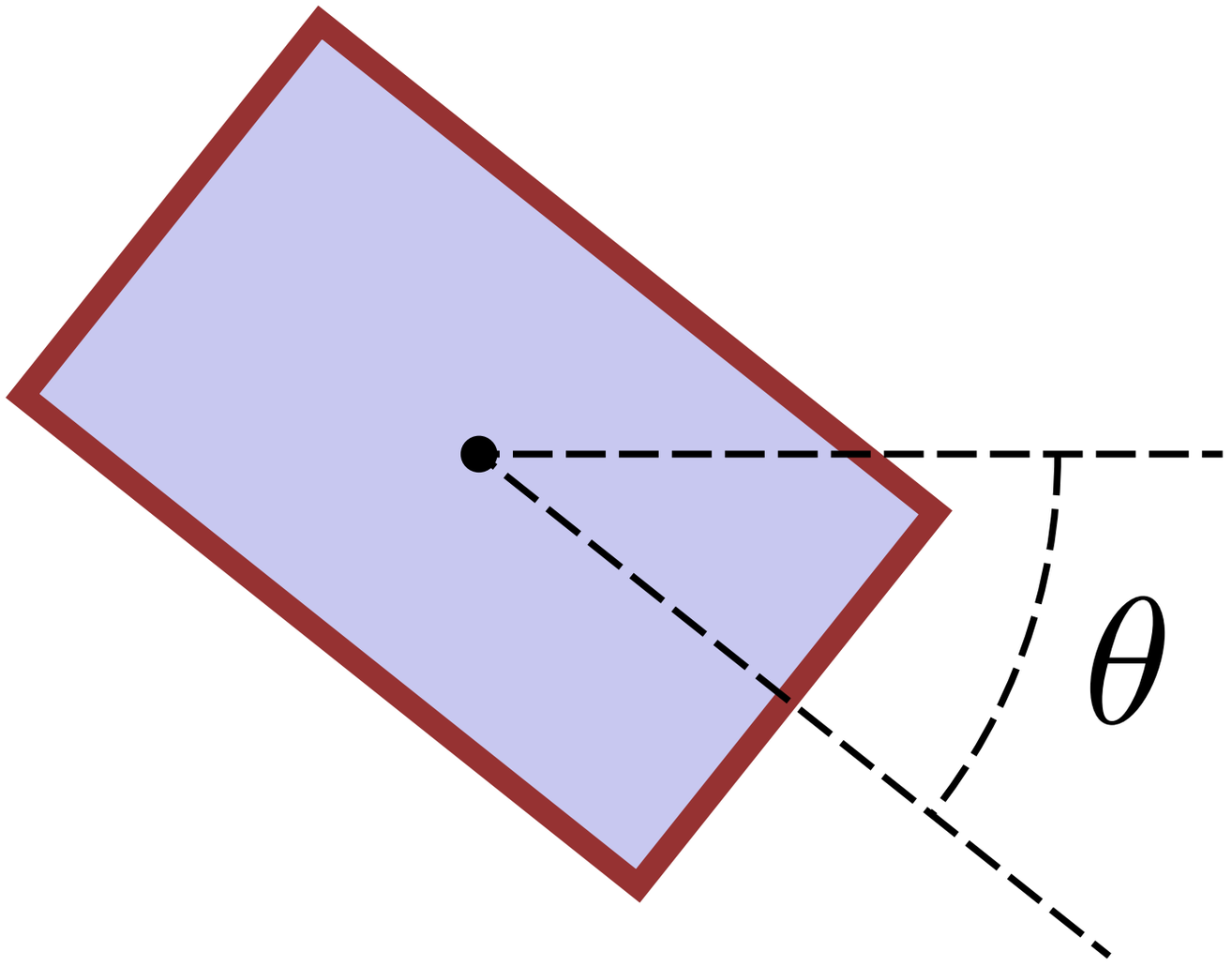
Module 06

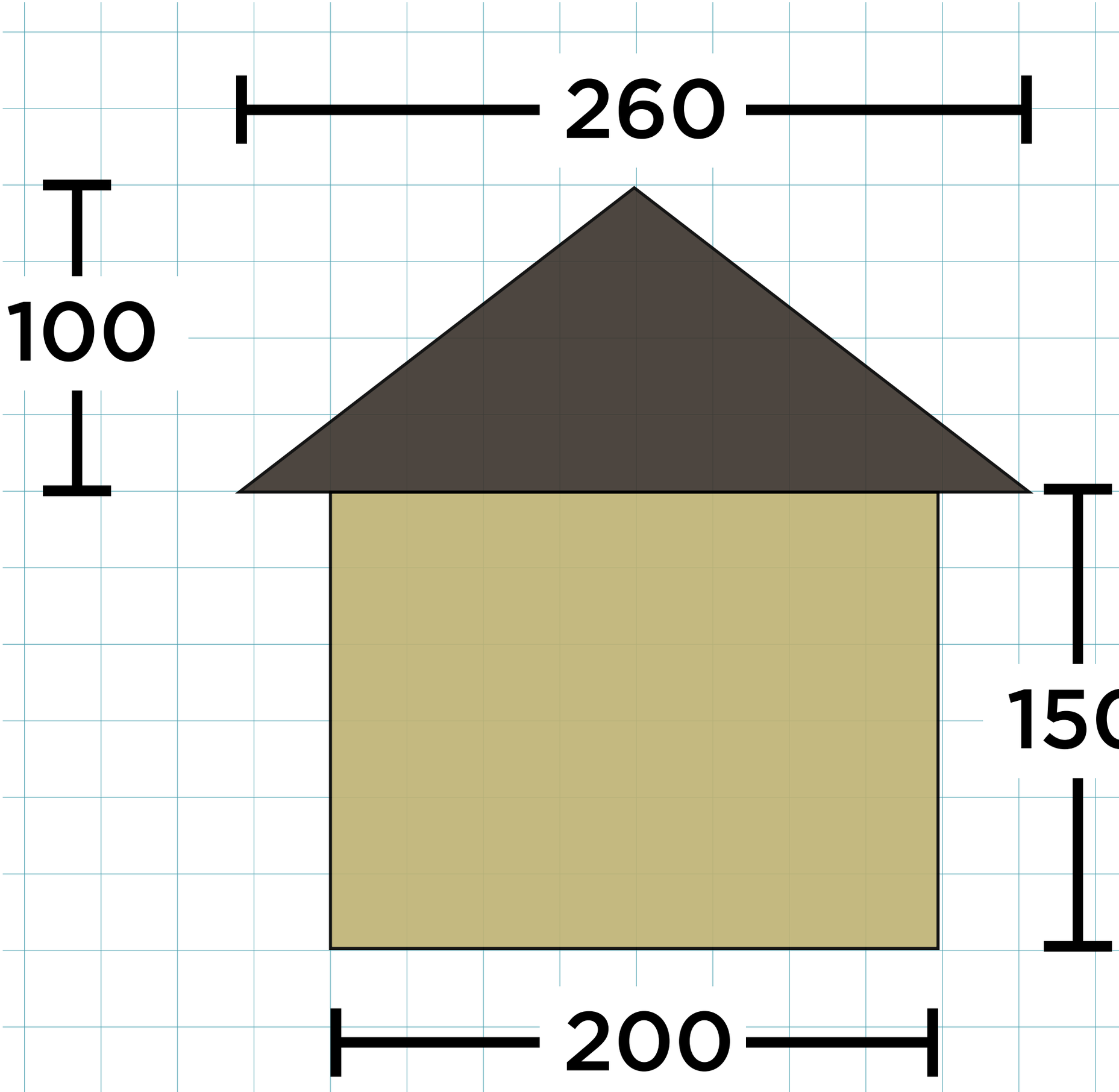
Geometric Context

translate()
rotate()
scale()

pushMatrix()
popMatrix()

Limitations




260
100
150
200

```
void setup()
{
  size( 300, 300 );
  background( 255 );
  fill( 191, 179, 117 );
  rect( 50, 125, 200, 150 );
  fill( 62, 54, 47 );
  triangle( 150, 25, 20, 125, 280, 125 );
}
```

```
void setup()
{
  size( 300, 300 );
  background( 255 );
  fill( 191, 179, 117 );
  rect( 60, 125, 200, 150 );
  fill( 62, 54, 47 );
  triangle( 160  25,  30, 125, 290,  125 );
}
```

Two houses

```
void setup()
{
  size( 600, 350 );
  background( 255 );

  fill( 191, 179, 117 );
  rect( 50, 125, 200, 150 );
  fill( 62, 54, 47 );
  triangle( 150, 25, 20, 125, 280, 125 );

  fill( 191, 179, 117 );
  rect( 360, 115, 200, 150 );
  fill( 62, 54, 47 );
  triangle( 460, 15, 330, 115, 590, 115 );
}
```

```
void drawHouseAt( float x, float y )
{
  [illegible]
}
```

```
void setup()
{
  size( 600, 350 );
  background( 255 );

  drawHouseAt( 0, 0 );
  drawHouseAt( 310, -10 );
}
```

```
void drawHouseAt( float x, float y )
{
  fill( 191, 179, 117 );
  rect( 50 + x, 125 + y, 200, 150 );
  fill( 62, 54, 47 );
  triangle( 150 + x, 25 + y, 20 + x, 125 + y, 280 + x, 125 + y );
}
```

```
void setup()
{
  size( 600, 350 );
  background( 255 );

  drawHouseAt( 0, 0 );
  drawHouseAt( 310, -10 );
}
```

```
float global_x = 0.0;
float global_y = 0.0;

void myRect( float x, float y, float w, float h )
{
  rect( x + global_x, y + global_y, w, h );
}

void myTriangle(
  float ax, float ay,
  float bx, float by,
  float cx, float cy )
{
  triangle(
    ax + global_x, ay + global_y,
    bx + global_x, by + global_y,
    cx + global_x, cy + global_y );
}
```

```
void drawHouse()
{
  fill( 191, 179, 117 );
  myRect( 50, 125, 200, 150 );
  fill( 62, 54, 47 );
  myTriangle( 150, 25, 20, 125, 280, 125 );
}

void setup()
{
  size( 600, 350 );
  background( 255 );

  global_x = 0;
  global_y = 0;
  drawHouse();

  global_x = 310;
  global_y = -10;
  drawHouse();
}
```

```
void myTranslate( float x, float y )
{
  global_x += x;
  global_y += y;
}

void setup()
{
  size( 600, 350 );
  background( 255 );

  myTranslate( 0, 0 );
  drawHouse();

  myTranslate( 310, -10 );
  drawHouse();
}
```

The built-in functions `translate()`, `rect()` **and** `triangle()` **already do the work of our** `myTranslate()`, `myRect()` **and** `myTriangle()`.

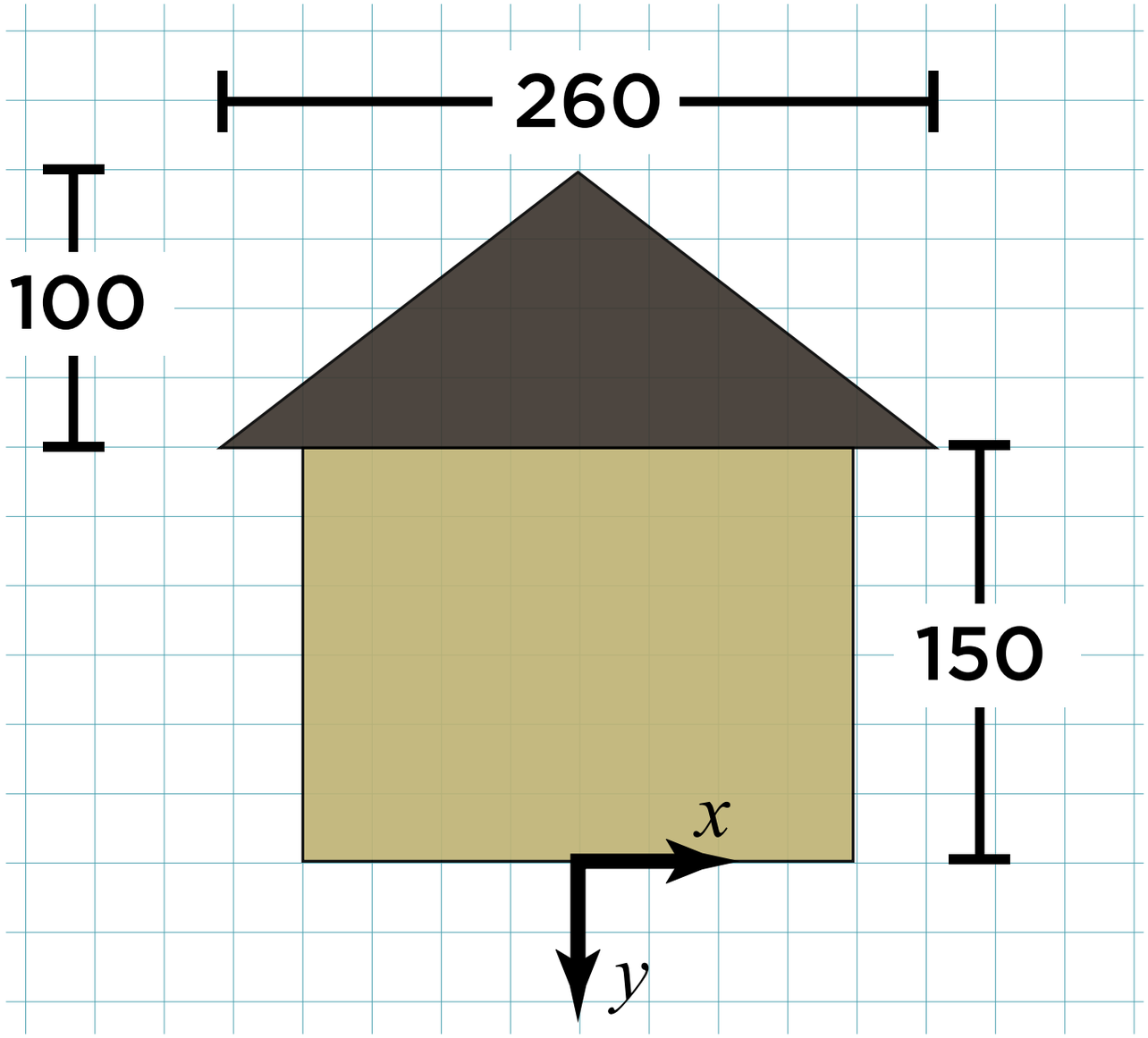
The built-in functions `translate()`, `rect()` **and** `triangle()` **already do the work of our** `myTranslate()`, `myRect()` **and** `myTriangle()`.

The global amount of translation is the “geometric context”.

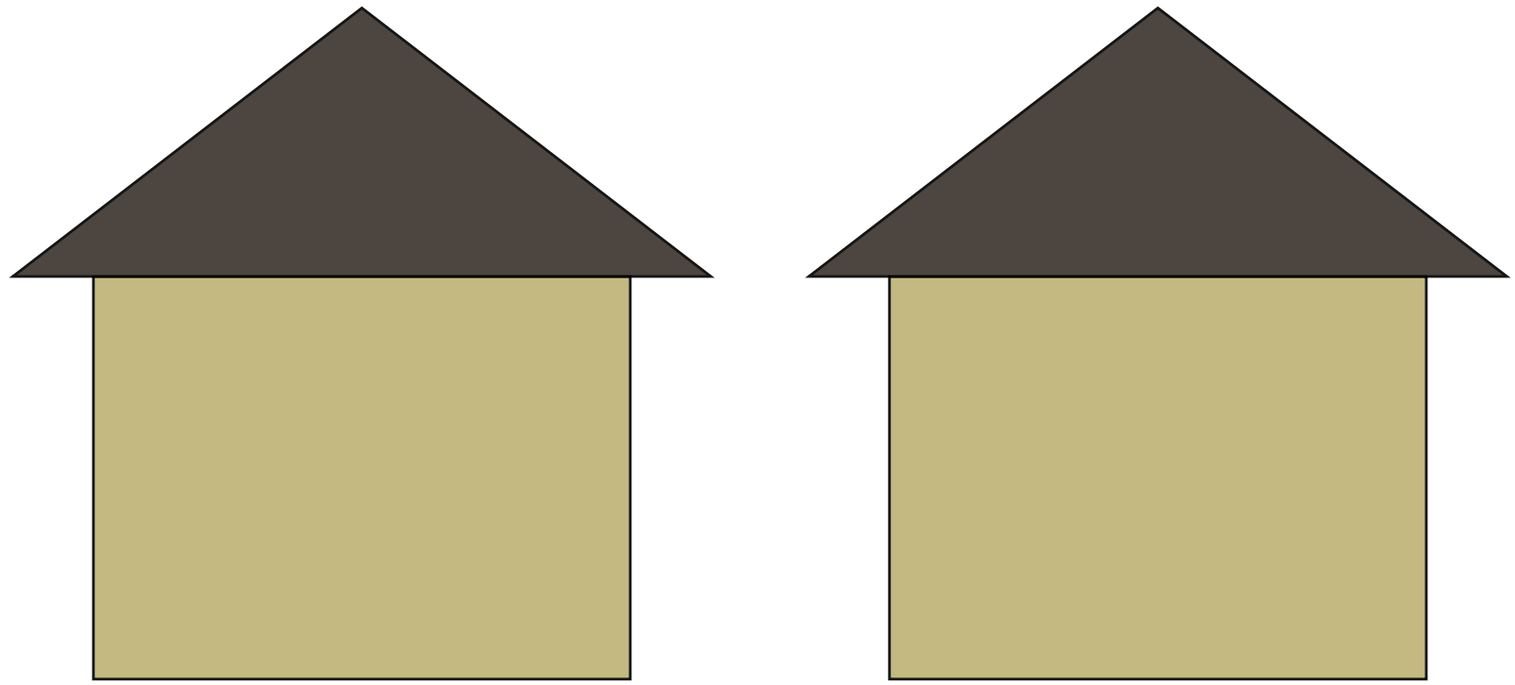
```
void drawHouse()
{
  fill( 191, 179, 117 );
  rect( 50, 125, 200, 150 );
  fill( 62, 54, 47 );
  triangle( 150, 25, 20, 125, 280, 125 );
}

void setup()
{
  size( 600, 350 );
  background( 255 );

  drawHouse();

  translate( 310, -10 );
  drawHouse();
}
```

```
void drawHouse()
{
  fill( 191, 179, 117 );
  rect( 50, 125, 200, 150 );
  fill( 62, 54, 47 );
  triangle( 150, 25, 20, 125, 280, 125 );
}

void setup()
{
  size( 600, 350 );
}

void draw()
{
  background( 255 );
  translate( mouseX, mouseY );
  drawHouse();
}
```

Geometric context allows us to draw any object in its “native coordinate system”.

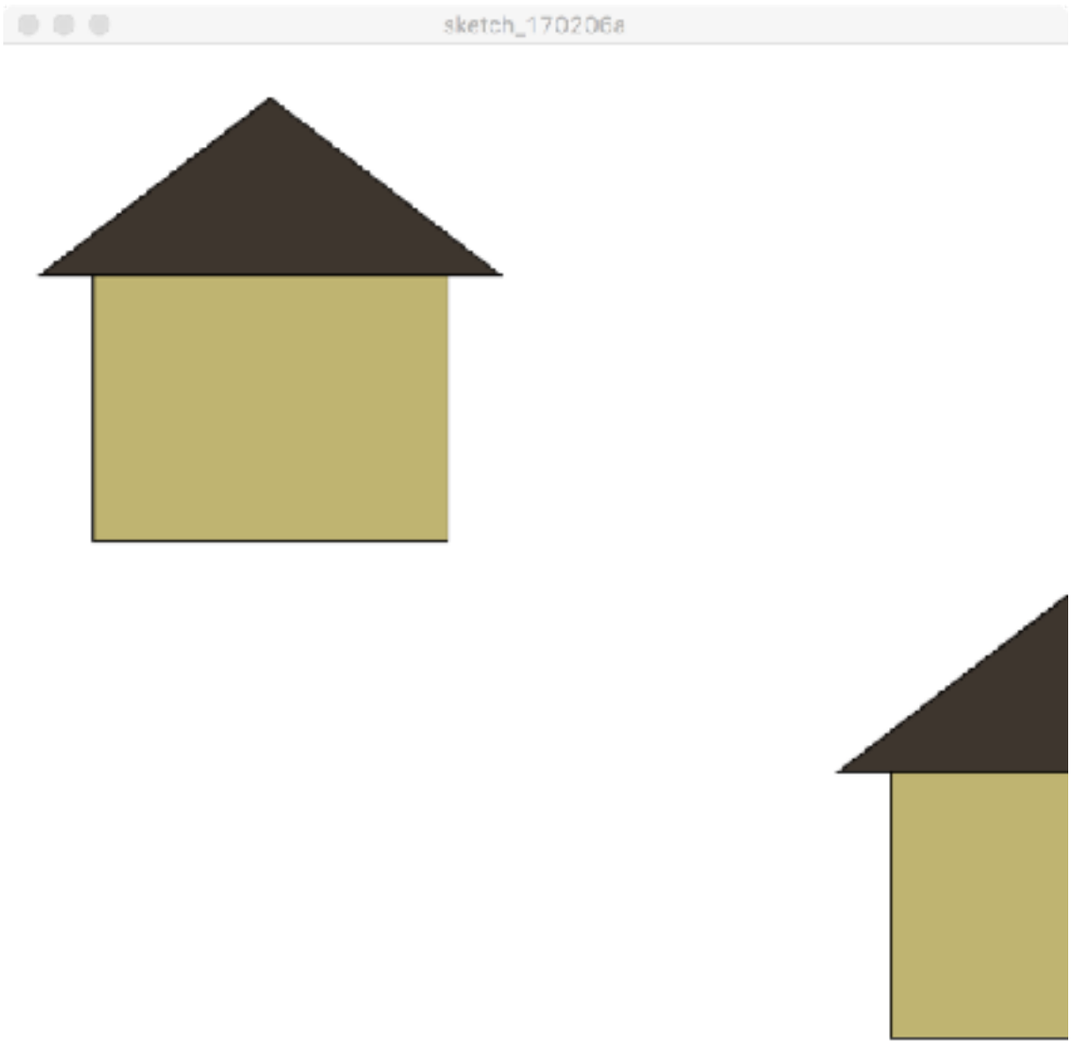



```
void drawHouse()
{
  fill( 191, 179, 117 );
  rect( -100, -150, 200, 150 );
  fill( 62, 54, 47 );
  triangle( -130, -150, 0, -250, 130, -150 );
}
```

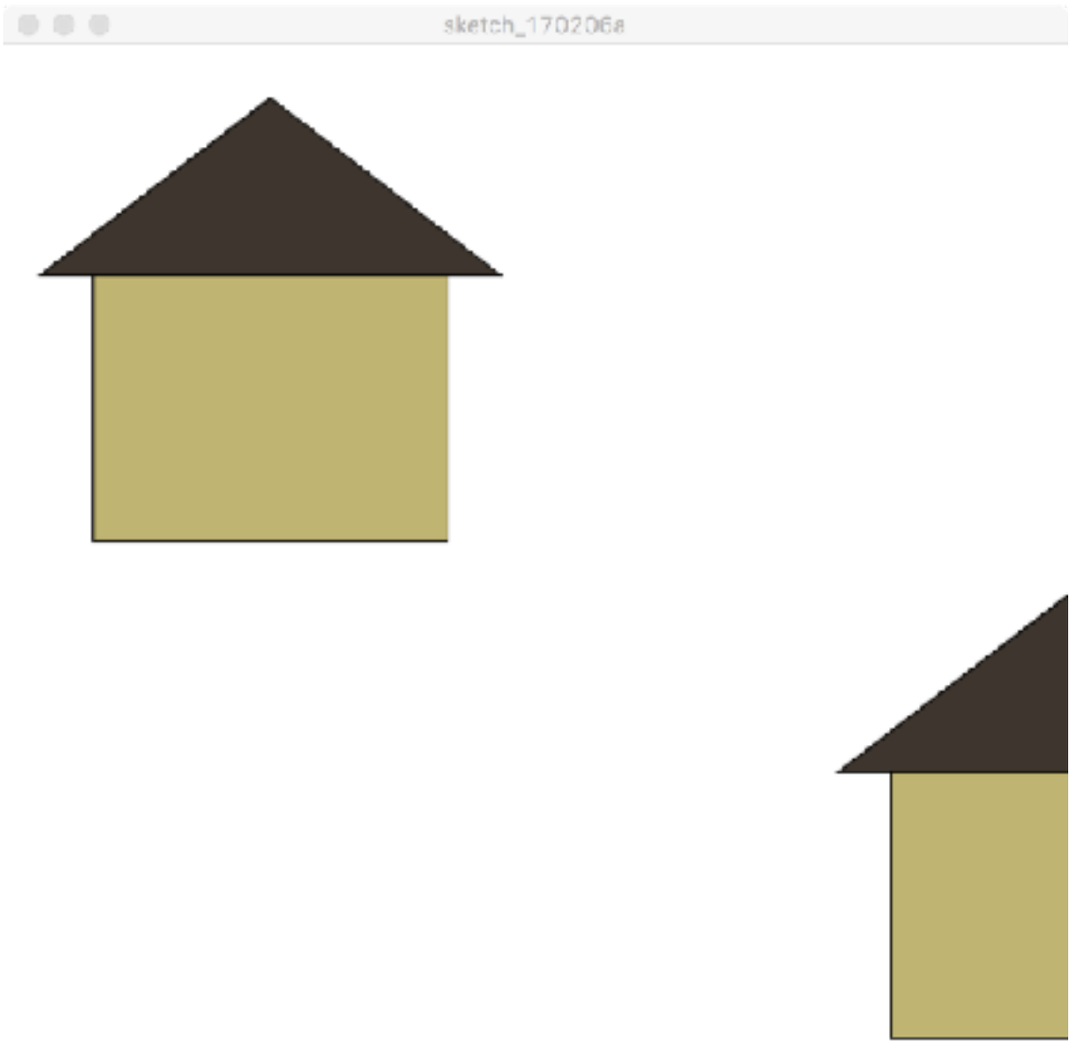


```
void draw()
{
  background( 255 );

  translate( 150, 280 );
  drawHouse();
  translate( 450, 280 );
  drawHouse();
  translate( 150, 580 );
  drawHouse();
  translate( 450, 580 );
  drawHouse();
}
```

```
void draw()
{
  background( 255 );

  translate( 150, 280 );
  drawHouse();
  translate( 450, 280 );
  drawHouse();
  translate( 150, 580 );
  drawHouse();
  translate( 450, 580 );
  drawHouse();
}
```



```
void draw()
{
  background( 255 );

  translate( 150, 280 );
  drawHouse();
  translate( 450, 280 );
  drawHouse();
  translate( 150, 580 );
  drawHouse();
  translate( 450, 580 );
  drawHouse();
}
```



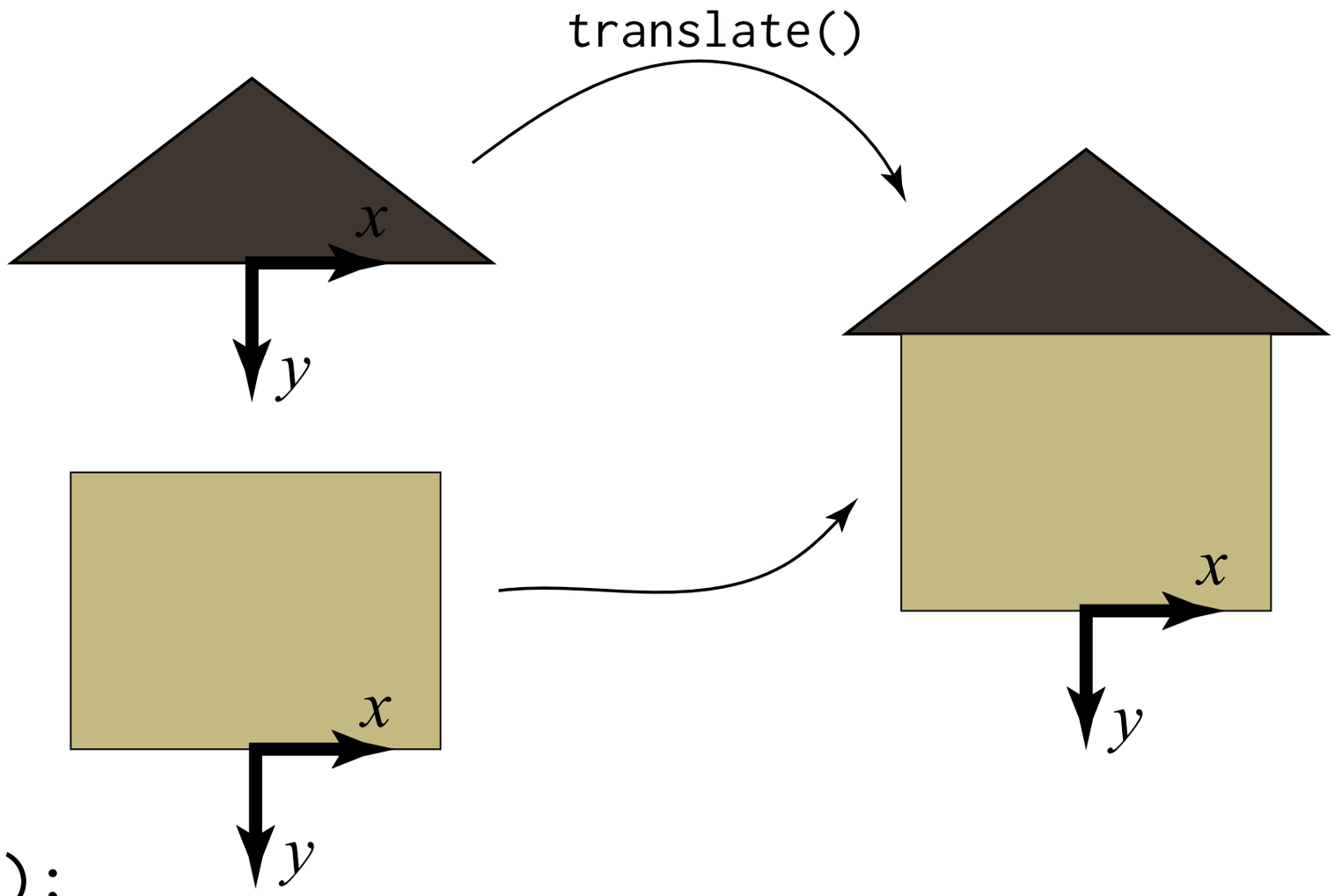
This doesn't work, because transformations *accumulate*.

`pushMatrix()`: **Set a checkpoint, remembering the current geometric context.**

`popMatrix()`: **Go back to the most recently saved context.**

```
void draw()
{
  background( 255 );

  pushMatrix();
  translate( 150, 280 );
  drawHouse();
  popMatrix();

  pushMatrix();
  translate( 450, 280 );
  drawHouse();
  popMatrix();

  pushMatrix();
  translate( 150, 580 );
  drawHouse();
  popMatrix();

  pushMatrix();
  translate( 450, 580 );
  drawHouse();
  popMatrix();
}
```

Draw each house within a temporary context, then throw that context away.



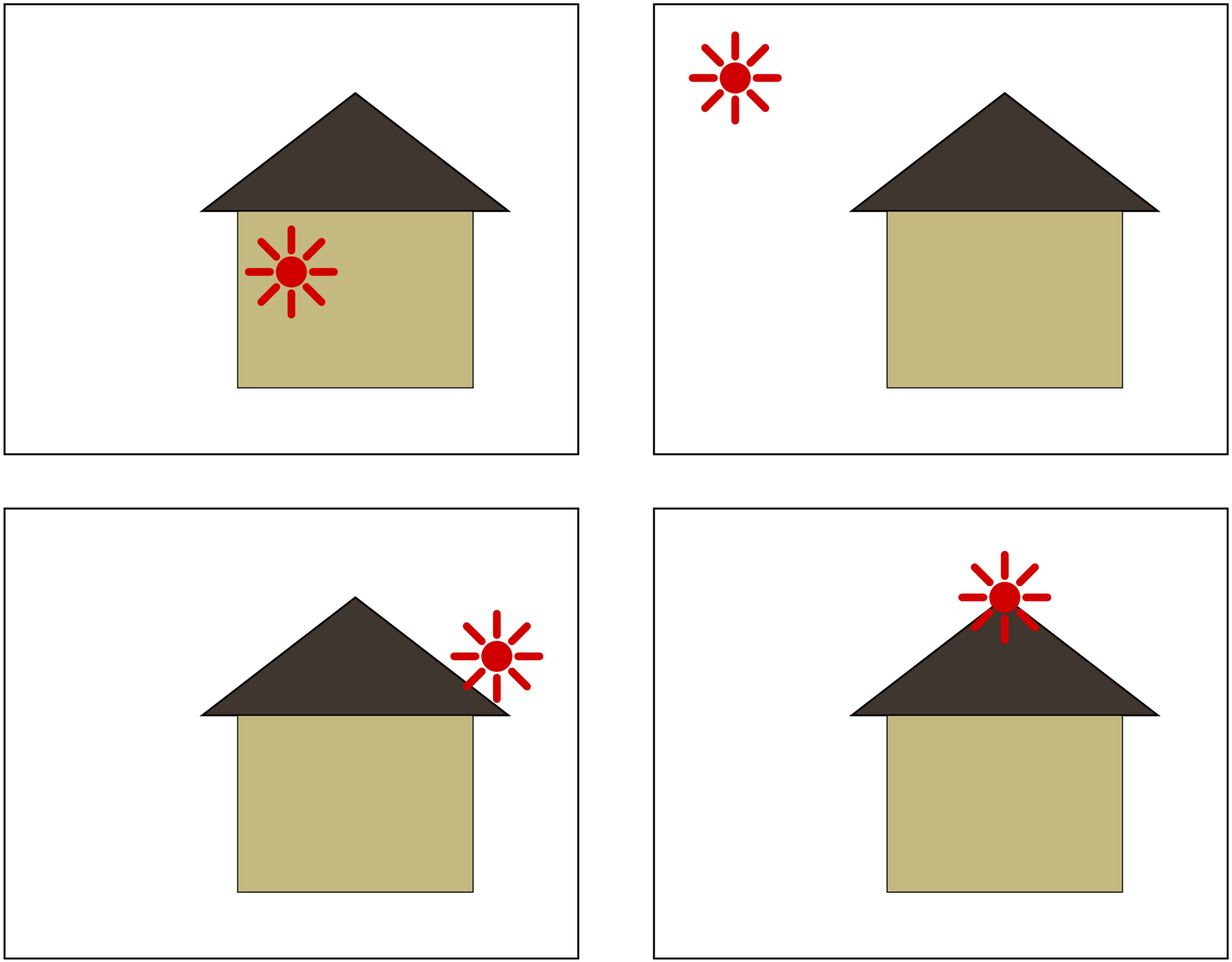

```
void drawHouse()
{
  fill( 191, 179, 117 );
  rect( -100, -150, 200, 150 );
  fill( 62, 54, 47 );

  pushMatrix();
  translate( 0, -150 );
  triangle( -130, 0, 0, -100, 130, 0 );
  popMatrix();
}
```

`rotate( theta )`: **Rotate the current geometric context by some angle** `theta` **(in radians).**

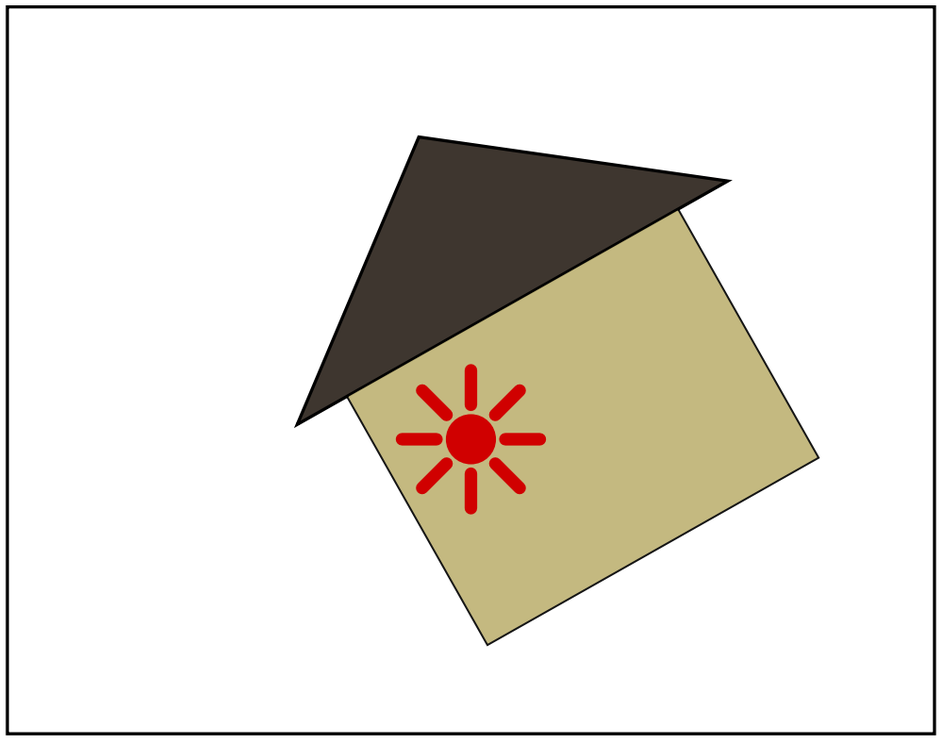
`scale( a, b )`: **Scale the current geometric context by ratios** `a` **in the x direction and** `b` **in the y direction.**

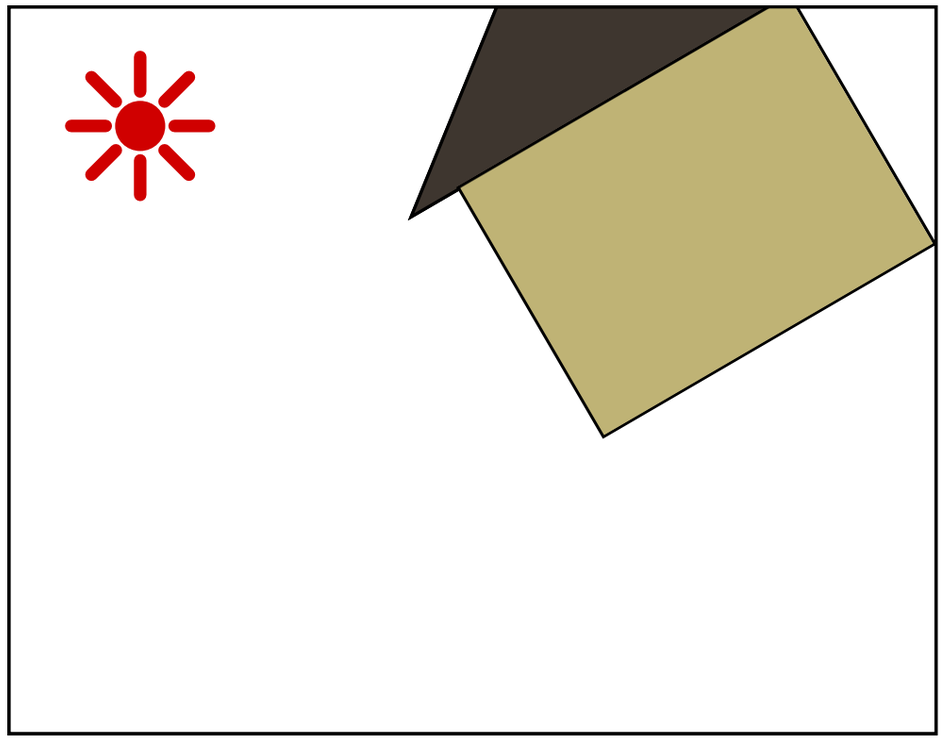
`scale( a )`: **Equivalent to** `scale( a, a )` **(i.e., scale uniformly in x and y).**

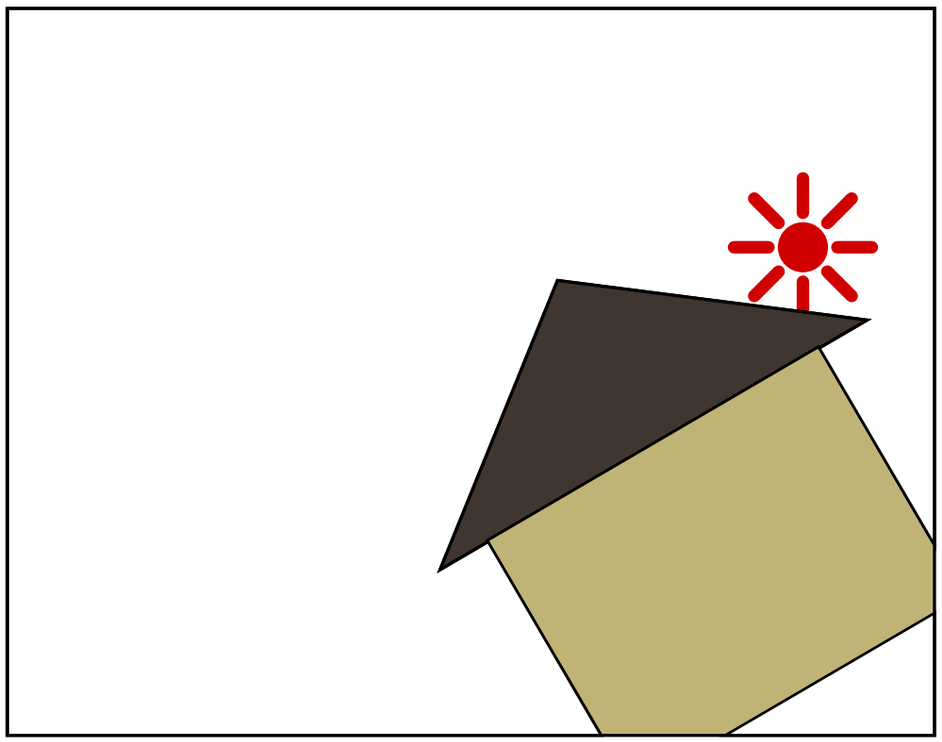
Rotation happens “about a point”.

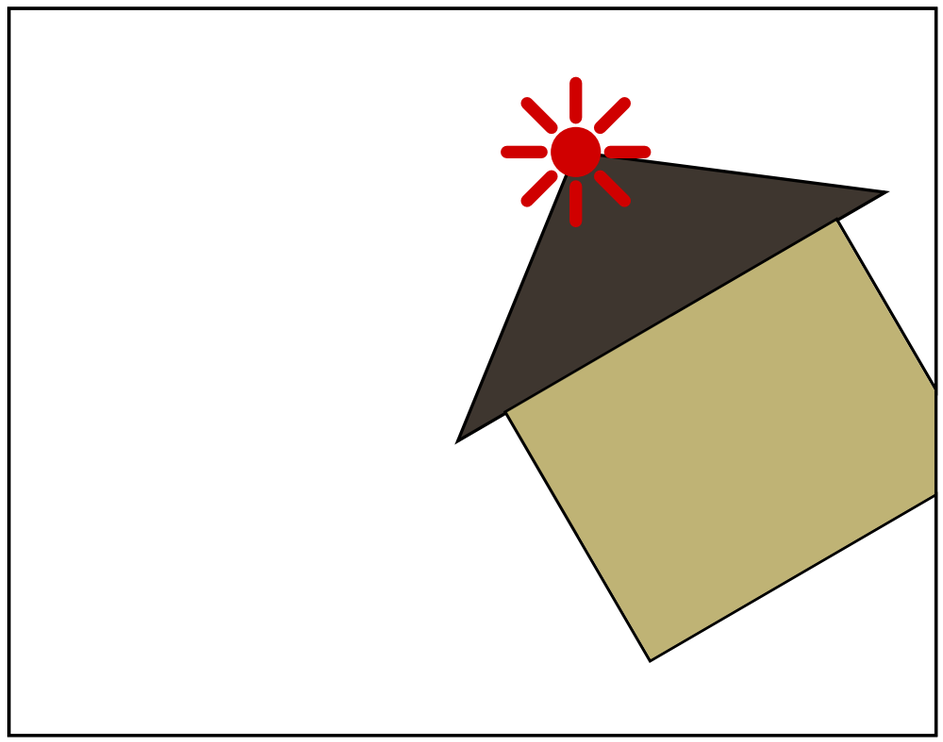


Rotation happens “about a point”.









`rotate( theta )`: **Rotate the current geometric context by some angle `theta` (in radians) *about the origin*.**

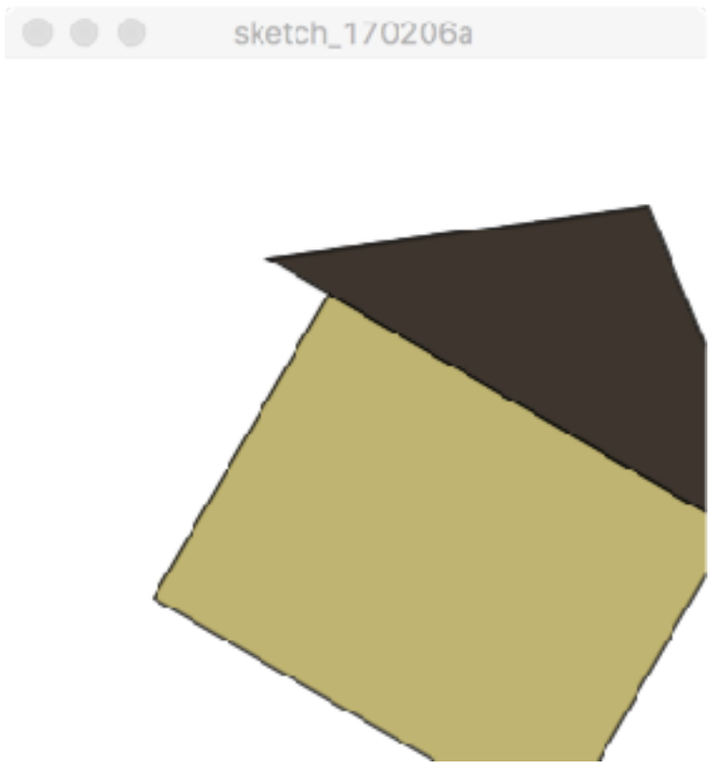
Order matters!!

```
void draw()
{
  background( 255 );

  translate( 150, 280 );
  rotate( radians( 30 ) );
  drawHouse();
}
```

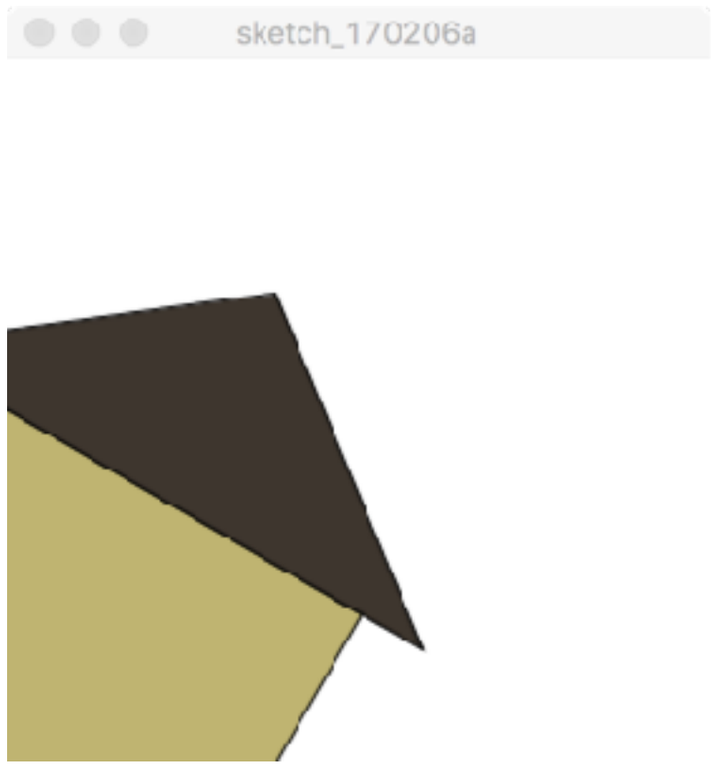
```
void draw()
{
  background( 255 );

  rotate( radians( 30 ) );
  translate( 150, 280 );
  drawHouse();
}
```

Order matters!!

```
void draw()
{
  background( 255 );

  translate( 150, 280 );
  rotate( radians( 30 ) );
  drawHouse();
}
```




```
void draw()
{
  background( 255 );

  rotate( radians( 30 ) );
  translate( 150, 280 );
  drawHouse();
}
```



Understanding order, Version 1

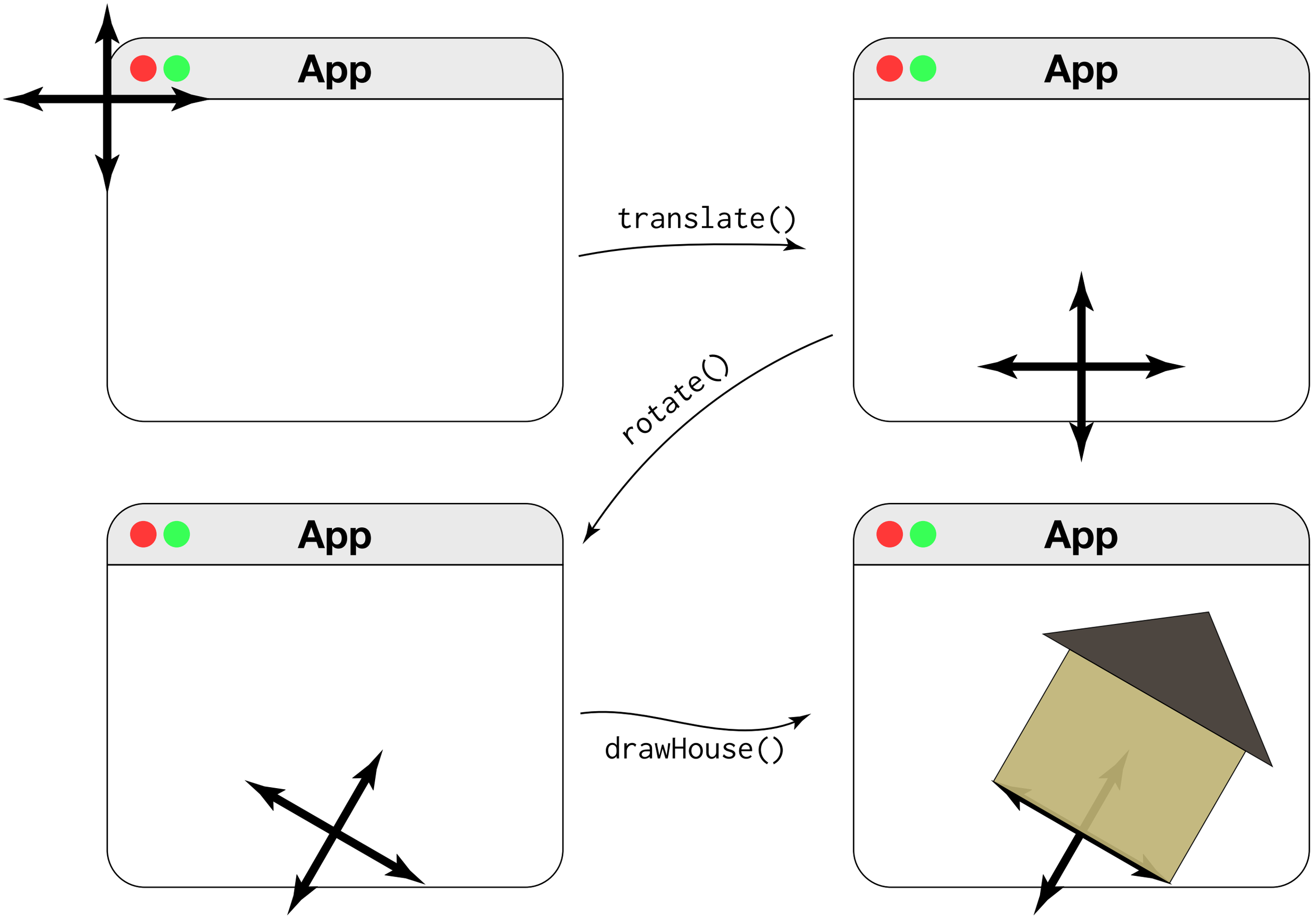
```
void draw()
{
  background( 255 );

  translate( 150, 280 );

}
```

"Whatever happens next, do it in a context that has been translated by (150, 280)."

Understanding order, Version 2

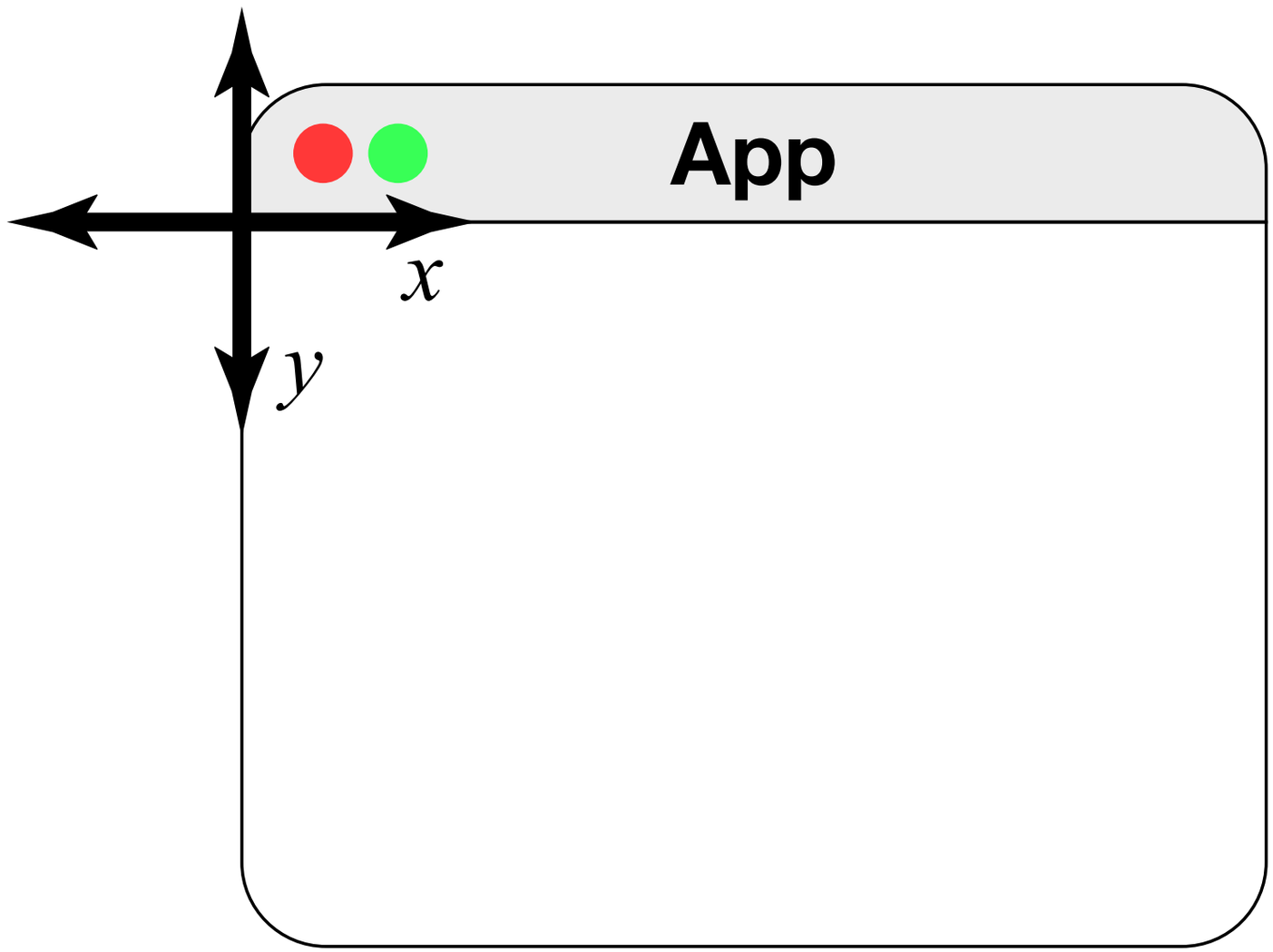
```
void draw()
{
  background( 255 );
  translate( 150, 280 );
}
```

"Translate the actual coordinate axes by this much. Later, draw using these transformed axes."


App
translate()
App
rotate()
App
drawHouse()
App

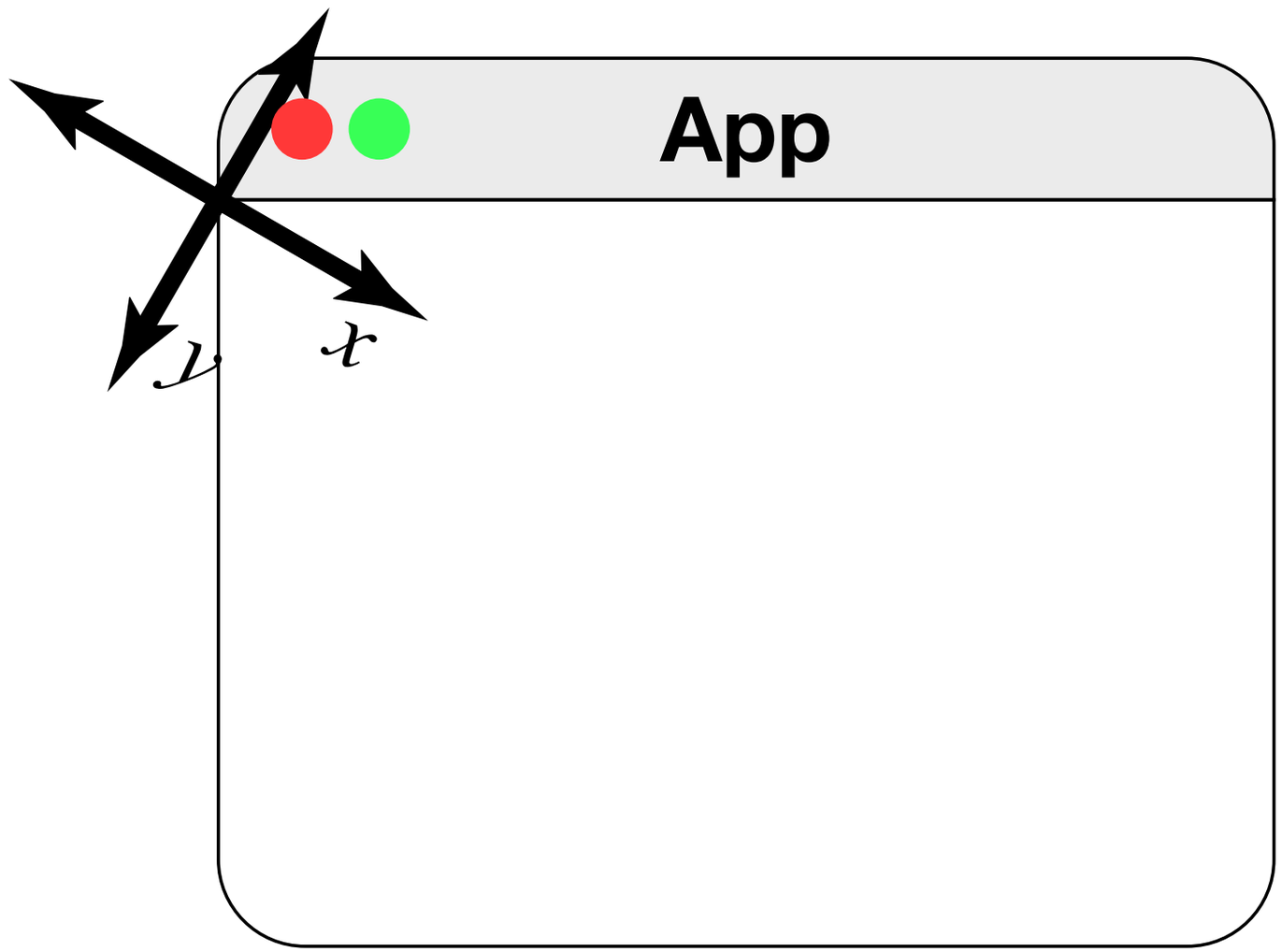
In this second model, each transformation must be applied in the transformed coordinate system that got you there!

```
rotate( radians( 30 ) );
translate( 100, 0 );
rect( 0, 0, 200, 100 );
```



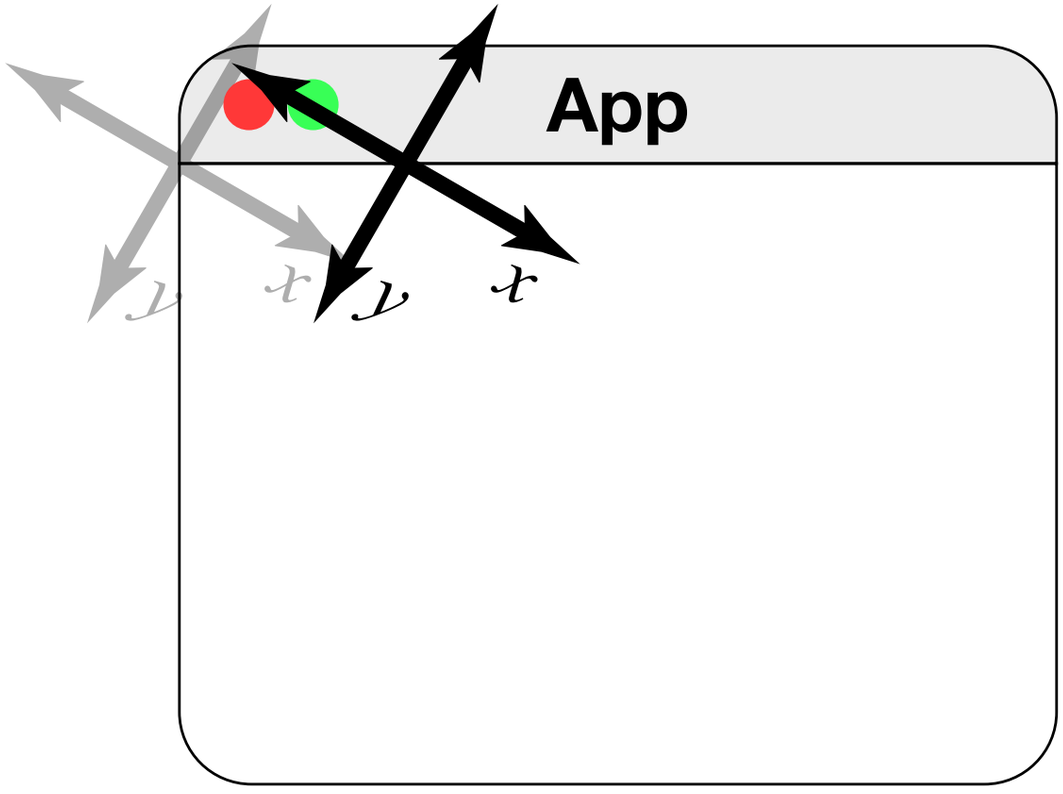
In this second model, each transformation must be applied in the transformed coordinate system that got you there!

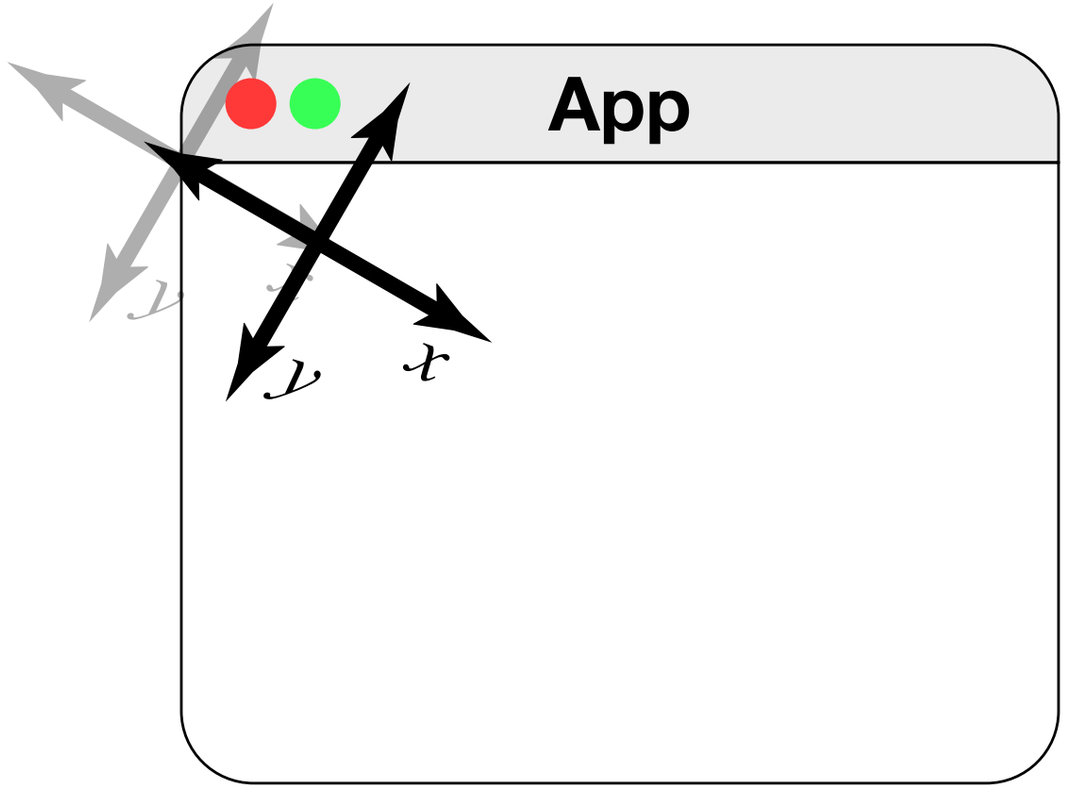
```
rotate( radians( 30 ) );
translate( 100, 0 );
rect( 0, 0, 200, 100 );
```



In this second model, each transformation must be applied **in the transformed coordinate system** that got you there!

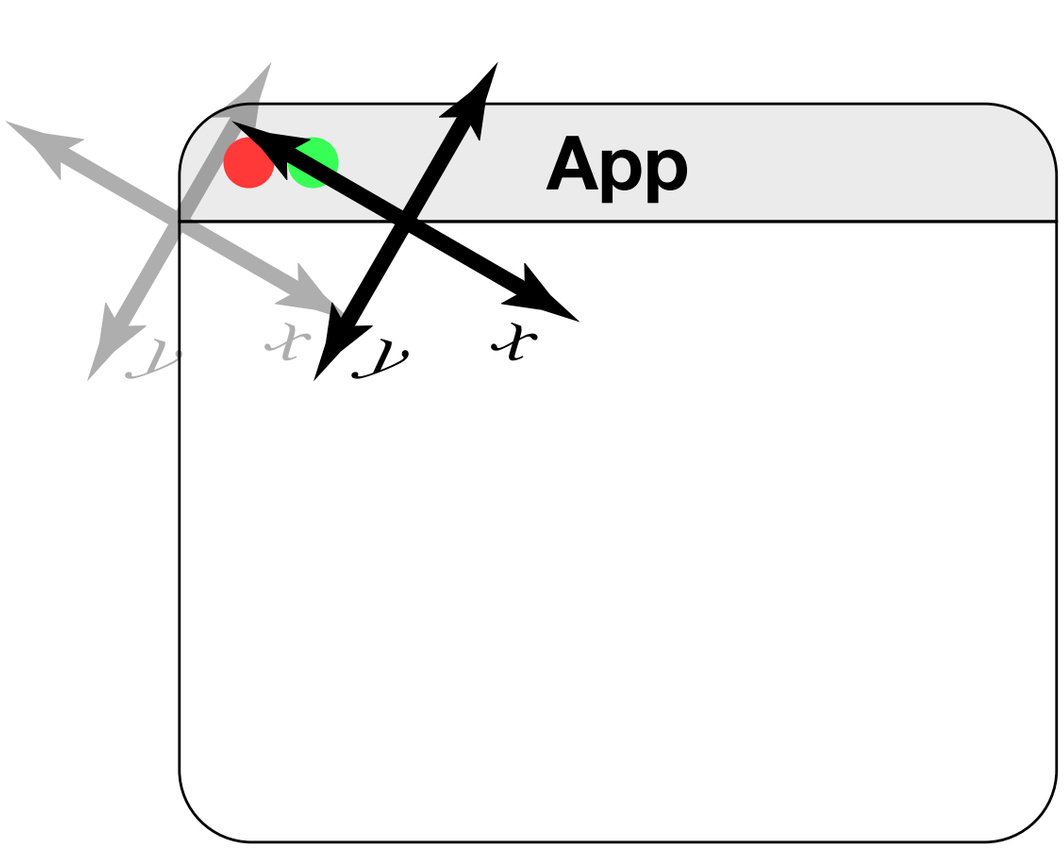
```
rotate( radians( 30 ) );
translate( 100, 0 );
rect( 0, 0, 200, 100 );
```

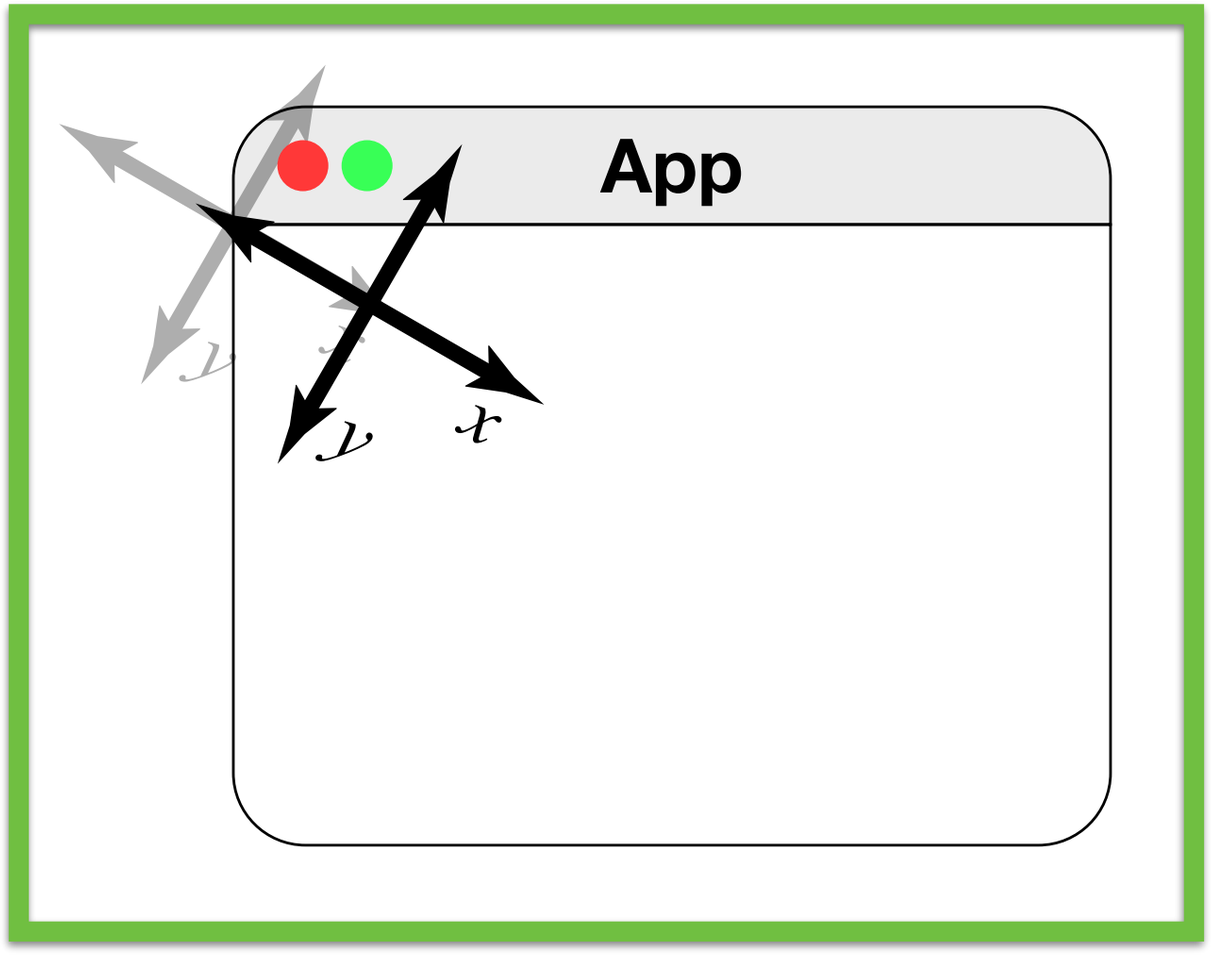




In this second model, each transformation must be applied **in the transformed coordinate system** that got you there!

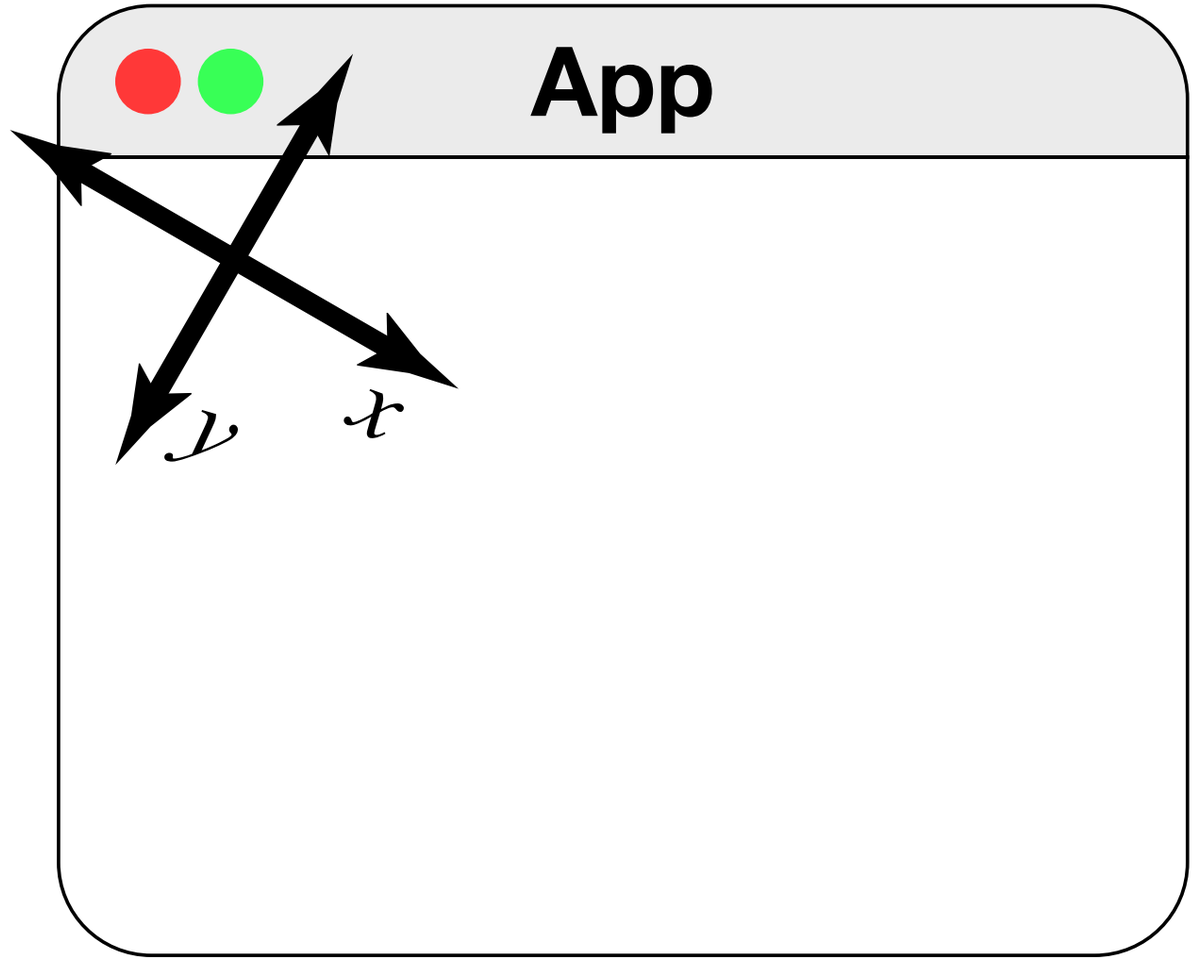
```
rotate( radians( 30 ) );
translate( 100, 0 );
rect( 0, 0, 200, 100 );
```





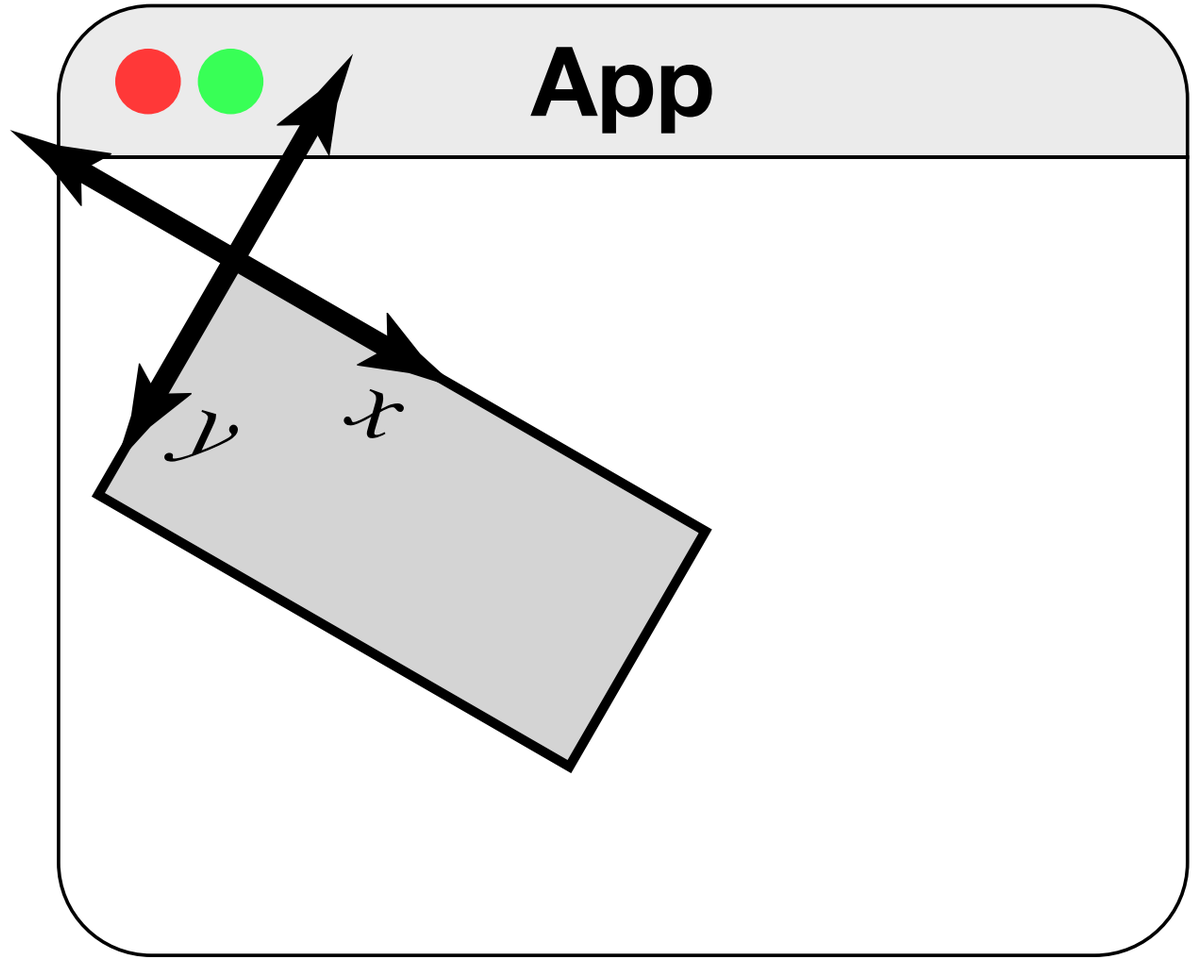
In this second model, each transformation must be applied in the transformed coordinate system that got you there!

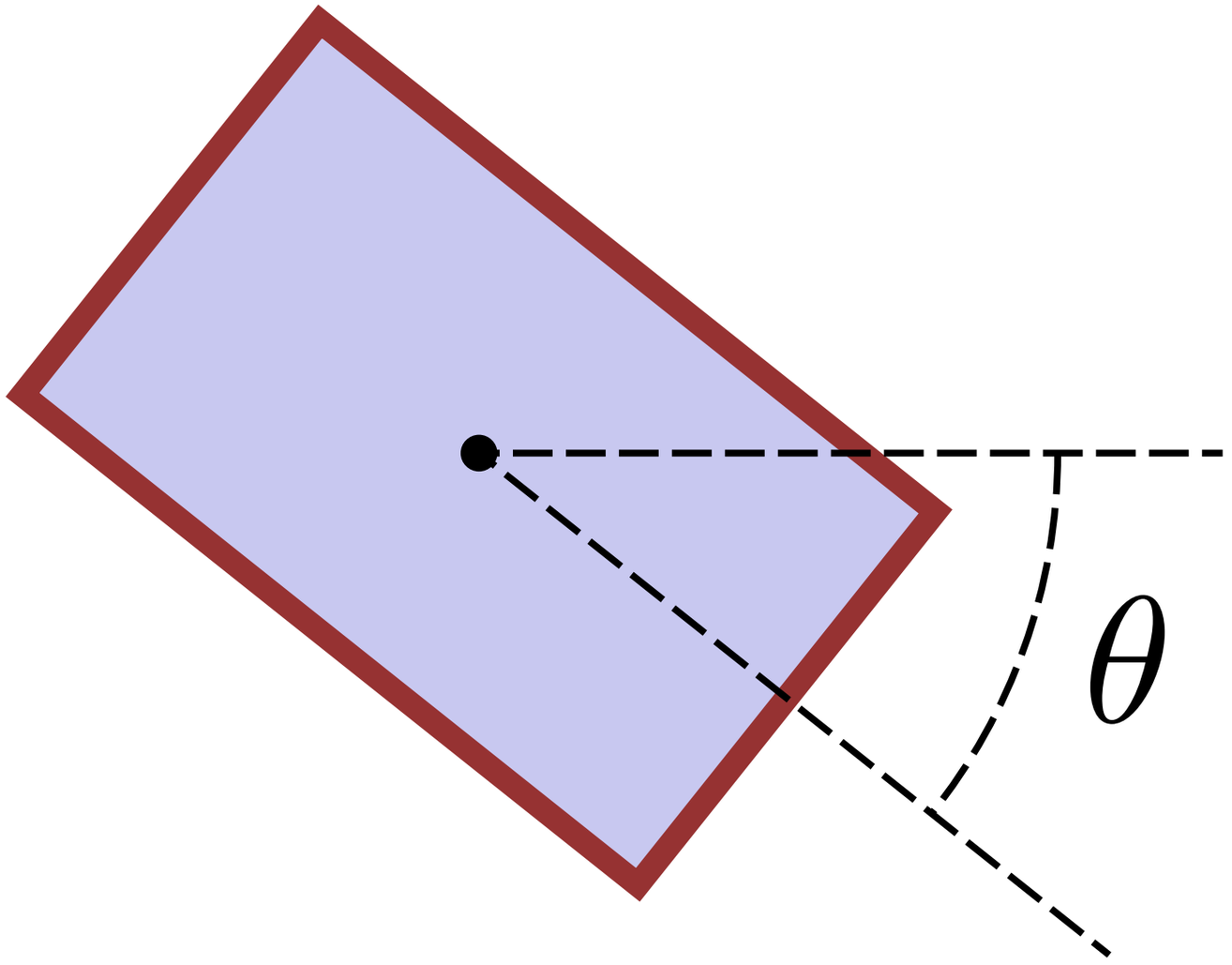
```
rotate( radians( 30 ) );
translate( 100, 0 );
rect( 0, 0, 200, 100 );
```

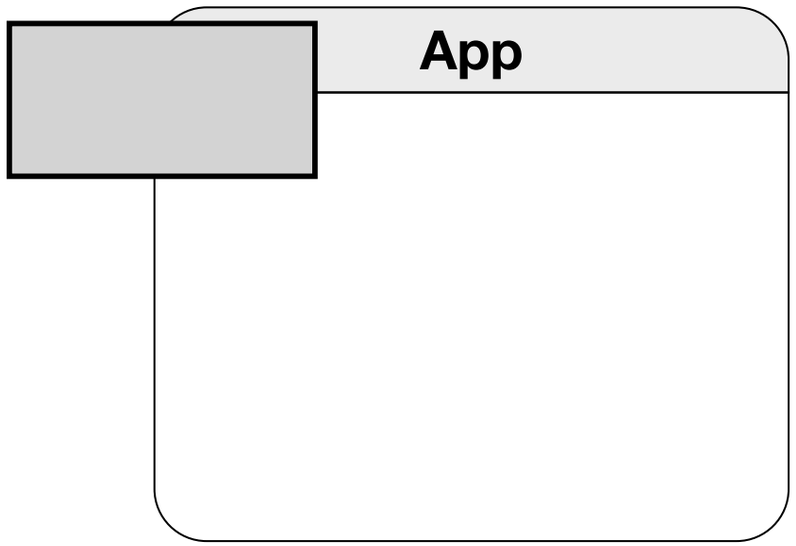


In this second model, each transformation must be applied in the transformed coordinate system that got you there!

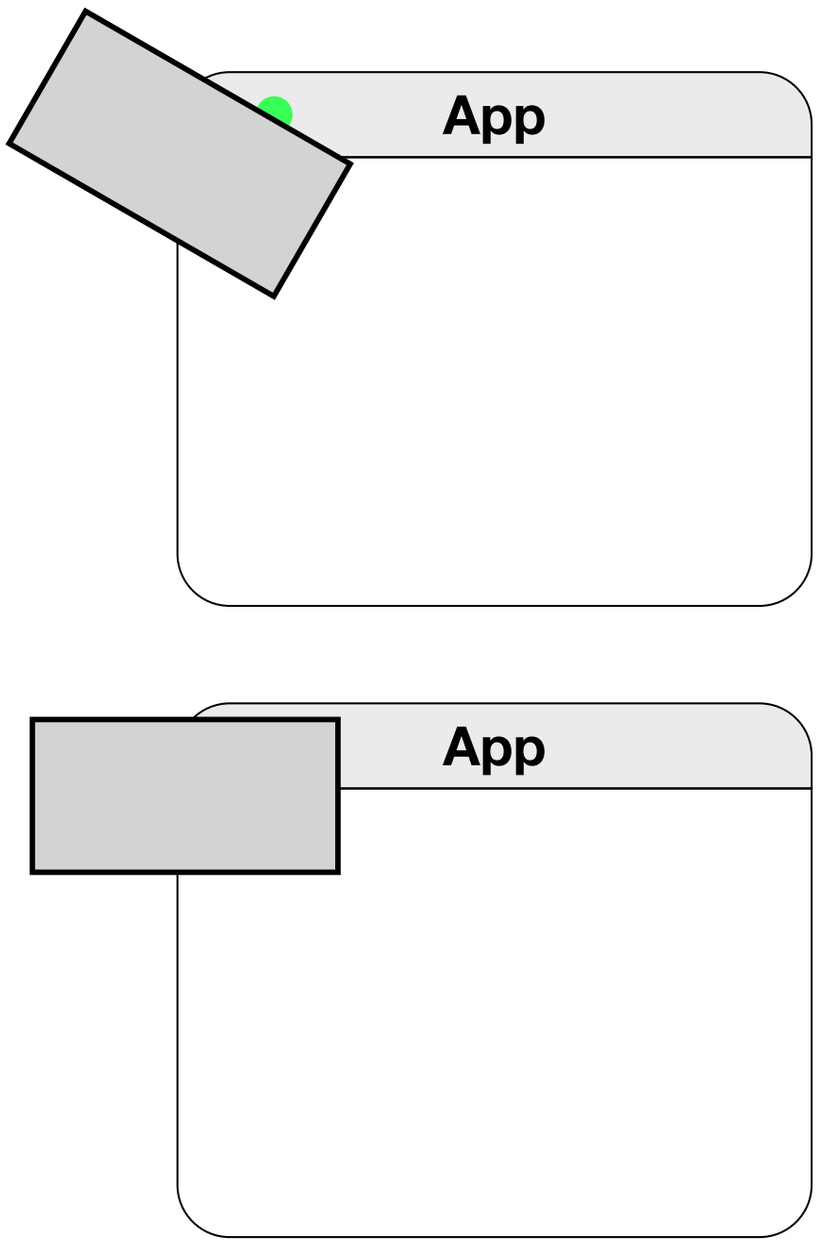
```
rotate( radians( 30 ) );
translate( 100, 0 );
rect( 0, 0, 200, 100 );
```




θ

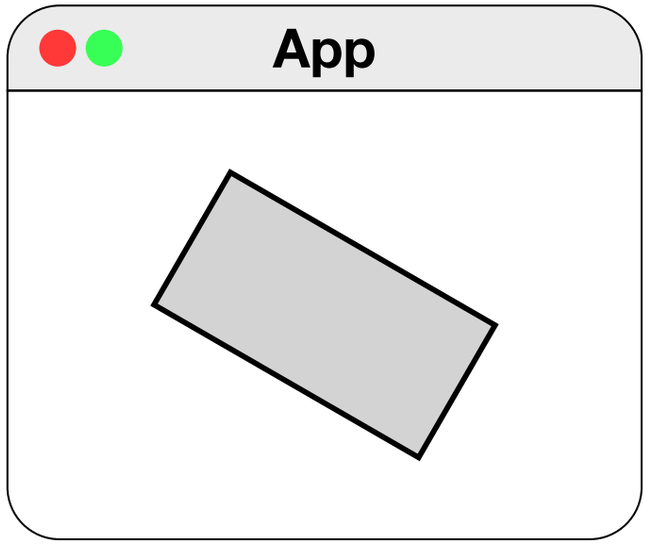


1. Draw a rectangle centred on (0,0)

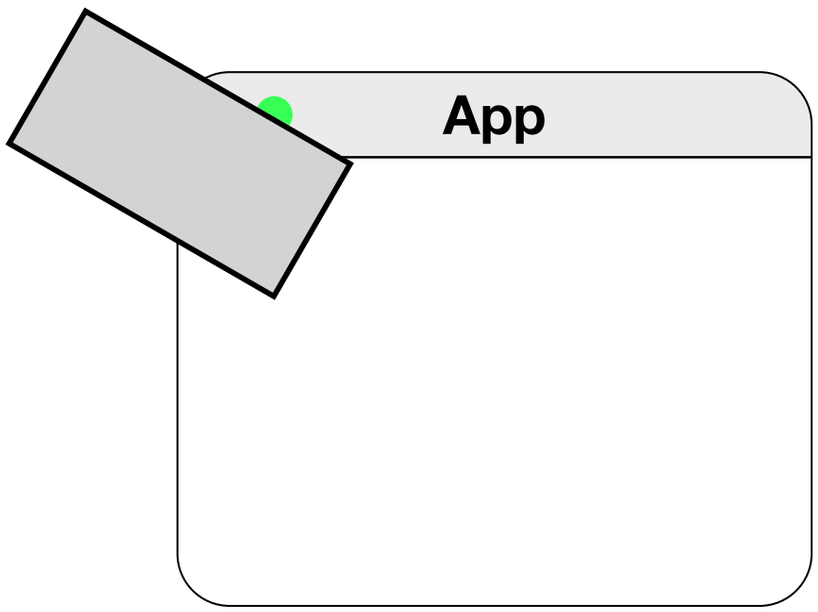


2. Rotate it

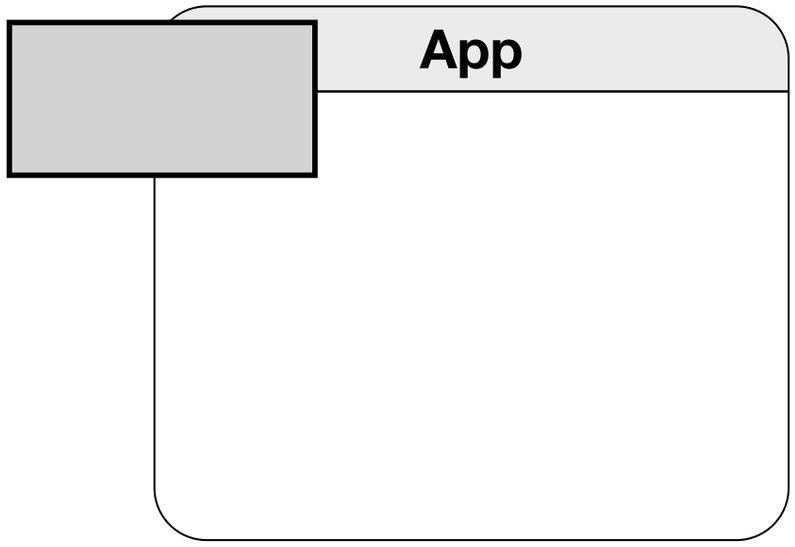
1. Draw a rectangle centred on (0,0)



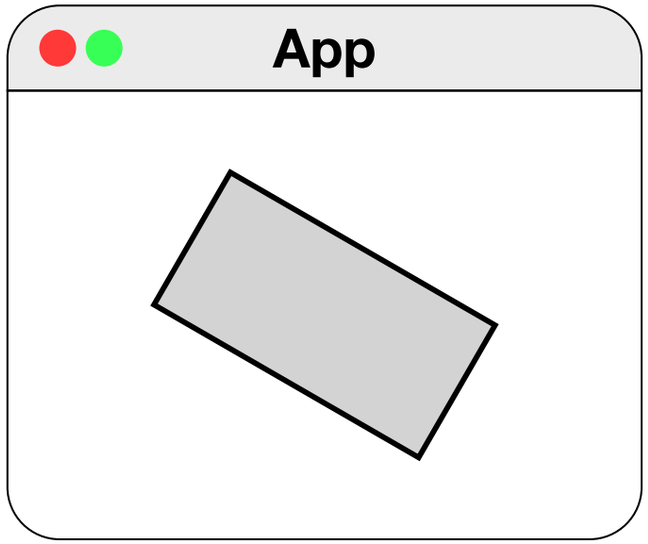
3. Translate (0,0) to the centre of the sketch



2. Rotate it

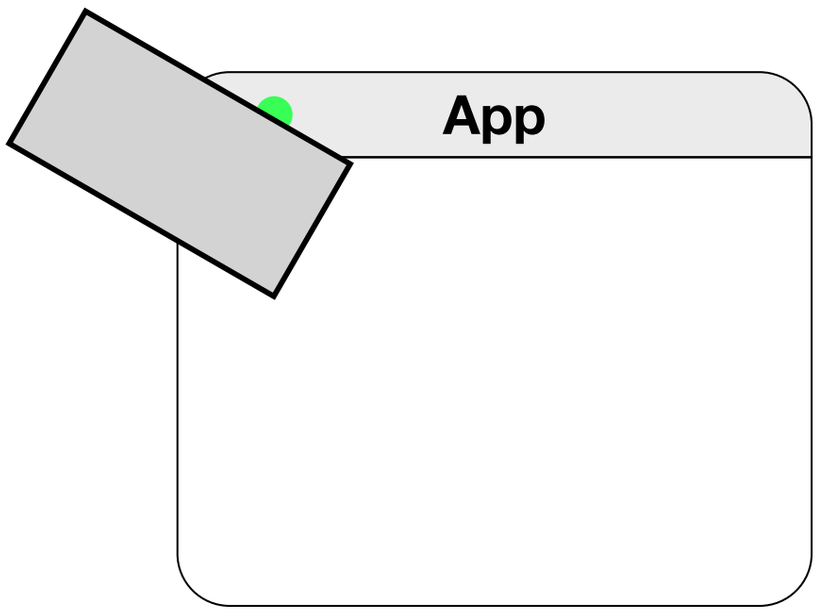


1. Draw a rectangle centred on (0,0)

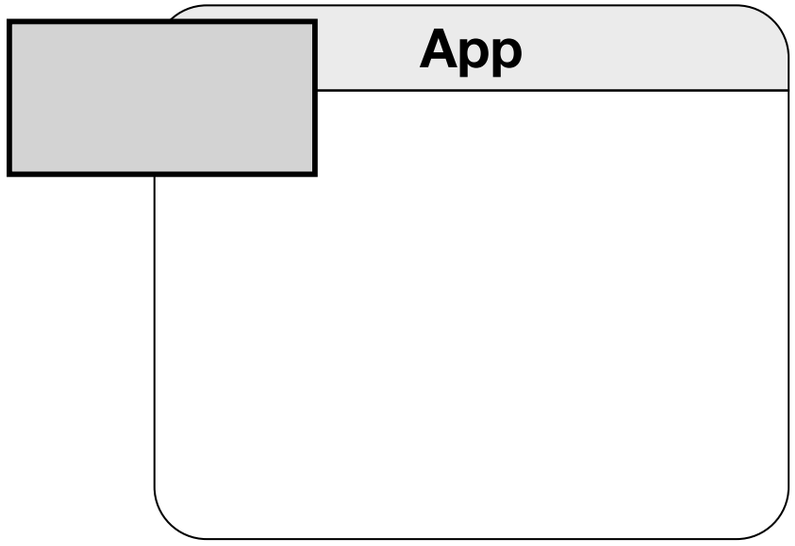


3. Translate (0,0) to the centre of the sketch

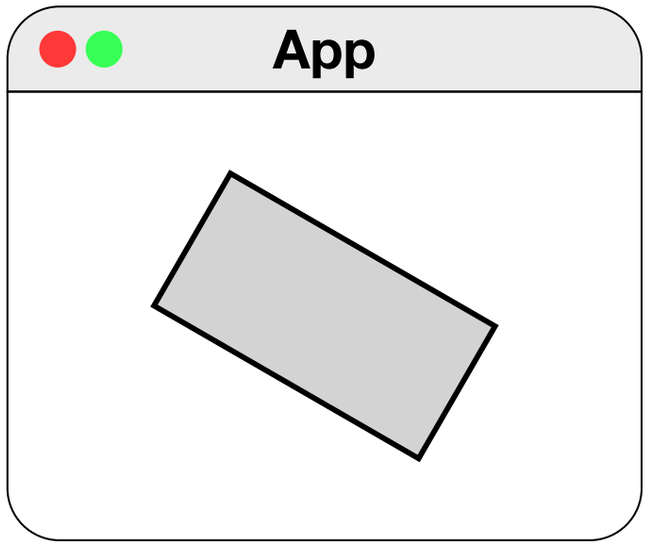
```
translate( width/2, height/2 );
```



2. Rotate it

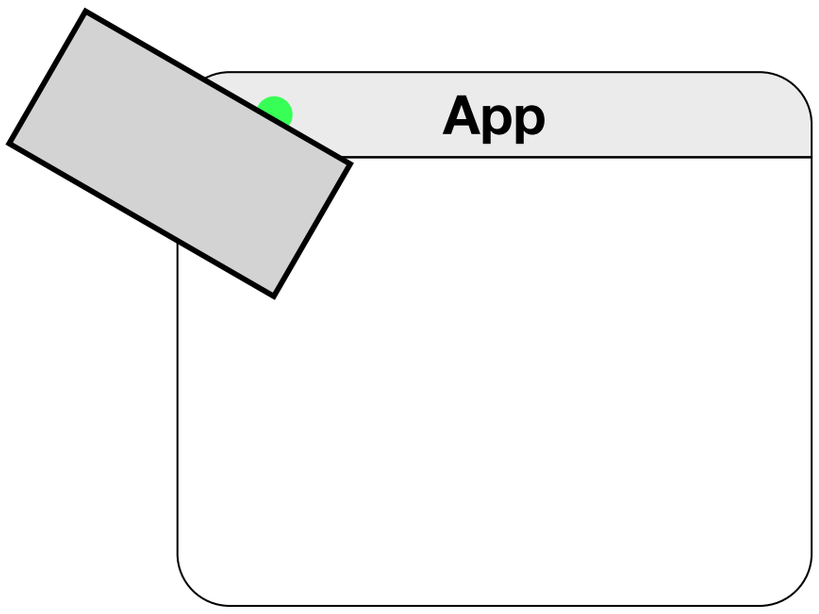


1. Draw a rectangle centred on (0,0)



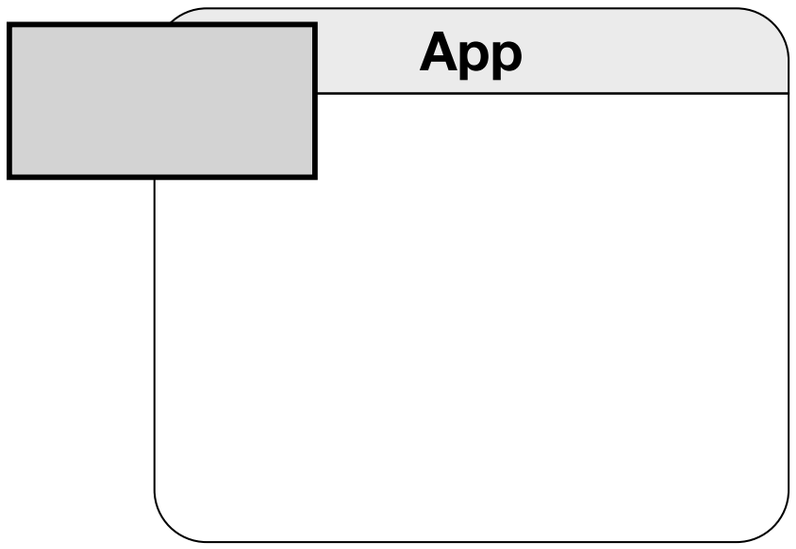
3. Translate (0,0) to the centre of the sketch

```
translate( width/2, height/2 );
```

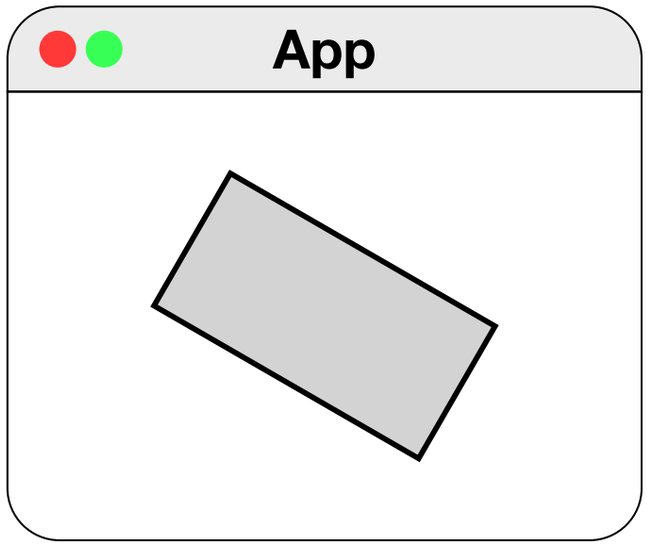


2. Rotate it

```
rotate( radians( 30 ) );
```

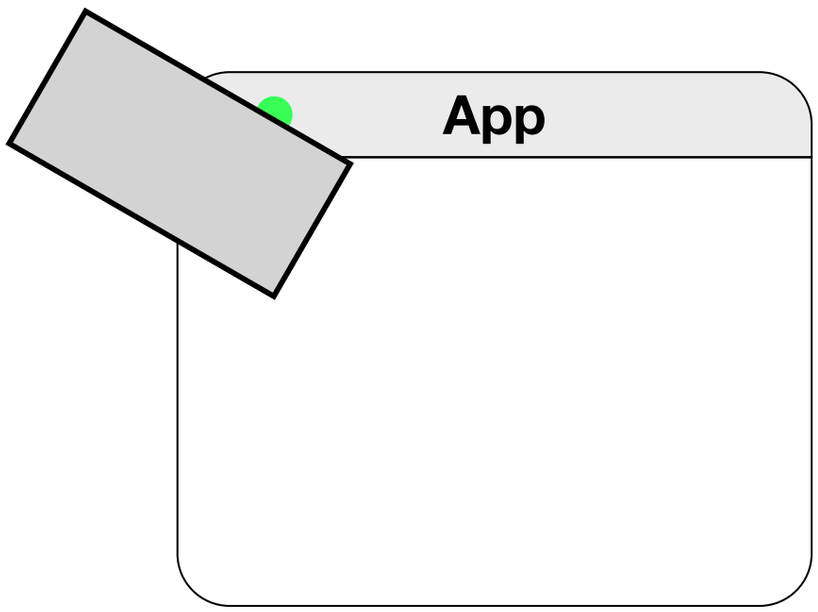


1. Draw a rectangle centred on (0,0)



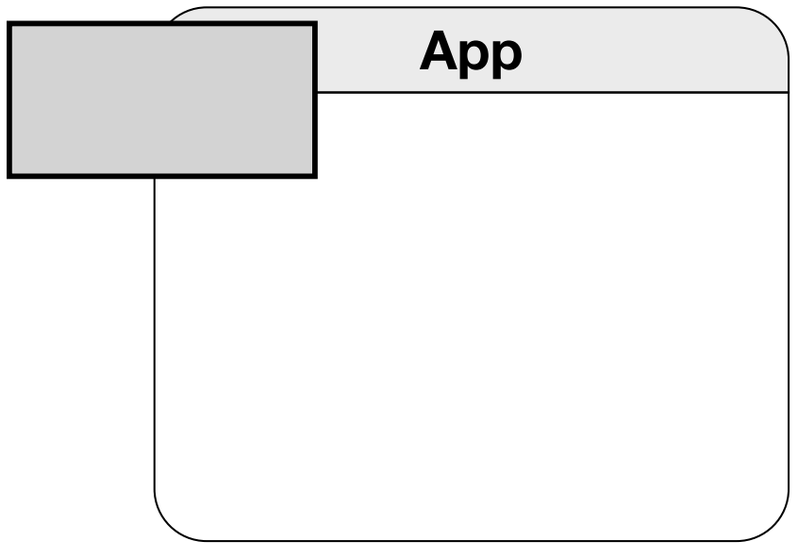
3. Translate (0,0) to the centre of the sketch

```
translate( width/2, height/2 );
```



2. Rotate it

```
rotate( radians( 30 ) );
```



1. Draw a rectangle centred on (0,0)

```
rect( -100, -50, 200, 100 );
```

Complex transformations

Sometimes it's easiest to build up a complicated transformation from smaller steps.

Example: rotating about an arbitrary point (x,y) by an angle `theta`.

Complex transformations

Sometimes it's easiest to build up a complicated transformation from smaller steps.

Example: rotating about an arbitrary point (x,y) by an angle `theta`.

1. Translate (x,y) to the origin

Complex transformations

Sometimes it's easiest to build up a complicated transformation from smaller steps.

Example: rotating about an arbitrary point (x,y) by an angle `theta`.

2. Rotate

1. Translate (x,y) to the origin

Complex transformations

Sometimes it's easiest to build up a complicated transformation from smaller steps.

Example: rotating about an arbitrary point (x,y) by an angle `theta`.

3. Translate the origin back to (x,y)

2. Rotate

1. Translate (x,y) to the origin

Complex transformations

Sometimes it's easiest to build up a complicated transformation from smaller steps.

Example: rotating about an arbitrary point (x,y) by an angle theta.

3. Translate the origin back to (x,y)
2. Rotate
1. Translate (x,y) to the origin

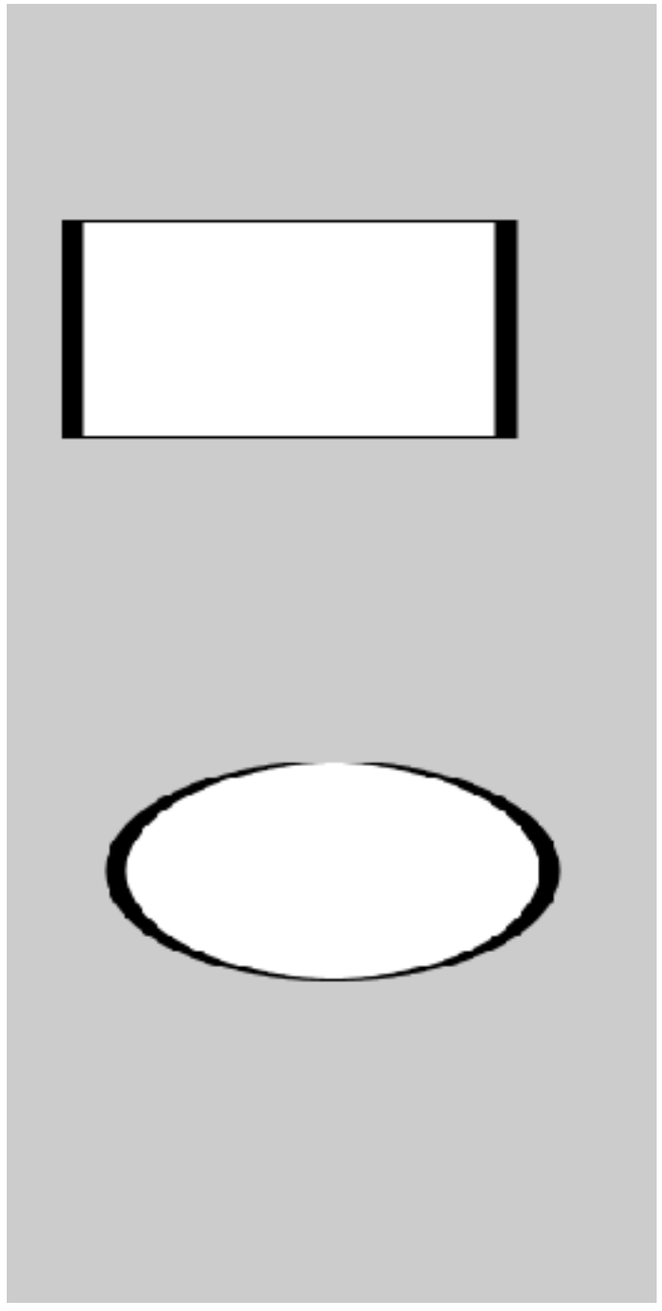
```
void rotateAboutPoint(
  float x, float y, float theta )
{
  translate( x, y );

  rotate( radians( theta ) );

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

`scale( a, b )`: **Scale the current geometric context by some factors** `a` **and** `b` ***about the origin***.

```
size( 200, 200 );
scale( a, b );
rect( 50, 50, 50, 50 );
```

Beware: scaling affects strokes too!

```
void setup()
{
  size( 300, 600 );
  scale( 10, 1 );
  rect( 3, 100, 20, 100 );
  ellipse( 15, 400, 20, 100 );
  save( "output.png" );
}
```



Complex transformations

```
void rotateAboutPoint(
  float x, float y, float theta )
{
  translate( x, y );
  rotate( radians( theta ) );
  translate( -x, -y );
}
```

Complex transformations

```
void rotateAboutPoint(
  float x, float y, float theta )
{
  translate( x, y );
  rotate( radians( theta ) );
  translate( -x, -y );
}
```

```
void scaleAboutPoint(
  float x, float y, float s )
{
  translate( x, y );
  scale( s );
  translate( -x, -y );
}
```

TRS

When in doubt, try the order TRS (Translate, Rotate, Scale) in your code.

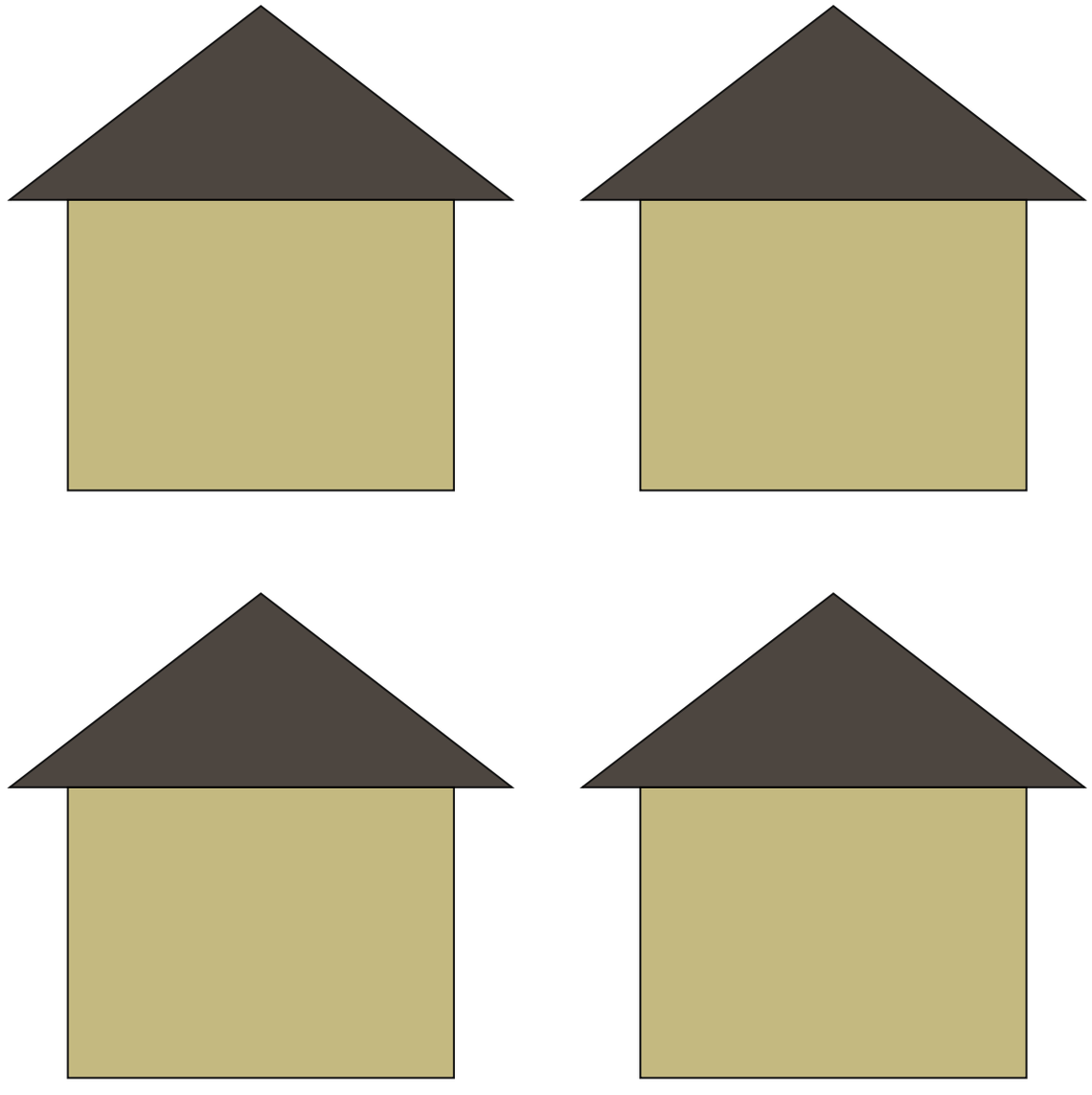
```
translate( x, y );
rotate( theta );
scale( sx, sy );
// Draw stuff here.
```

Context affects everything!

Geometric context affects everything drawn on the screen: shapes, images, illustrations, text, ...

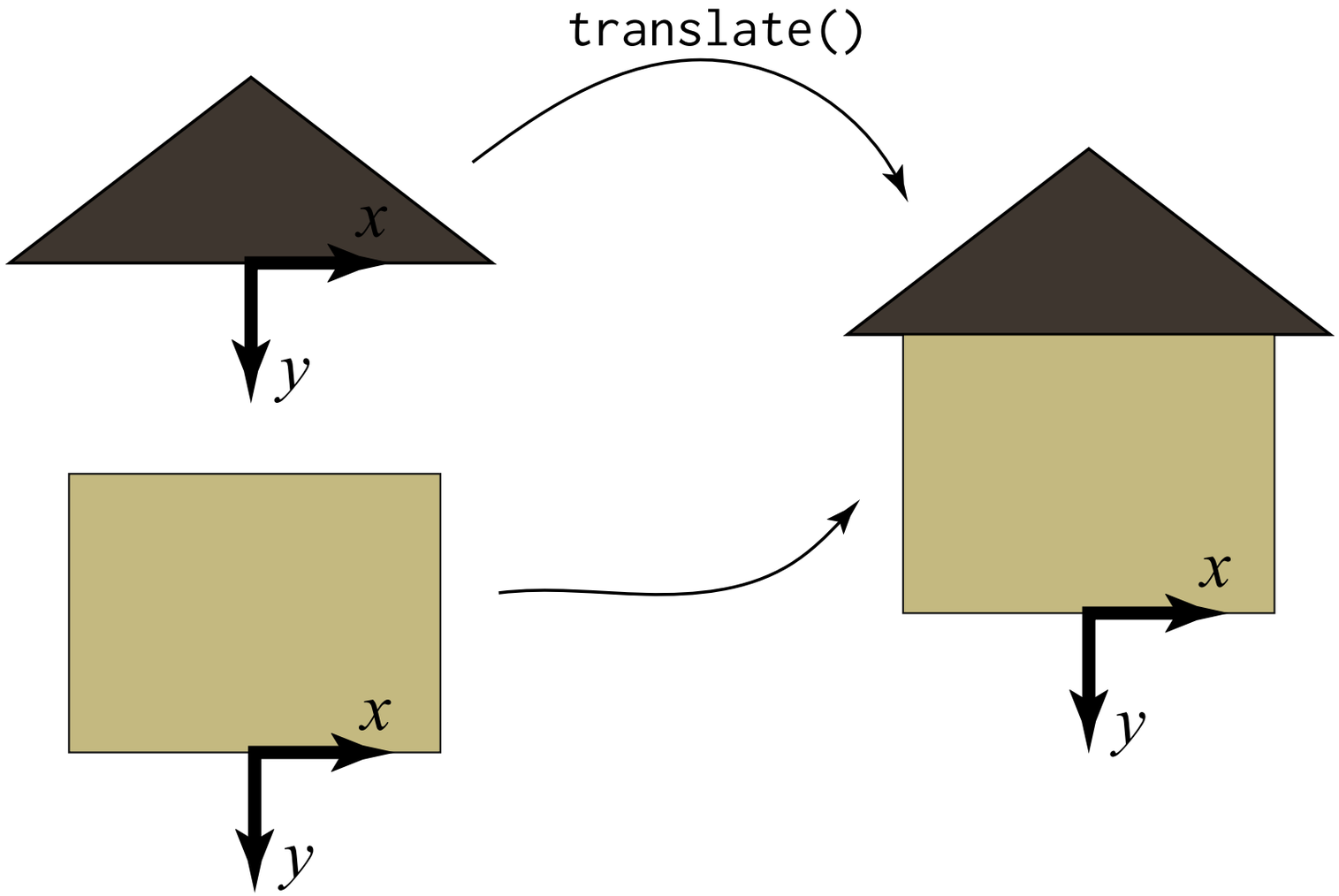
Hierarchical Modelling

With geometric context, we can define functions that express “reusable components” in drawings.



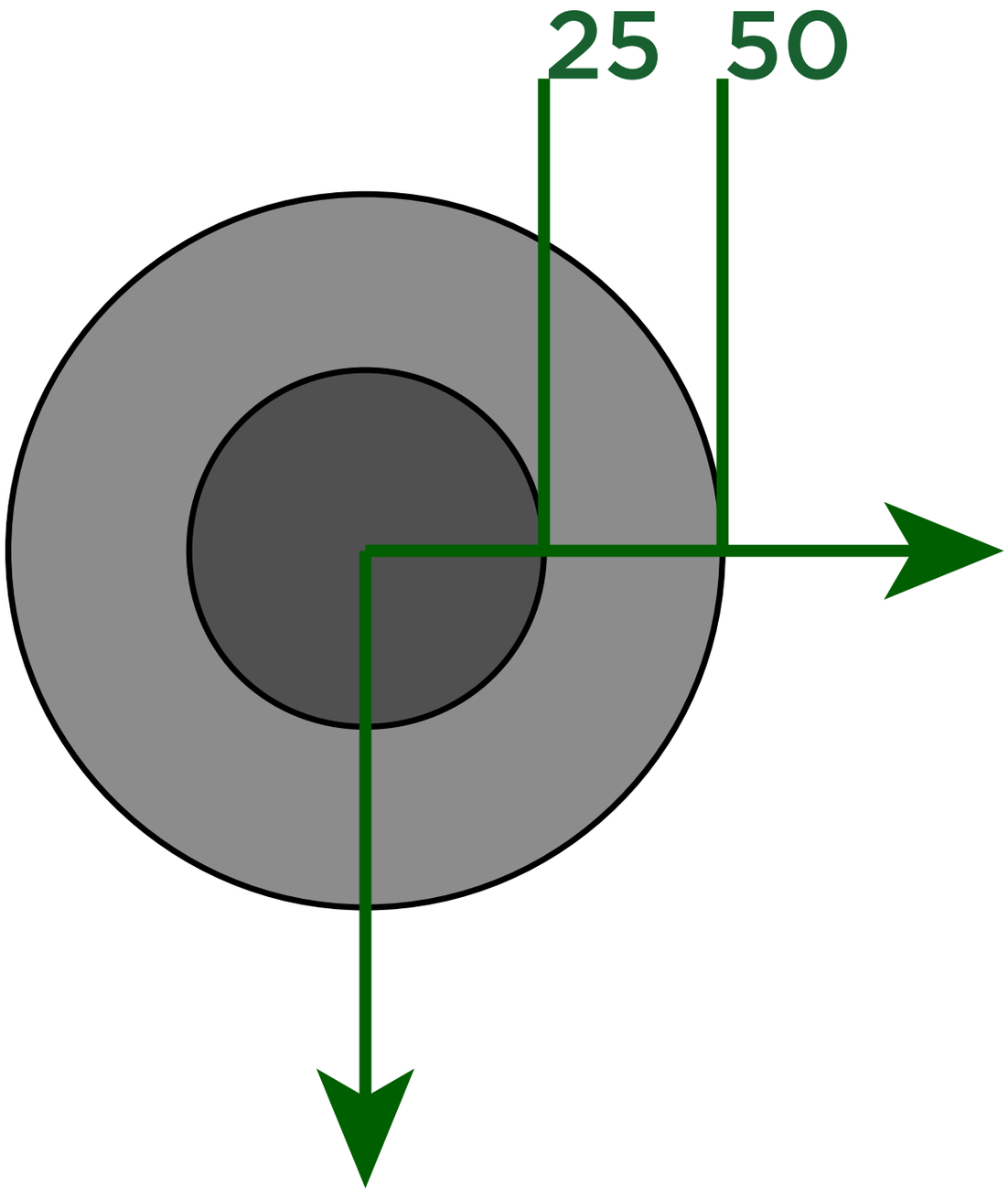
Hierarchical Modelling

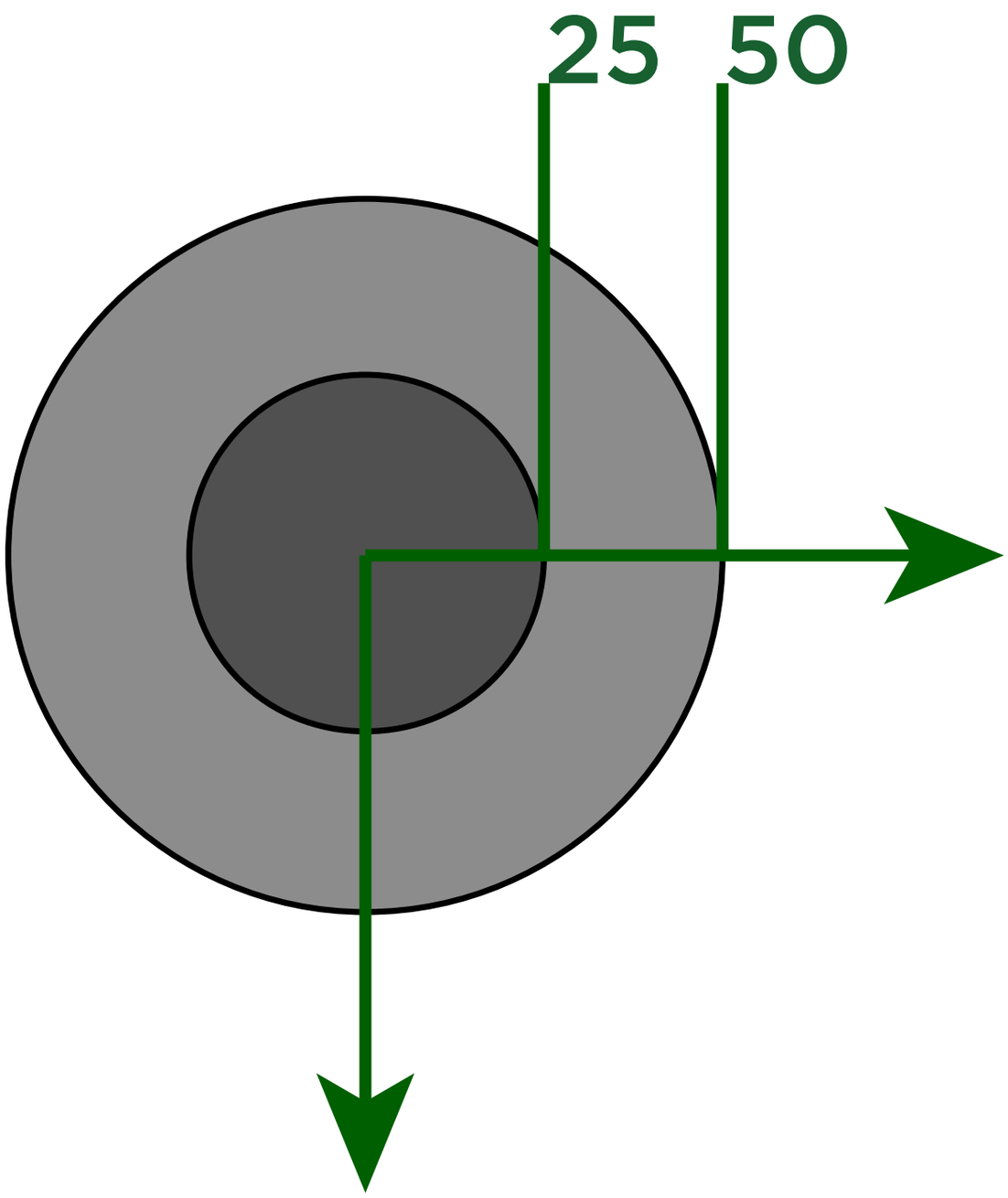
Geometric context also lets us express the relative spatial relationships between parts of an object.



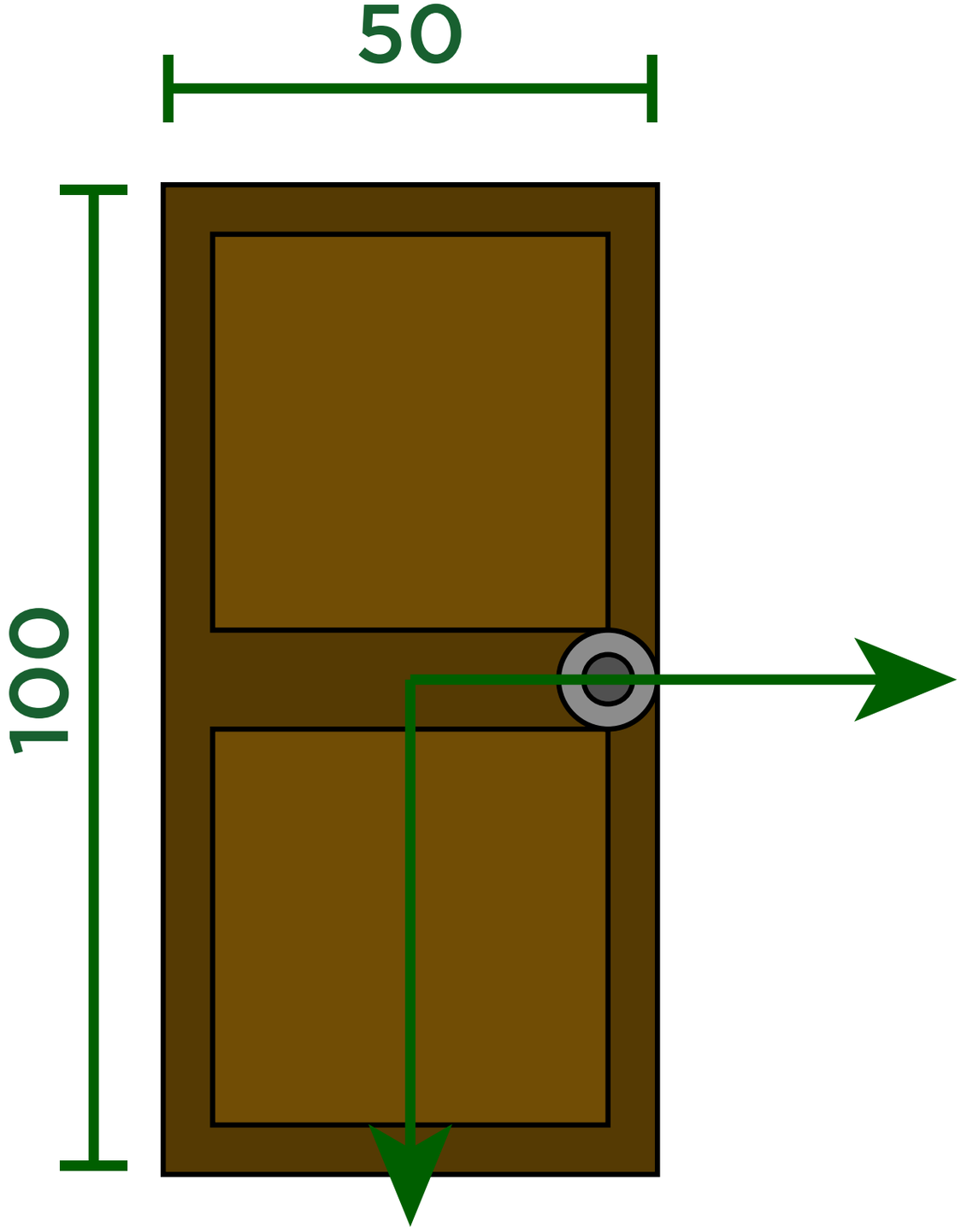
Hierarchical Modelling

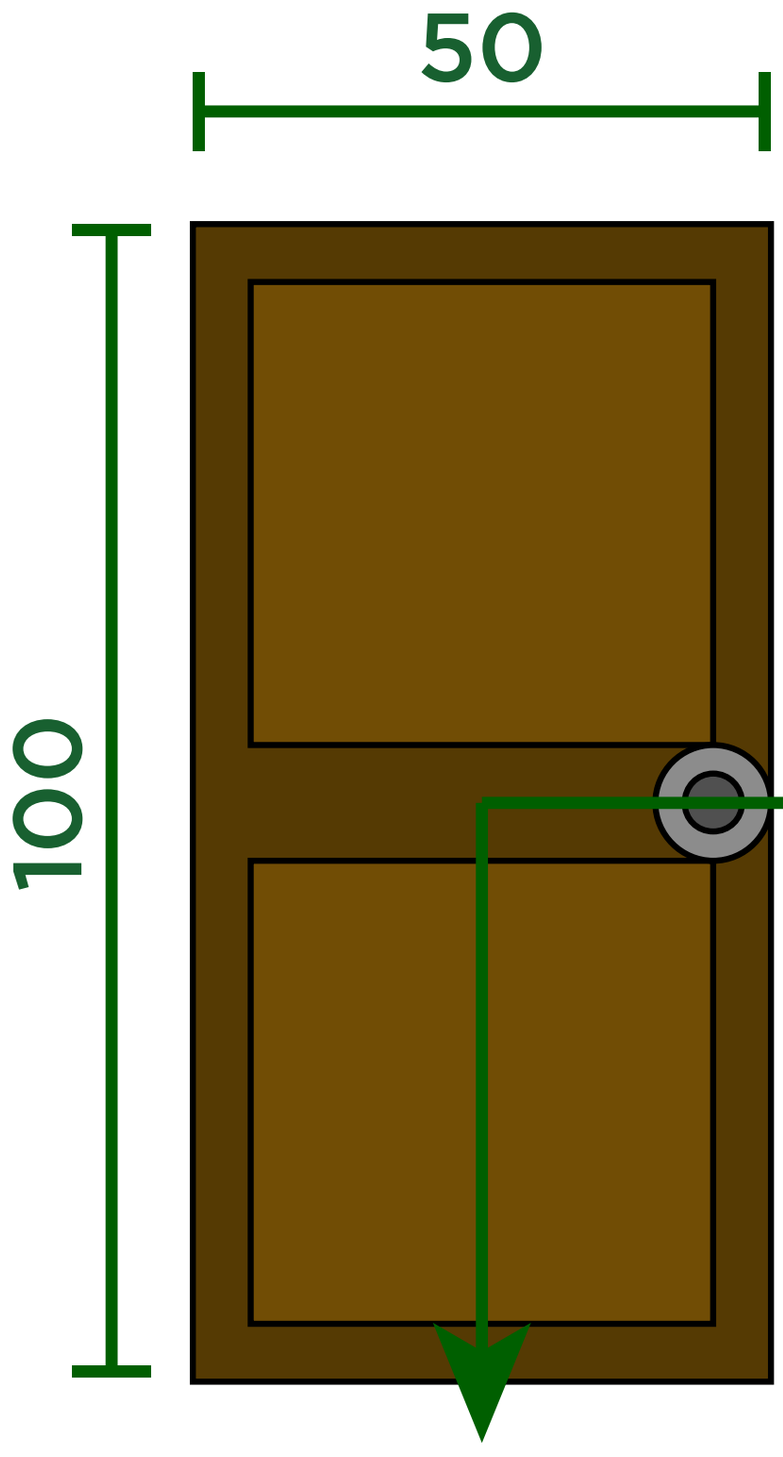
We can use these properties to build up complicated, interesting drawings from hierarchies of simpler pieces.


25
50

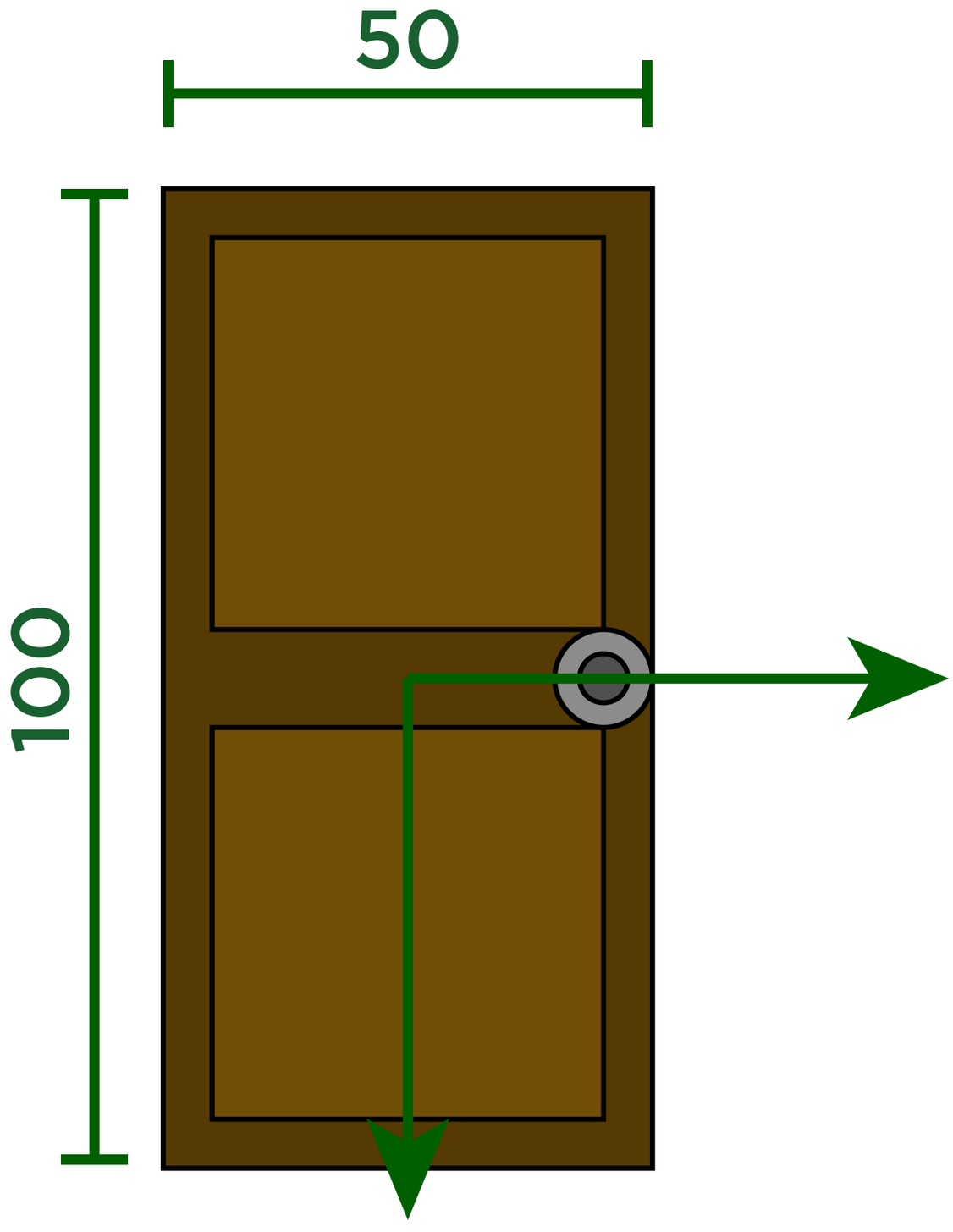



```
void doorknob()
{
   fill( 140 );
   ellipse( 0, 0, 100, 100 );
   fill( 80 );
   ellipse( 0, 0, 50, 50 );
}
```

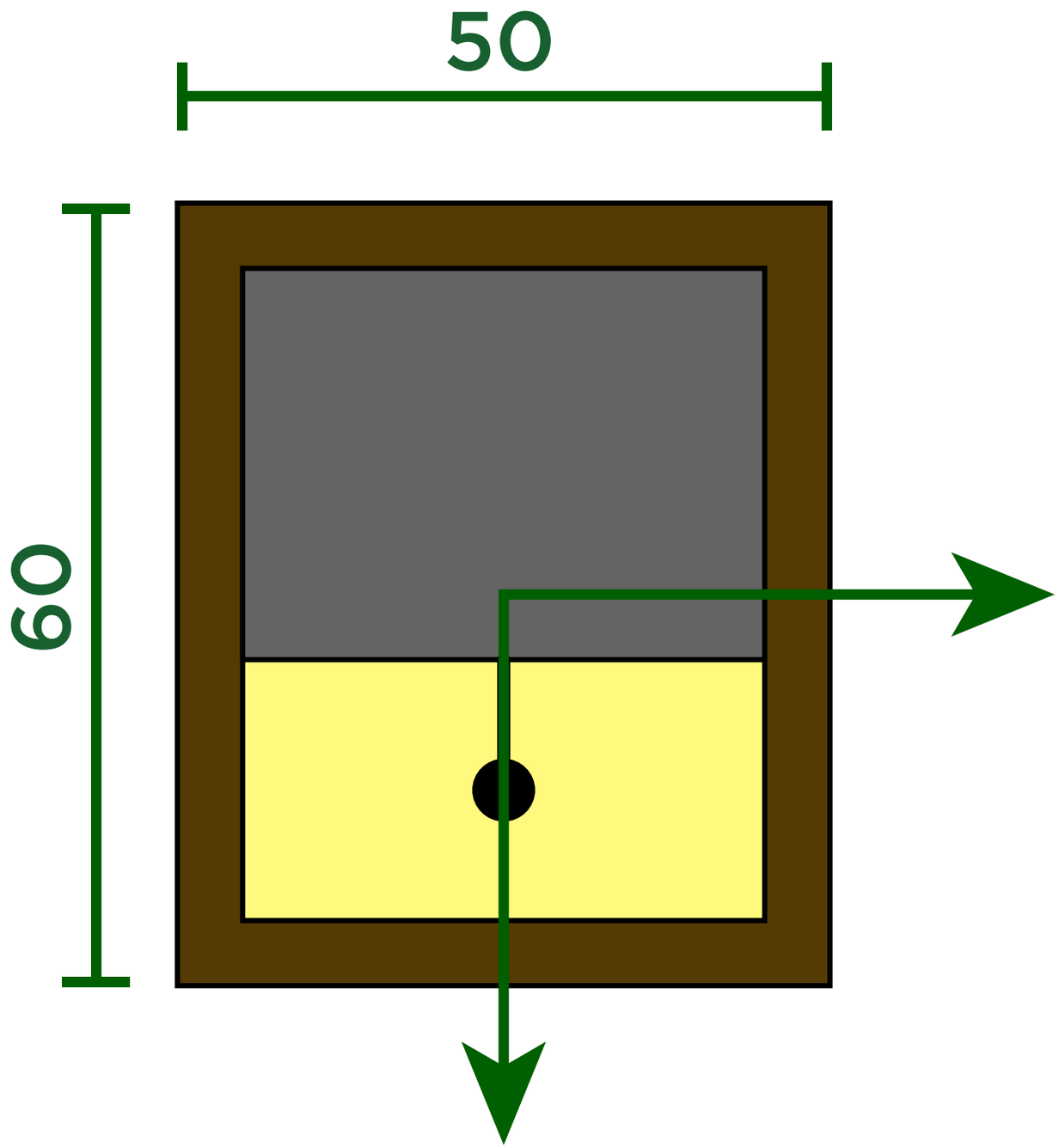

50
100




```
void door()
{
  fill( #553A03 );
  rect( -25, -50, 50, 100 );
  fill( #714D05 );
  rect( -20, -45, 40, 40 );
  rect( -20, 5, 40, 40 );
}
```




```
void door()
{
  fill( #553A03 );
  rect( -25, -50, 50, 100 );
  fill( #714D05 );
  rect( -20, -45, 40, 40 );
  rect( -20, 5, 40, 40 );

  pushMatrix();
  translate( 20, 0 );
  scale( 0.1 );
  doorknob();
  popMatrix();
}
```




```
void window()
{
  fill( #553A03 );
  rect( -25, -30, 50, 60 );
  fill( #FFF97E );
  rect( -20, -25, 40, 50 );
  fill( 100 );
  rect( -20, -25, 40, 30 );
  line( 0, 5, 0, 15 );
  fill( 0 );
  ellipse( 0, 15, 4, 4 );
}
```