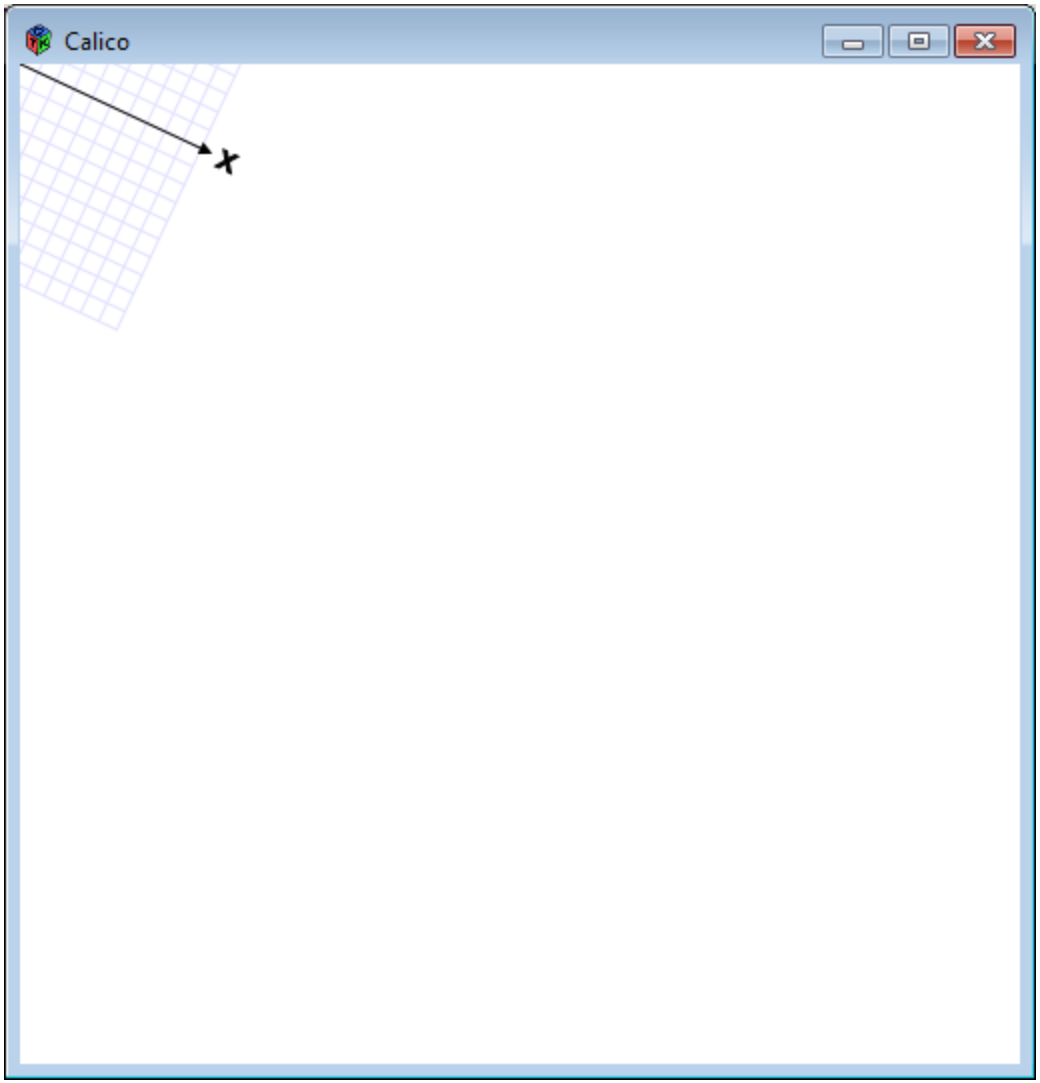
grid.pde

Rotate

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

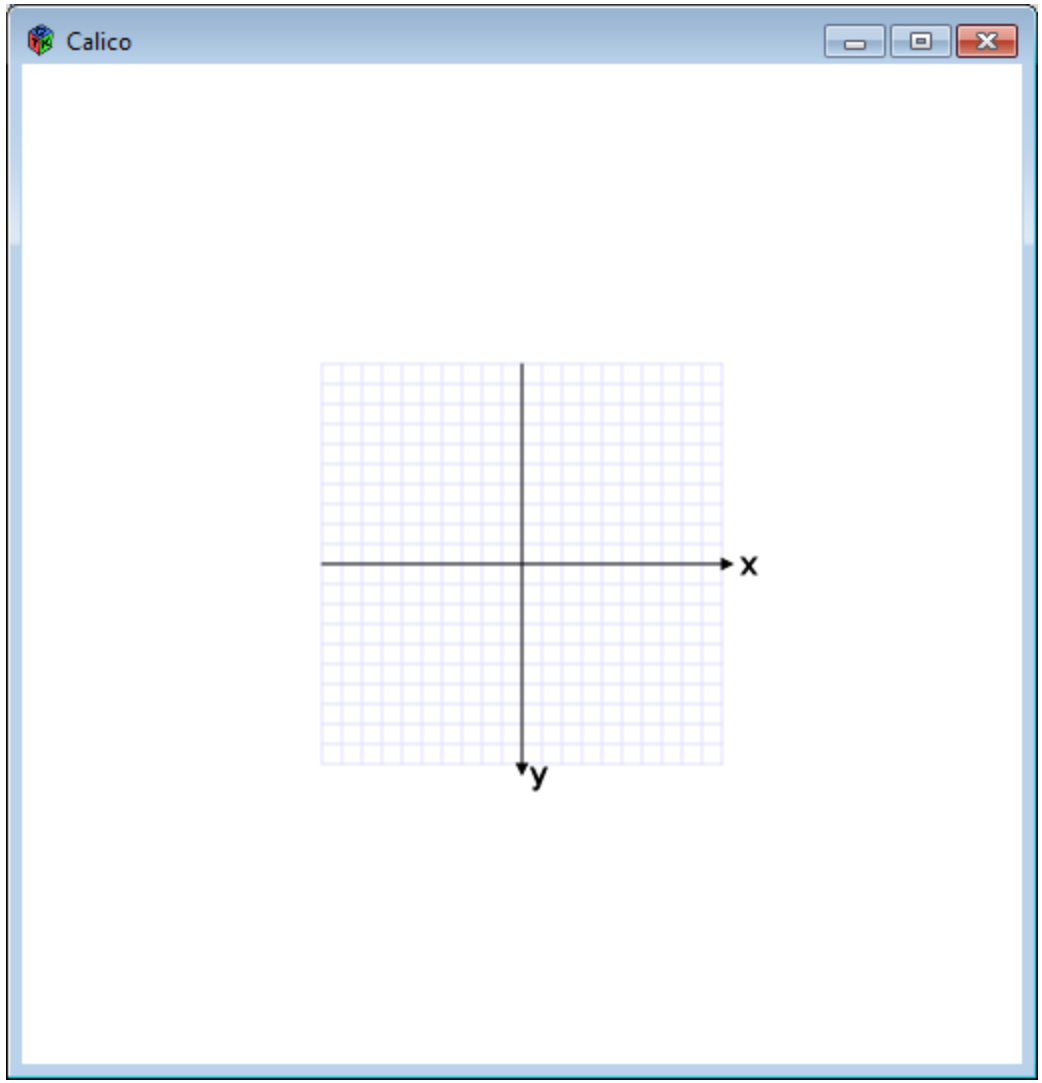
```
rotate( radians )
```

```
from Processing import *
window(500, 500)

background(255)

rotate(25.0 * (PI/180.0) )
grid()
```



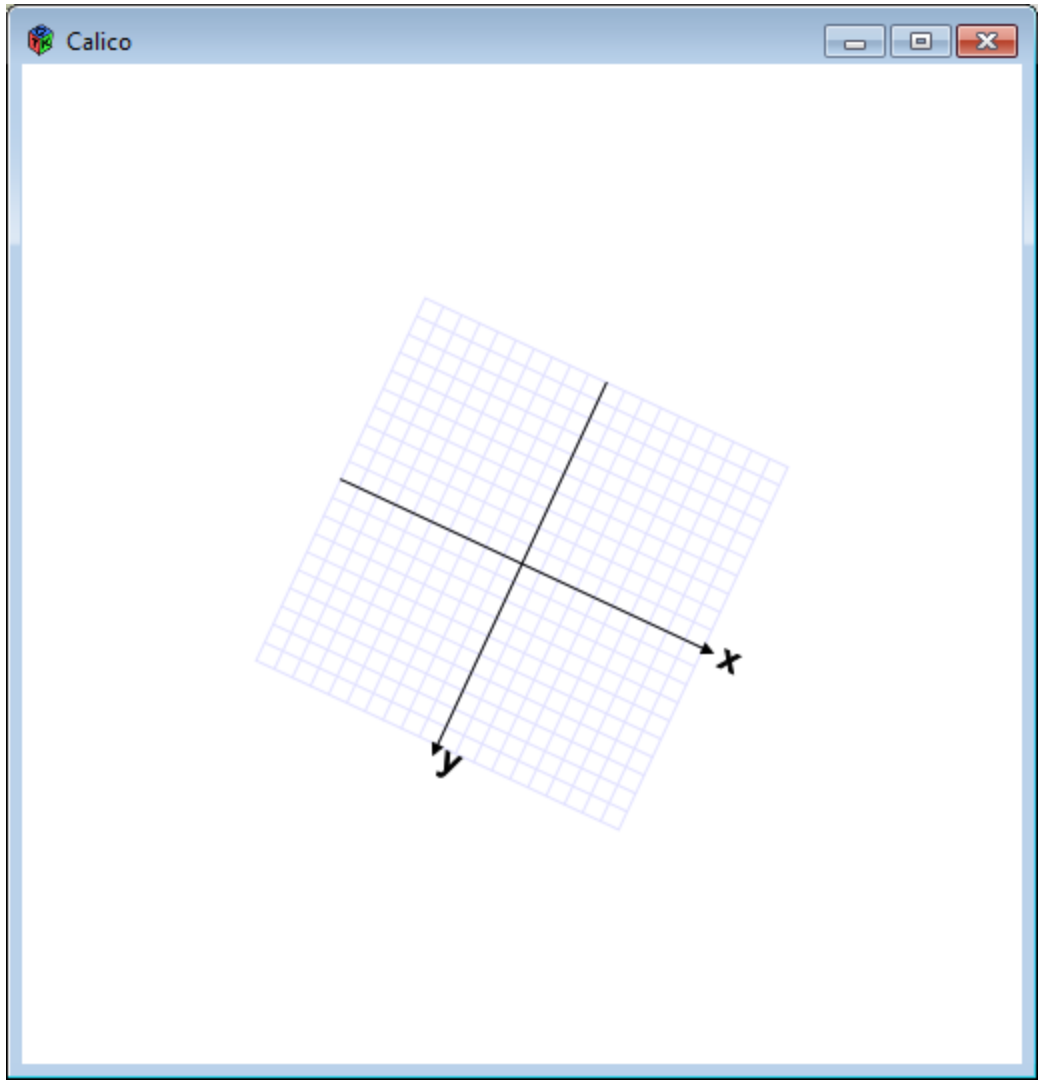
grid.pde

```
from Processing import *
window(500, 500)

background(255)

translate(250.0, 250.0)
#rotate( 25.0 * (PI/180.0) )
#scale( 2 )
grid()
```



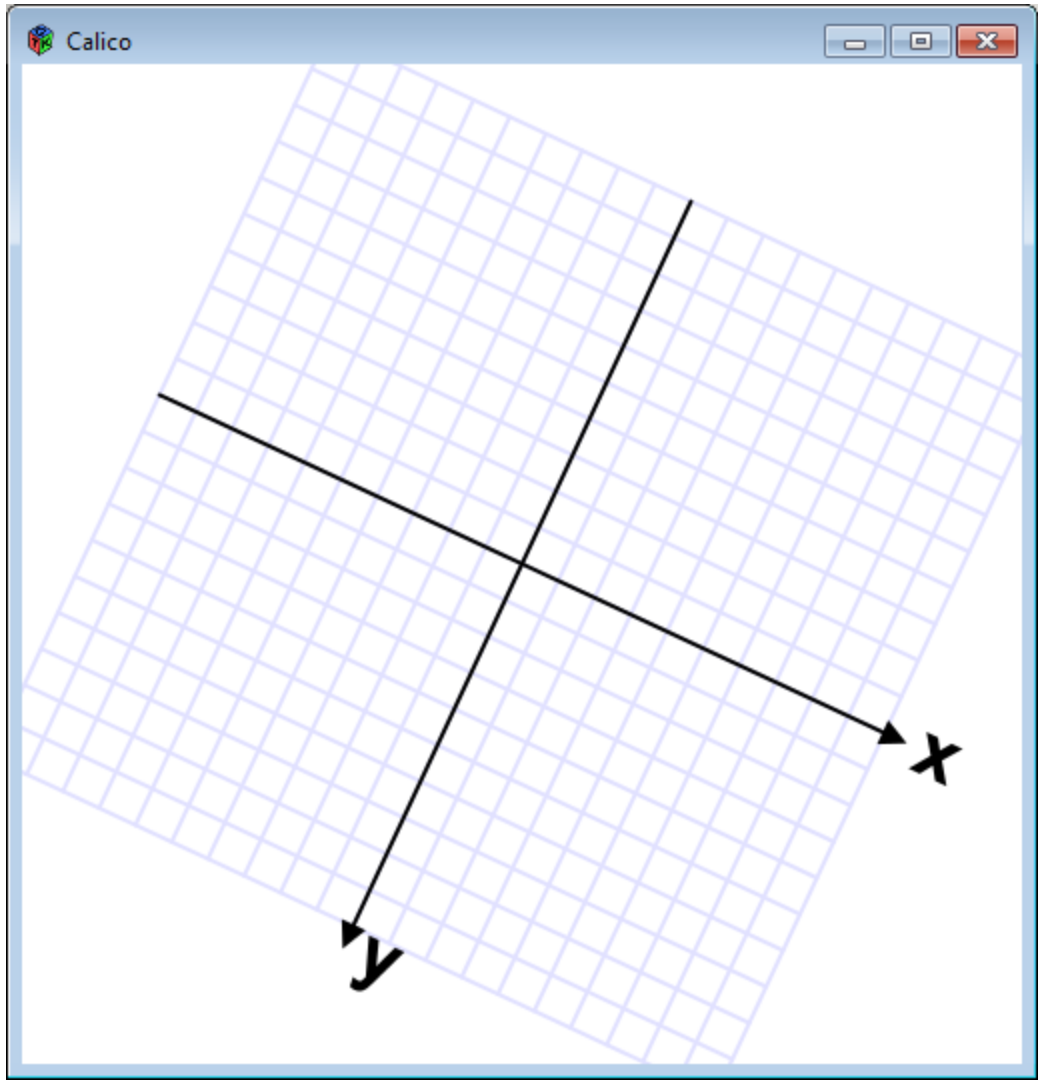
grid.pde

```
from Processing import *
window(500, 500)

background(255)

translate(250.0, 250.0)
rotate( 25.0 * (PI/180.0) )
#scale( 2 )
grid()
```



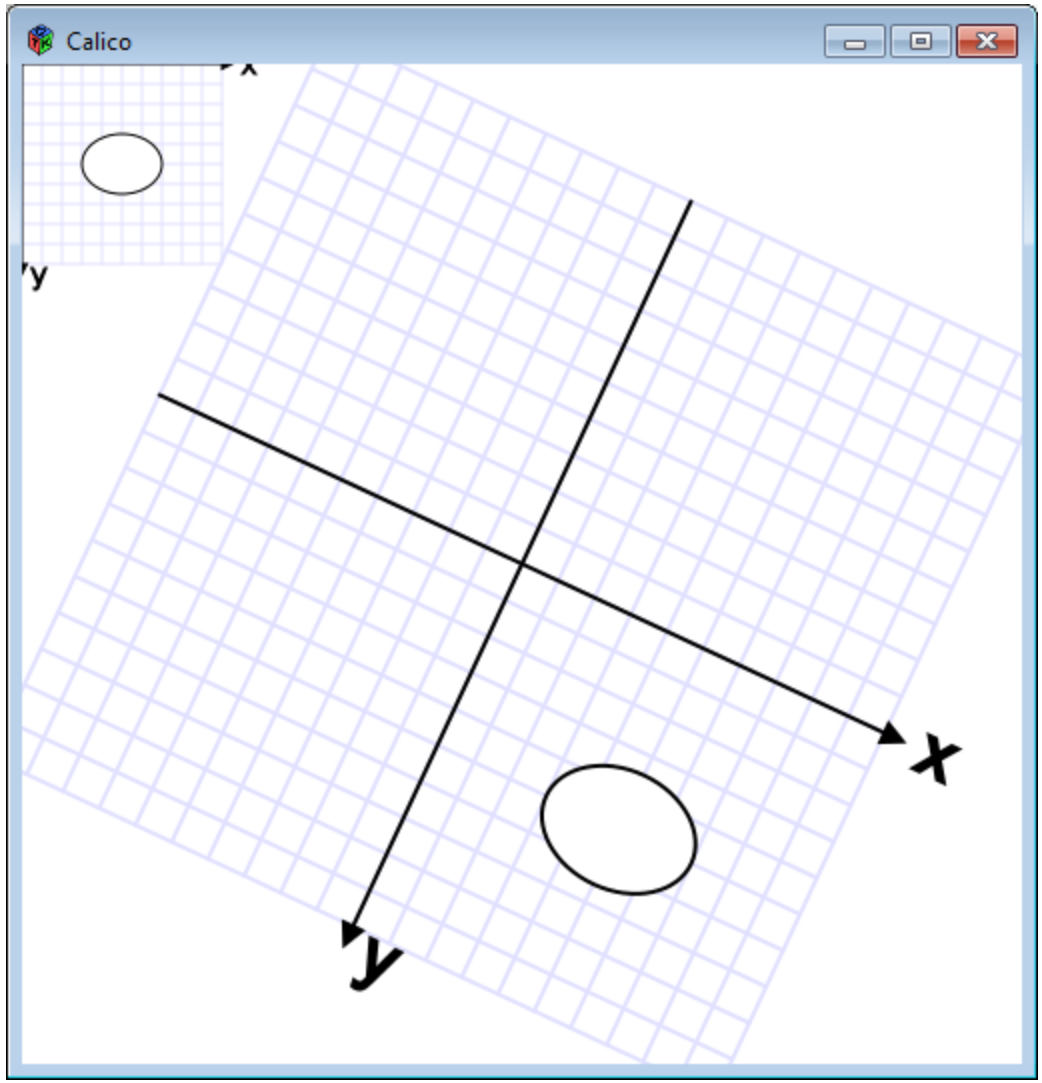
grid.pde

```
from Processing import *
window(500, 500)

background(255)

translate(250.0, 250.0)
rotate( 25.0 * (PI/180.0) )
scale( 2 )
grid()
```



grid.pde

```
from Processing import *
window(500, 500)

background(255)
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)
```

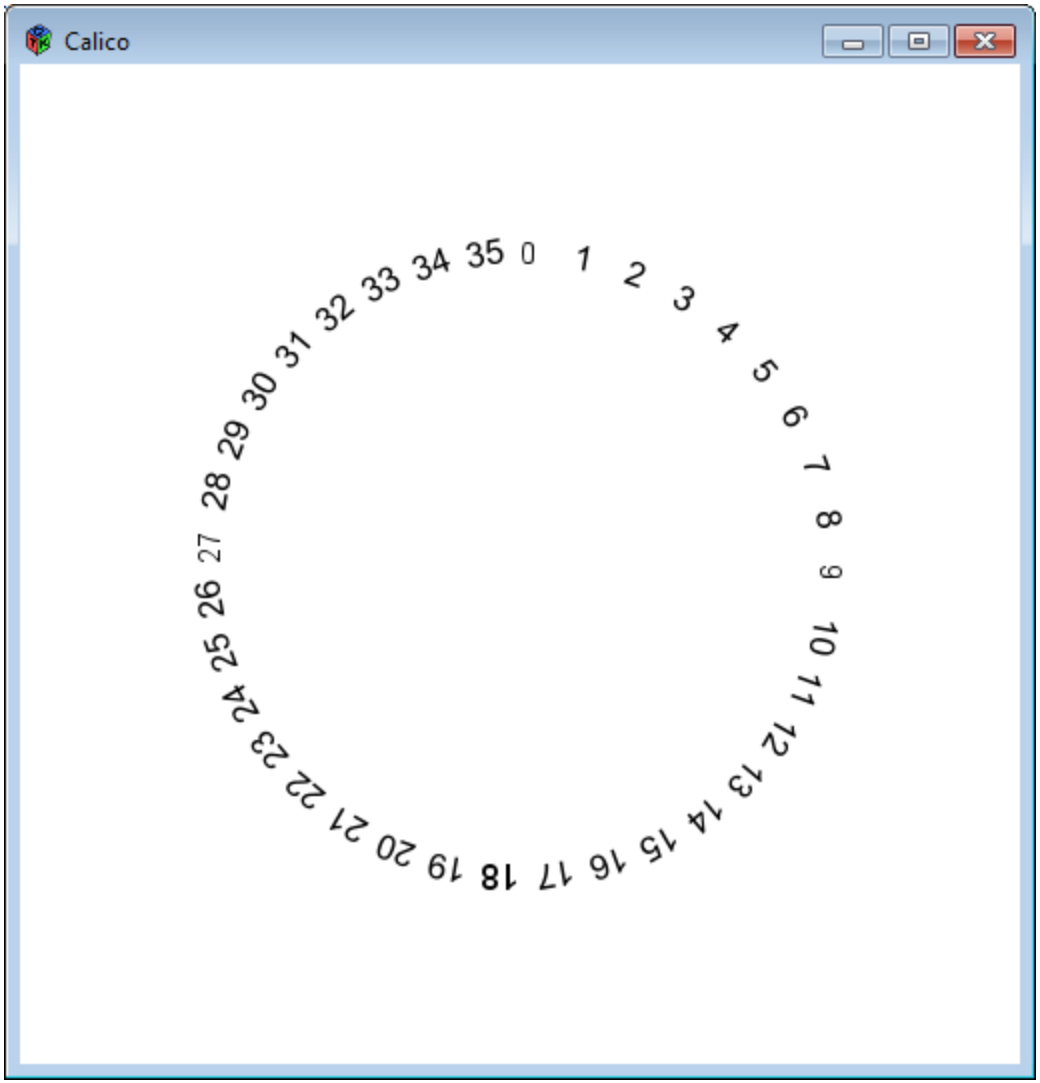


grid.pde

Some things to remember:

1. Transformations are cumulative.
2. Rotation angles are measured in radians
   - π radians = 180°
   - radians = (PI/180.0) * degrees
3. Order matters

```
# example3.py

from Processing import *
window(500, 500)

background(255)
noStroke()
fill(0)

translate( 0.5*width(), 0.5*height()

for i in range(36):
    text( i, 0.0, -150.0 )
    rotate(10.0 * (PI/180.0))
```

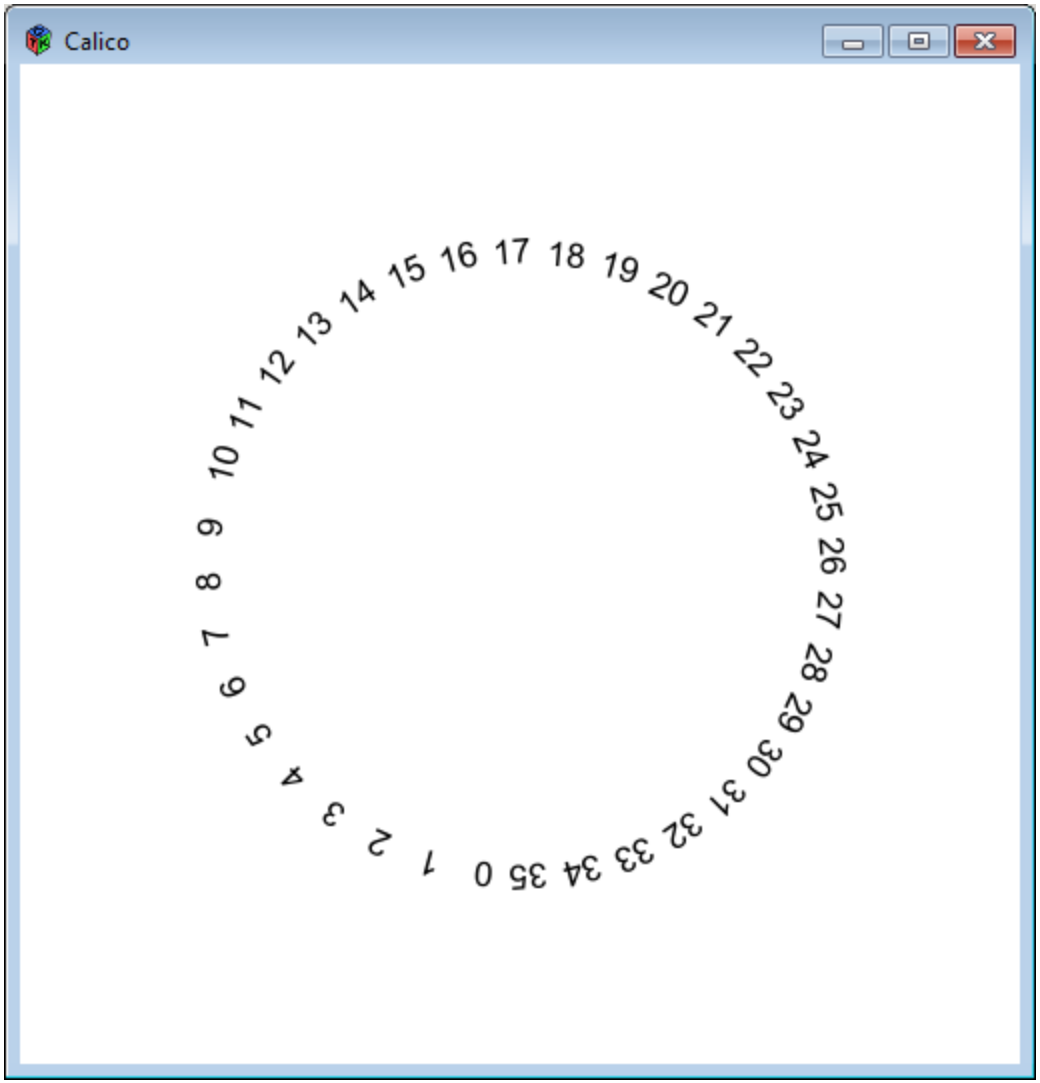


example3.pde

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

Total rotation accumulates.

```
# example4.py

from Processing import *
window(500, 500)

start = 0.0
w = width()
h = height()

def draw(o, e):
    background(255)
    noStroke()
    fill(0)

    translate( 0.5*w, 0.5*h )
    rotate(start)

    for i in range(36):
        text( i, 0.0, -150.0 )
        rotate(10.0 * (PI/180.0))

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

frameRate(50)
onLoop += draw
loop()
```



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

Problem

- Transformations accumulate!

- resetMatrix()
  - Roll back all transformations to original

- How roll back some transformations, not all?
  - pushMatrix()
    - Saves the current transformation state
  - Perform transformation and drawing, as needed
  - popMatrix()
    - Restores transformation to state when pushMatrix was called

```
# pushpop1.py
from Processing import *

window(500, 500)

# Translate the origin of the coordinate system
# to the center of the sketch window
translate(250, 250)

# Rotate and draw a line and ellipse
def draw(o, e):
    rotate( radians(5) )
    line(0, 0, 100, 0)          # Drawing args are constant
    ellipse( 100, 0, 10, 10)

# Draw again on mouse pressed
onMousePressed += draw
```

- Can we use resetMatrix() to prevent the accumulation of transformations?
- What if we move translate() into the draw() function?

- Always bracket your transformations with pushMatrix() and popMatrix() unless you explicitly want to accumulate transformations

- pushMatrix() and popMatrix() can be nested

More Examples

textSki.py

sun.py

Assignment #3