

Module 06

Geometric Context

CS 106 Winter 2020

translate()

rotate()

scale()

push()

pop()

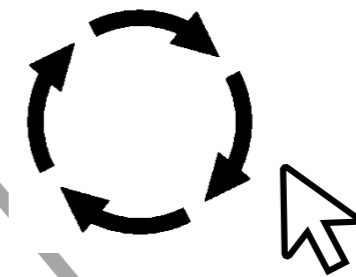
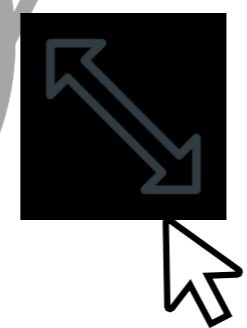
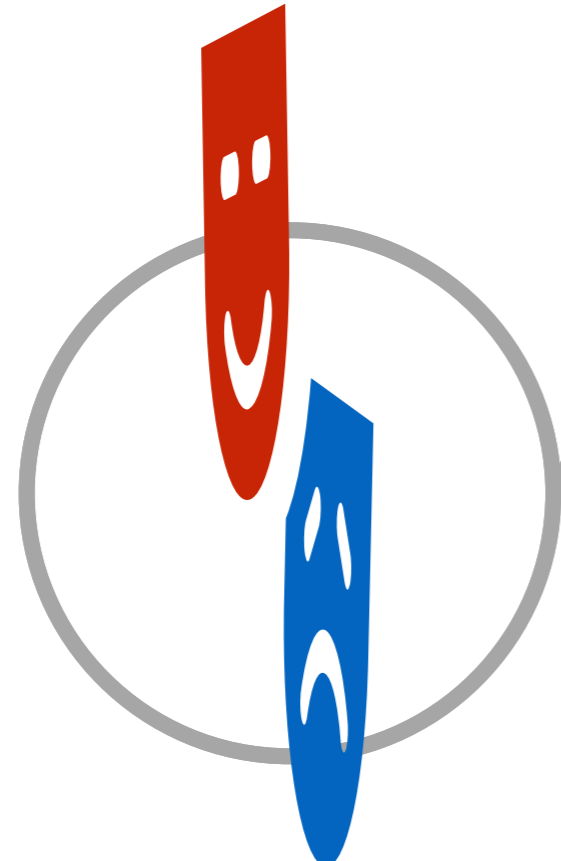
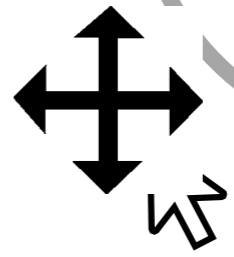
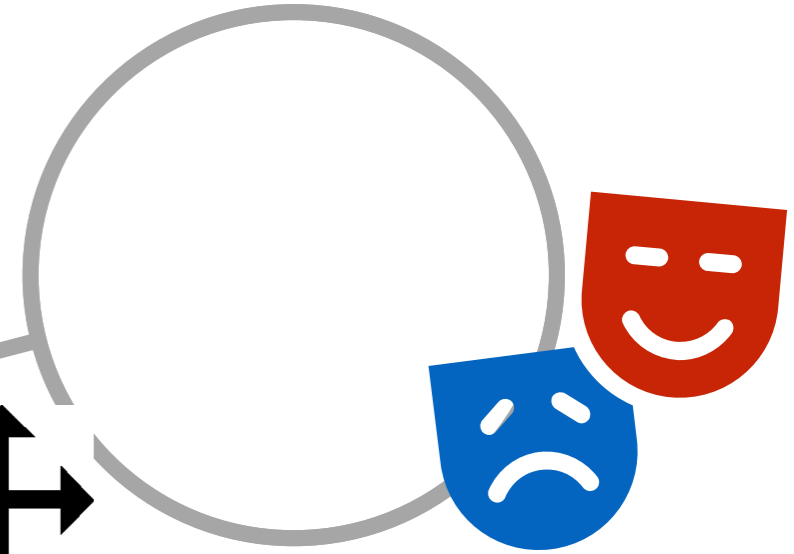


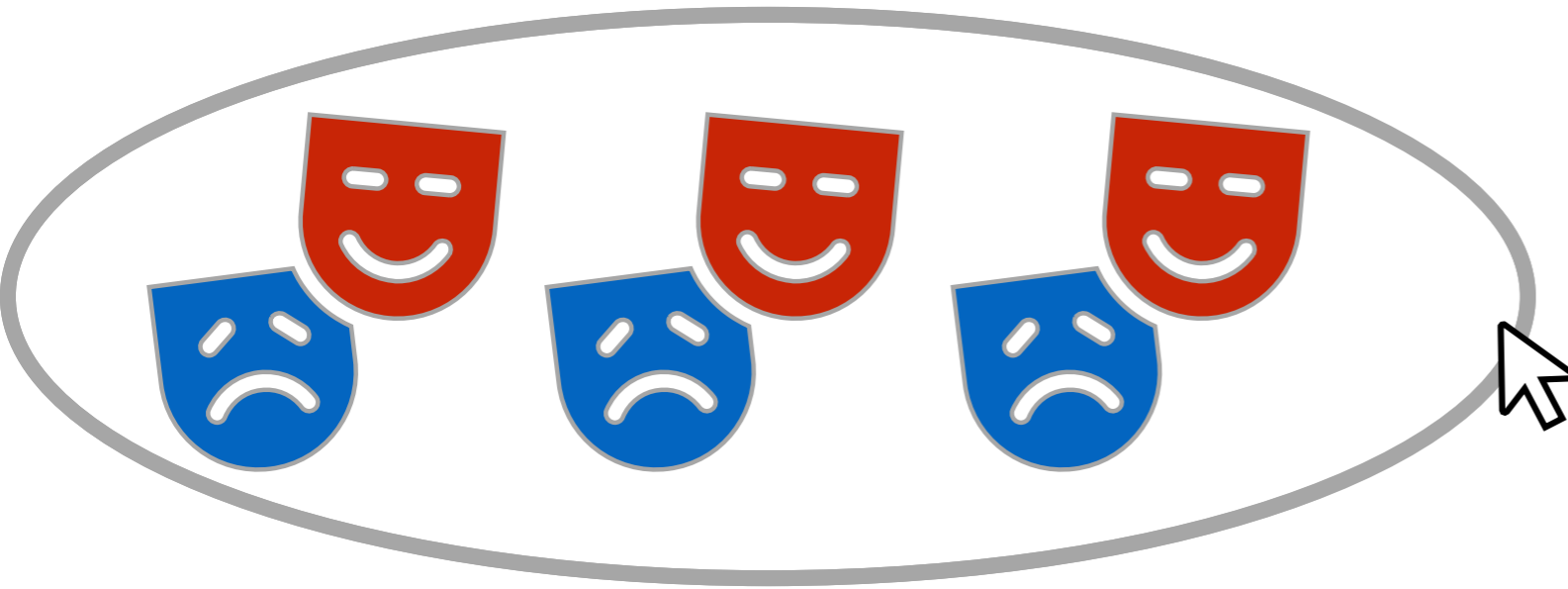
becomes...

translate()



rotate()

scale()





Group Objects

-  **Group**
-  **Ungroup**

push()



Checkpoint

pop()



Restore



translate()

rotate()

scale()

push()

&

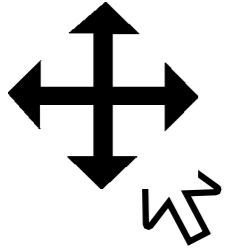
pop()

Each is
simple/familiar
enough

but...

get them in the right order

Tricky order



Person with a drawing tool:

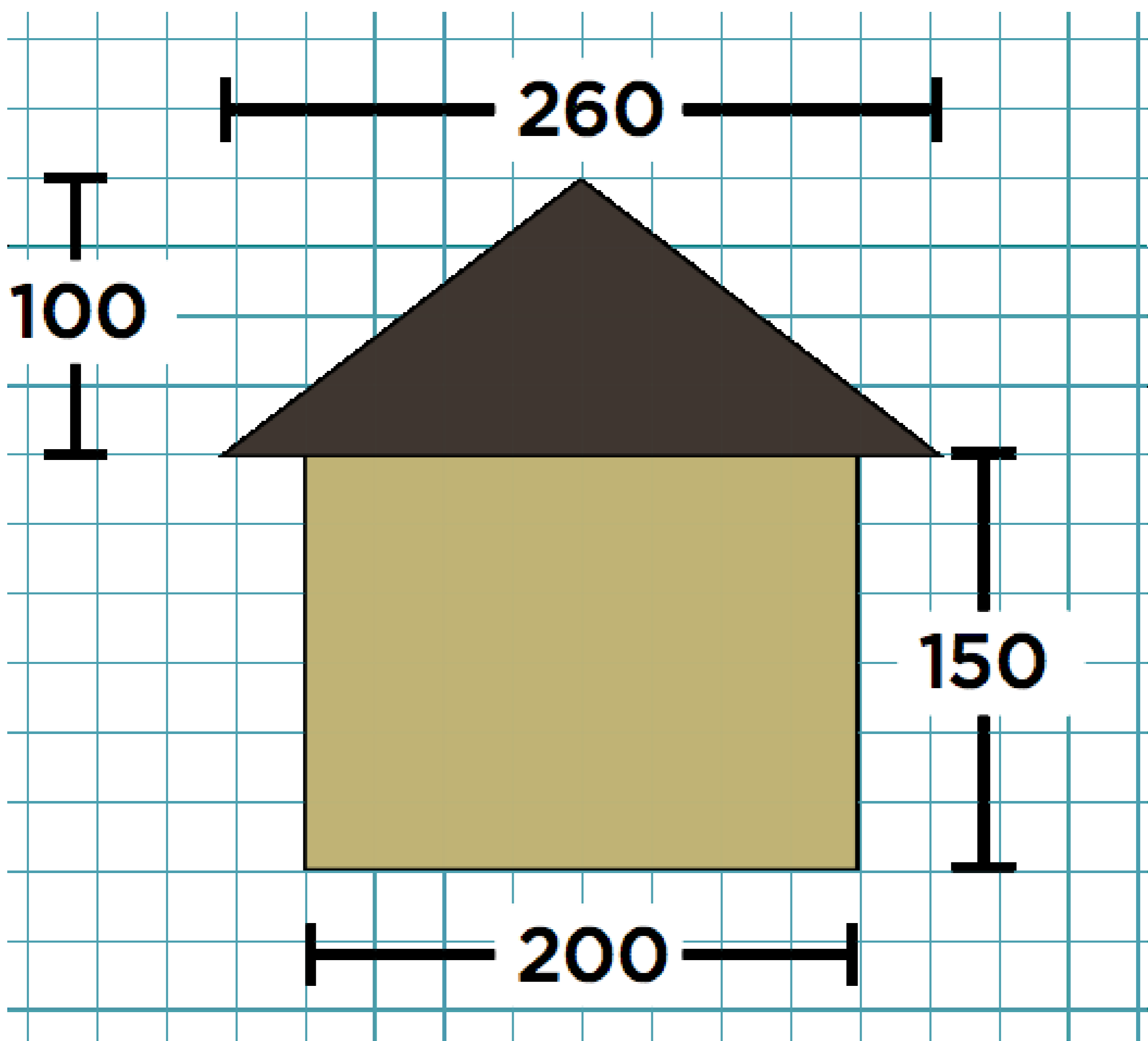
1. Draw it. Who cares where.
2. See it there.
3. Move it where it goes.
4. See it there.

Code in a drawing context:

1. Move the context.
2. Draw. (No peeking.)
3. Leave the context.
4. See what you drew.

Understanding Context

with *translate*




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

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

```
function setup() {  
  createCanvas(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

```
function setup() {  
  createCanvas(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);  
}
```

```
function drawHouseAt(x, y) {
```

```
}
```

```
function setup() {  
  createCanvas(600, 350);  
  background(255);
```

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

```
}
```

```
function drawHouseAt(x, 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);  
}
```

```
function setup() {  
  createCanvas(600, 350);  
  background(255);  
  
  drawHouseAt(0, 0);  
  drawHouseAt(310, -10);  
}
```

```
let global_x = 0.0;
```

```
let global_y = 0.0;
```

```
function myRect(x, y, w, h) {  
    rect(x + global_x, y + global_y, w, h);  
}
```

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

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

```
function setup() {  
    createCanvas(600, 350);  
    background(255);  
  
    global_x = 0;  
    global_y = 0;  
    drawHouse();  
  
    global_x = 310;  
    global_y = -10;  
    drawHouse();  
}
```



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

```
function setup() {  
    createCanvas(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 global amount of translation is the “geometric context”.

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

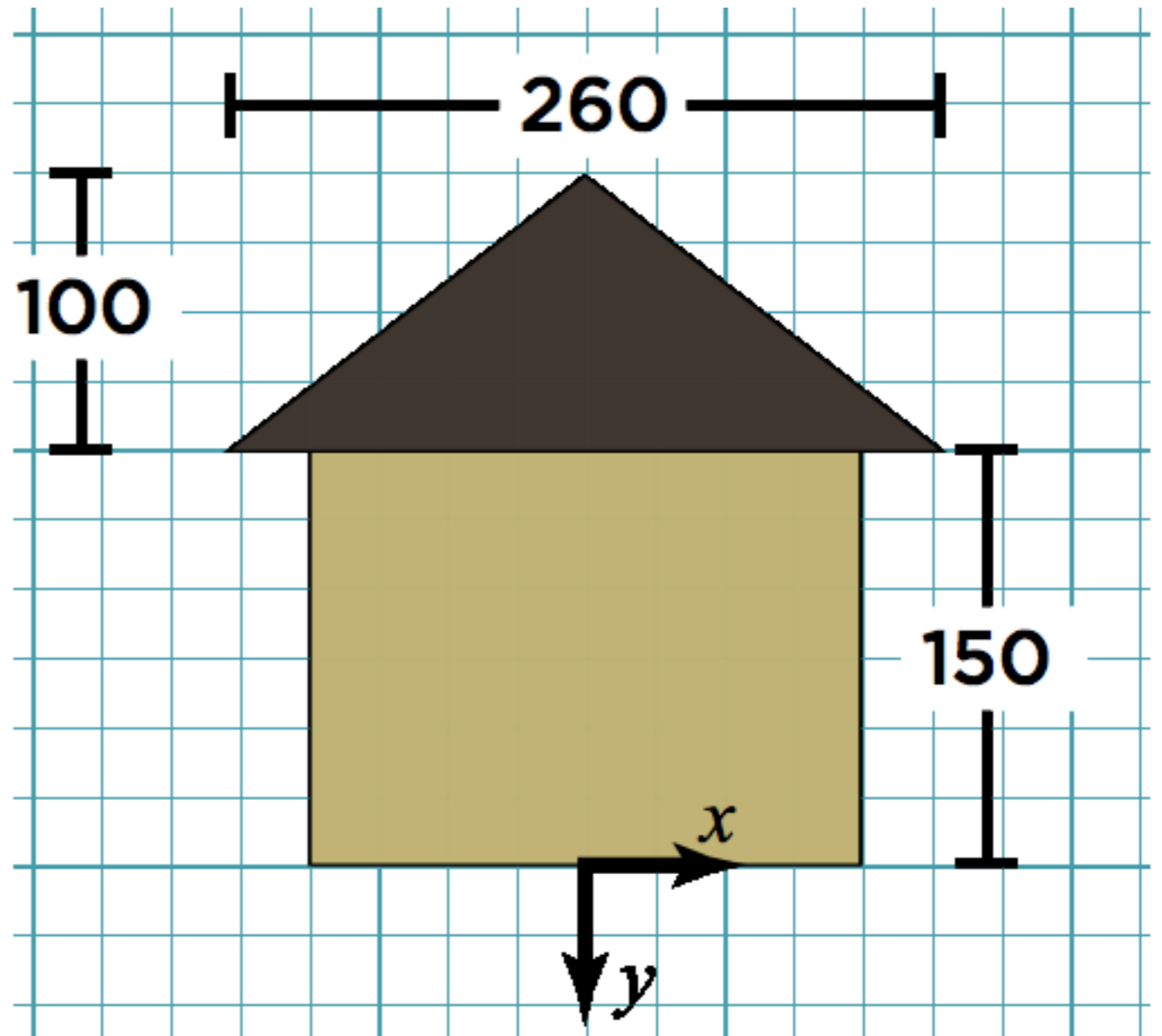
```
function setup() {  
    createCanvas(600, 350);  
    background(255);  
  
    drawHouse();  
  
    translate(310, -10);  
    drawHouse();  
}
```

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

```
function setup() {  
    createCanvas(600, 350);  
}
```

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

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

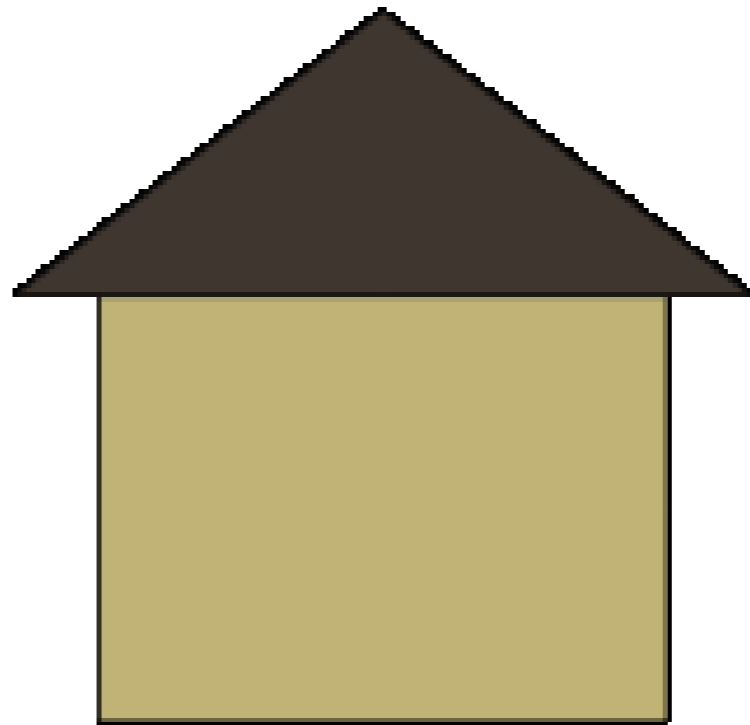
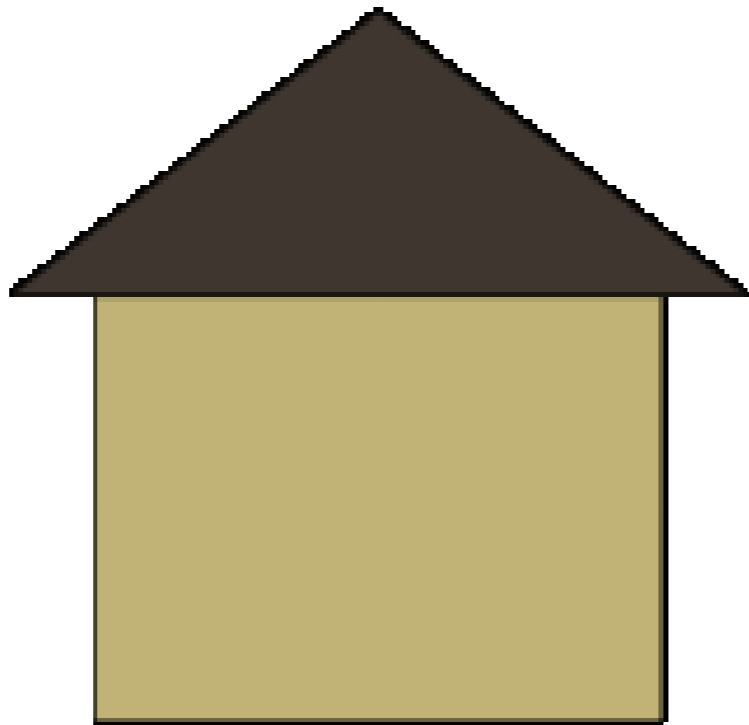
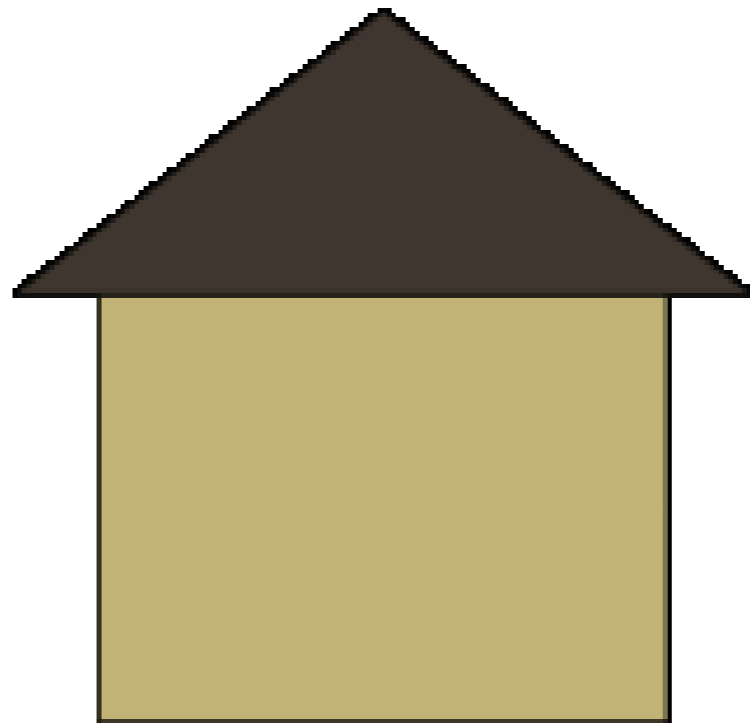
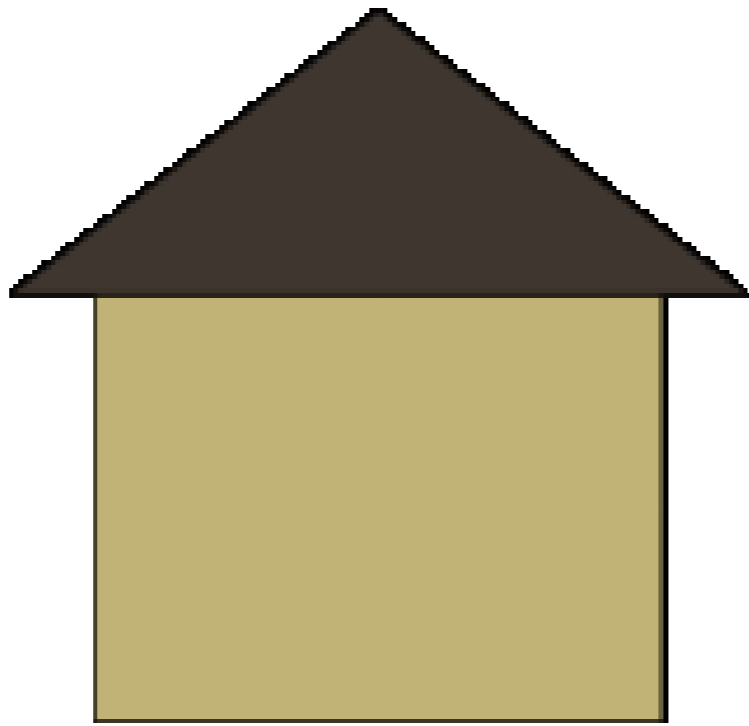


```
function drawHouse() {  
    fill(191, 179, 117);  
    rect(-100, -150, 200, 150);  
    fill(62, 54, 47);  
    triangle(-130, -150, 0, -250, 130, -150);  
}
```

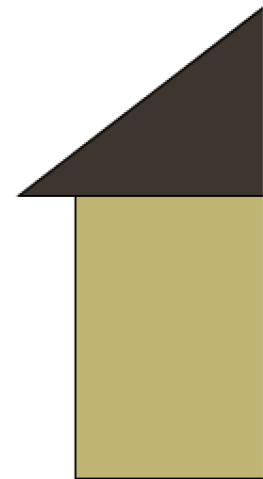
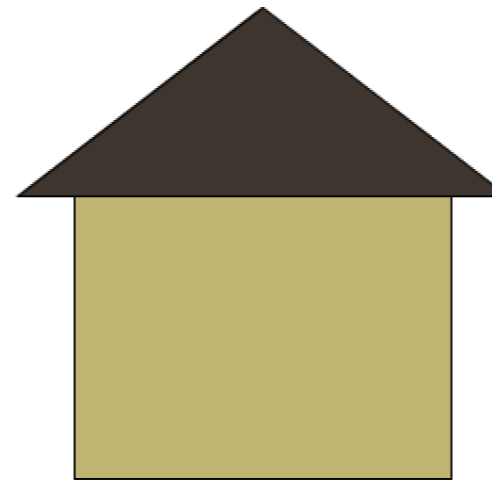
Context Accumulates

so we

take control with *push* and *pop*



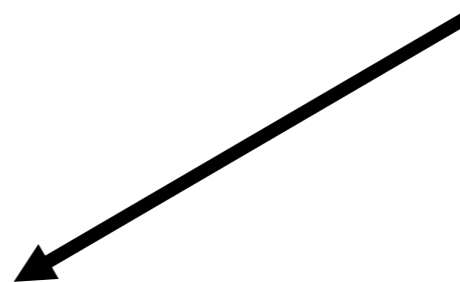
```
function draw() {  
  background(255);  
  
  translate(150, 280);  
  drawHouse();  
  translate(450, 280);  
  drawHouse();  
  translate(150, 580);  
  drawHouse();  
  translate(450, 580);  
  drawHouse();  
}
```



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

Initial context: translate by 0, 0

```
function draw() {  
  background(255);  
  
  translate(150, 280);  
  drawHouse();  
  translate(450, 280);  
  drawHouse();  
  translate(150, 580);  
  drawHouse();  
  translate(450, 580);  
  drawHouse();  
}
```



```
function draw() {  
  background(255);  
  
  translate(150, 280);  
  drawHouse(); ← Translated by 150, 280  
  translate(450, 280);  
  drawHouse();  
  translate(150, 580);  
  drawHouse();  
  translate(450, 580);  
  drawHouse();  
}
```

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

```
  translate(150, 280);
```

```
  drawHouse();
```

```
  translate(450, 280);
```

```
  drawHouse();
```

```
  translate(150, 580);
```


```
  drawHouse();
```

```
  translate(450, 580);
```

```
  drawHouse();
```

```
}
```

Translated by 150 + 450, 280
+ 280




```
void draw()  
{  
    background( 255 );  
  
    translate( 150, 280 );  
    drawHouse();  
    translate( 450, 280 );  
    drawHouse();  
    translate( 150, 580 );  
    drawHouse(); ←  
    translate( 450, 580 );  
    drawHouse();  
}
```

Translated by
 $150 + 450 + 150,$
 $280 + 280 + 580$

```
void draw()  
{  
    background( 255 );  
  
    translate( 150, 280 );  
    drawHouse();  
    translate( 450, 280 );  
    drawHouse();  
    translate( 150, 580 );  
    drawHouse();  
    translate( 450, 580 );  
    drawHouse();  
}
```

Translated by
150 + 450 + 150 + 450,
280 + 280 + 580 + 580



`push()`: Set a “checkpoint”, remembering the current geometric context.

`pop()`: Go back to the context that was saved, before the last push that hadn't been popped yet

Geometric
Context:

```
push();
```

```
pop();
```

Arrays:

```
arr.push(x);
```

```
arr.pop();
```

Okay, not homonyms.

But importantly, the two *push* functions live in different universes, and won't affect each other. (Same for *pop*.)

```
function draw() {  
  background(255);  
  
  push();  
  translate(150, 280);  
  drawHouse();  
  pop();  
  
  push();  
  translate(450, 280);  
  drawHouse();  
  pop();  
  
  push();  
  translate(150, 580);  
  drawHouse();  
  pop();  
  
  push();  
  translate(450, 580);  
  drawHouse();  
  pop();  
}
```

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

*Make a shape
from these points*

```
beginShape();
```

```
vertex(...);
```

```
vertex(...);
```

```
vertex(...);
```

```
endShape();
```

*Put these shapes
over **here***

```
push();
```

```
translate(...);
```

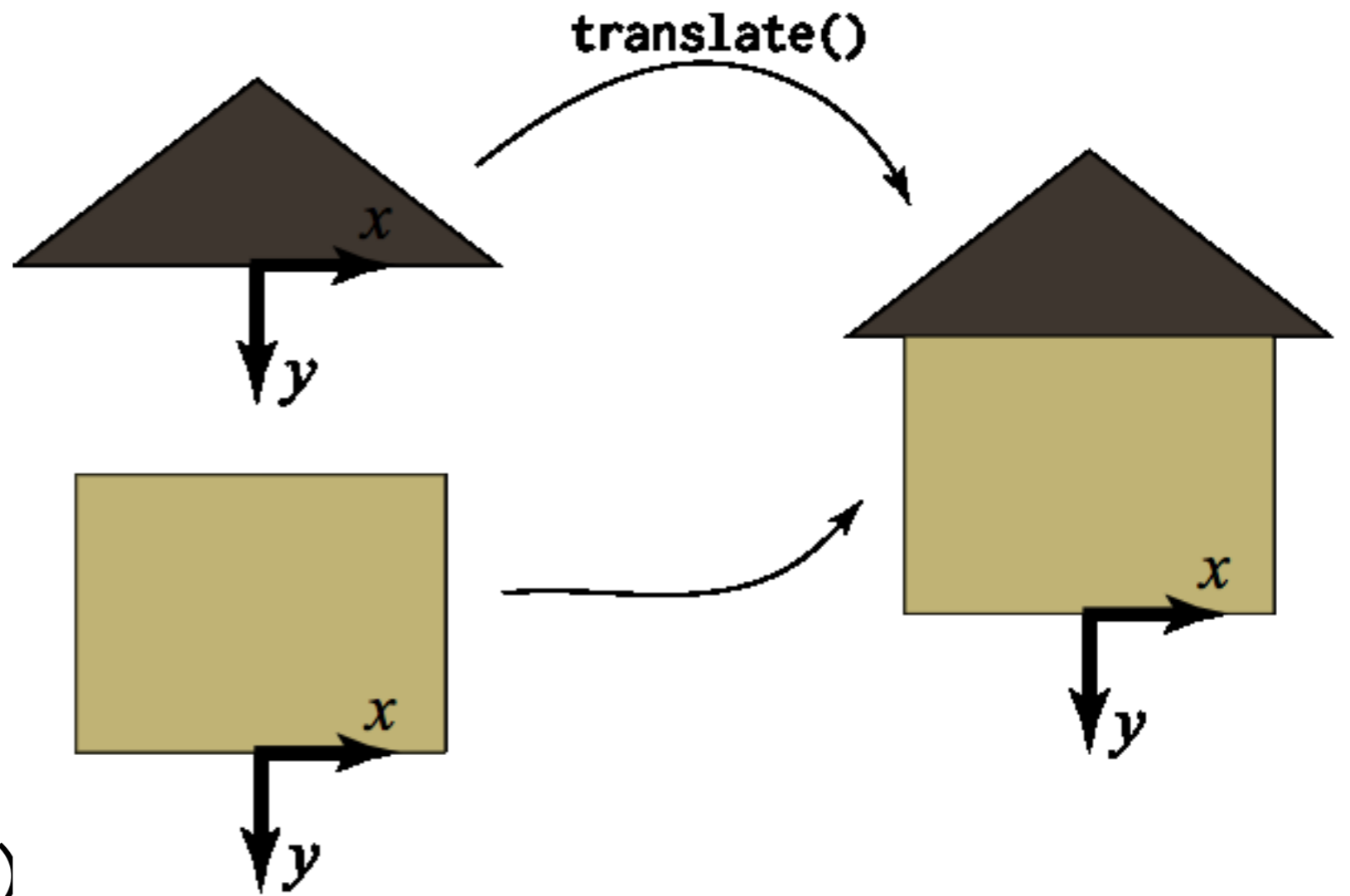
```
rect(...);
```

```
triangle(...);
```

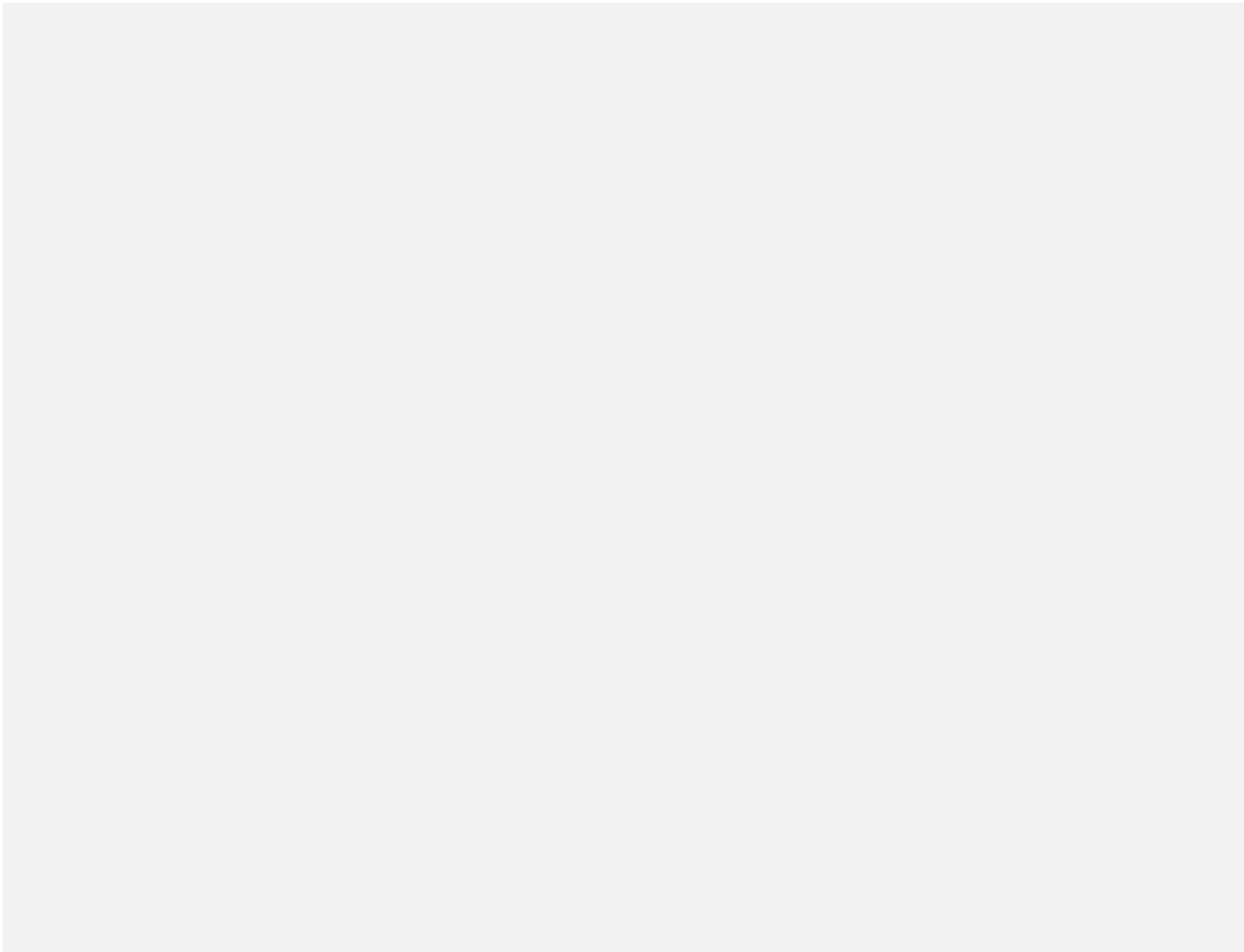
```
ellipse(...);
```

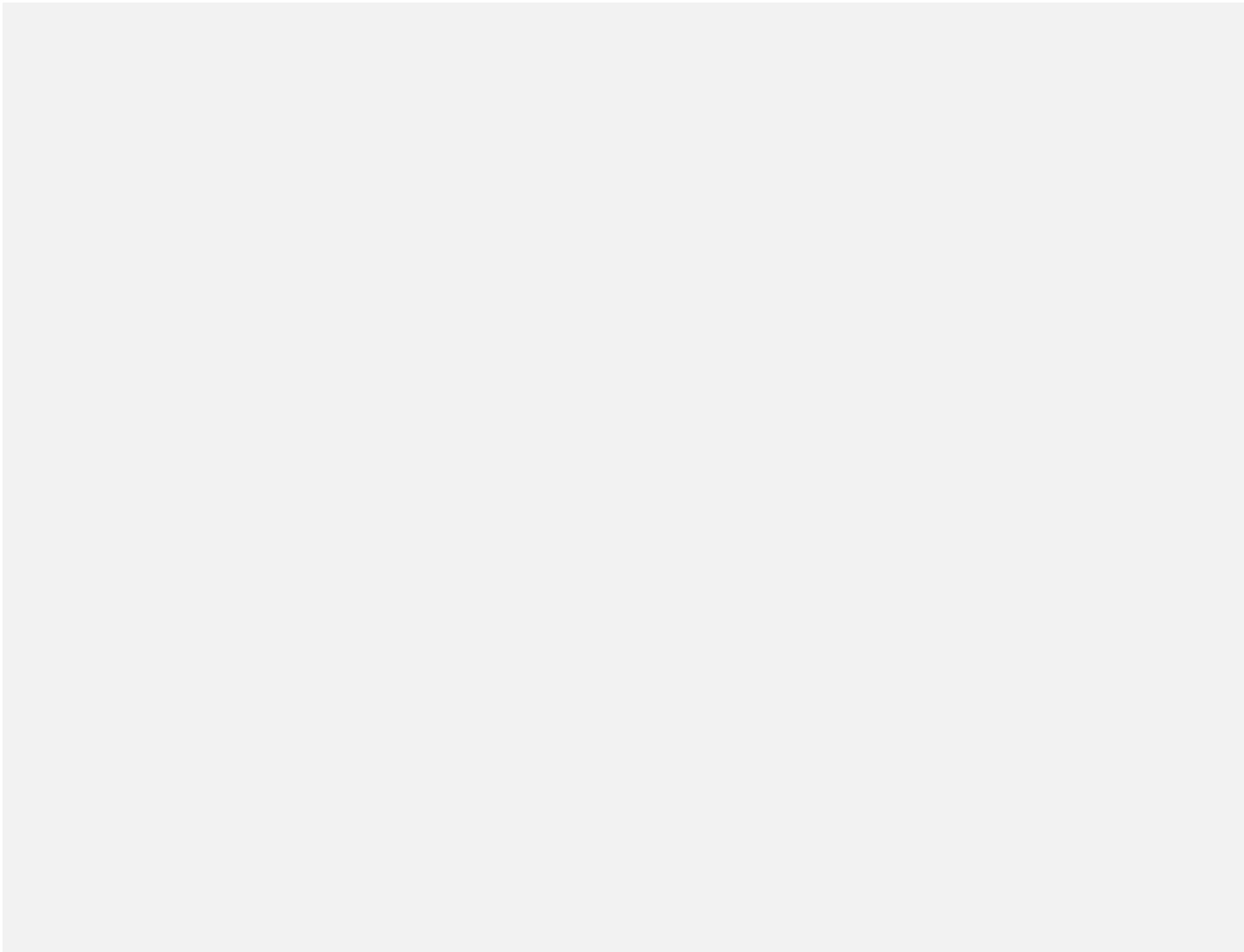
```
pop();
```

```
function draw() {  
    background(255);  
  
    push();  
    translate(150, 280);  
    drawHouse();  
    translate(300, 0);  
    drawHouse();  
    pop();  
  
    push();  
    translate(150, 580);  
    drawHouse();  
    translate(300, 0);  
    drawHouse();  
    pop();  
}
```



```
function drawHouse()  
  fill(191, 179, 117),  
  rect(-100, -150, 200, 150);  
  fill(62, 54, 47);  
  
  push();  
  translate(0, -150);  
  triangle(-130, 0, 0, -100, 130, 0);  
  pop();  
}
```





When order matters

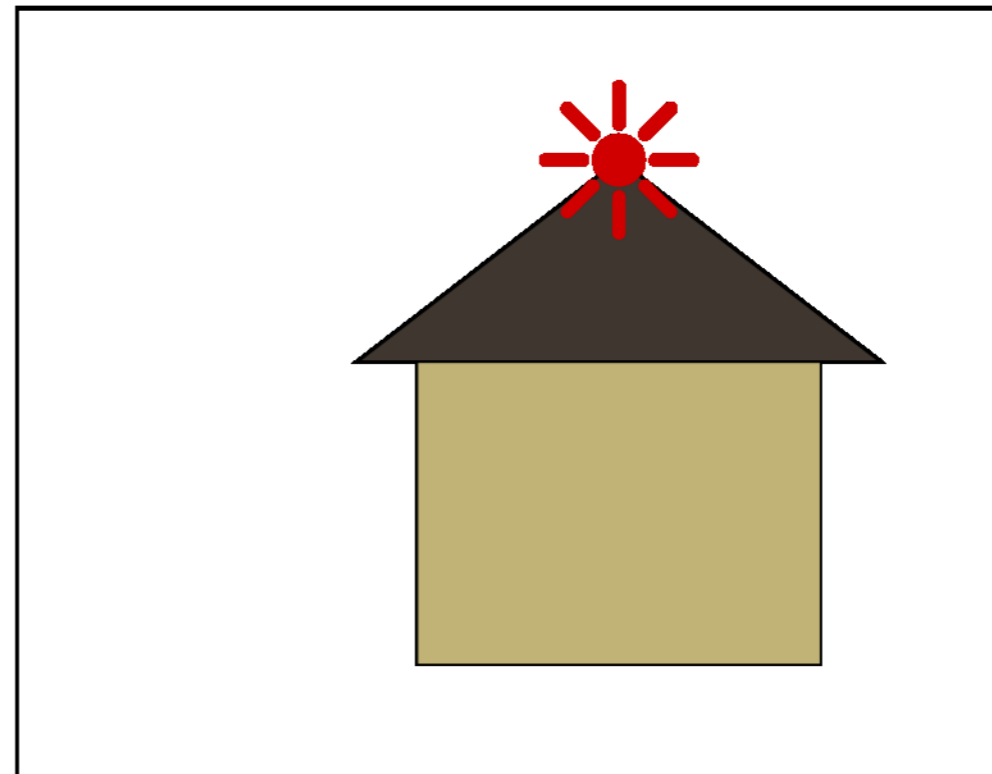
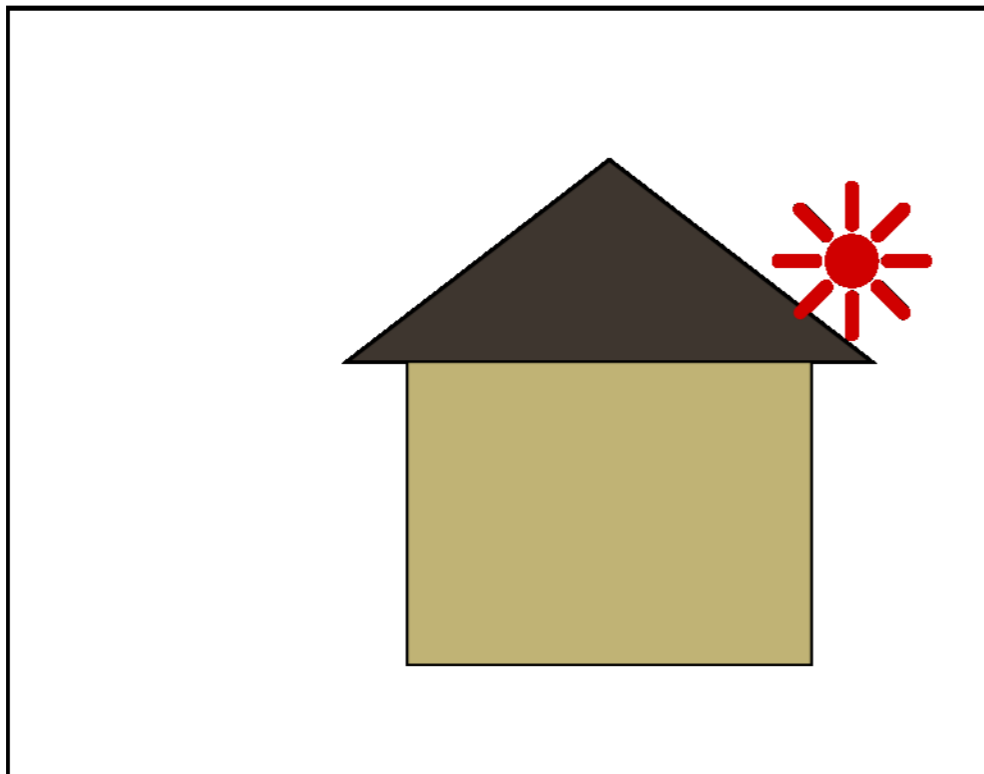
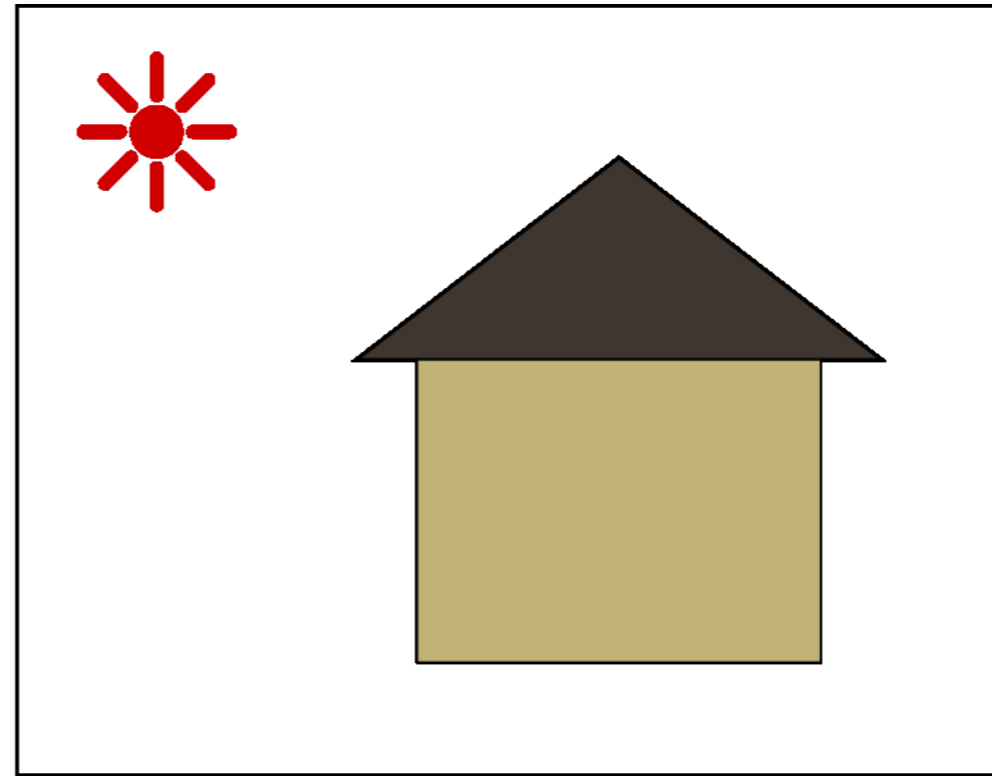
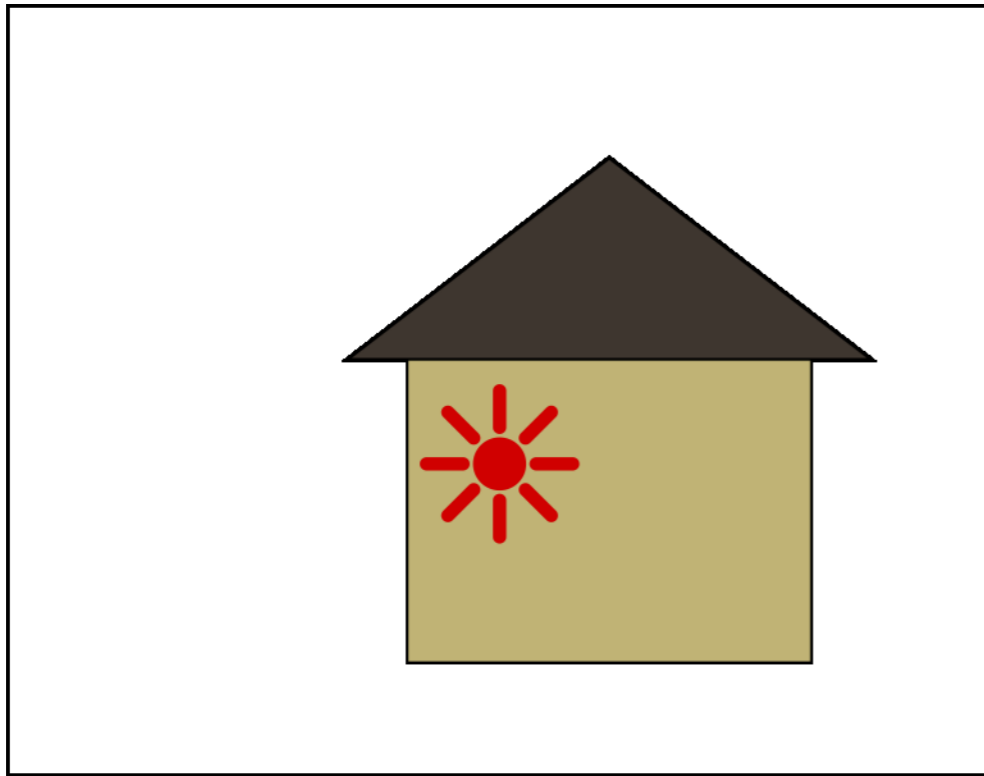
Getting *rotate* and *scale* right

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

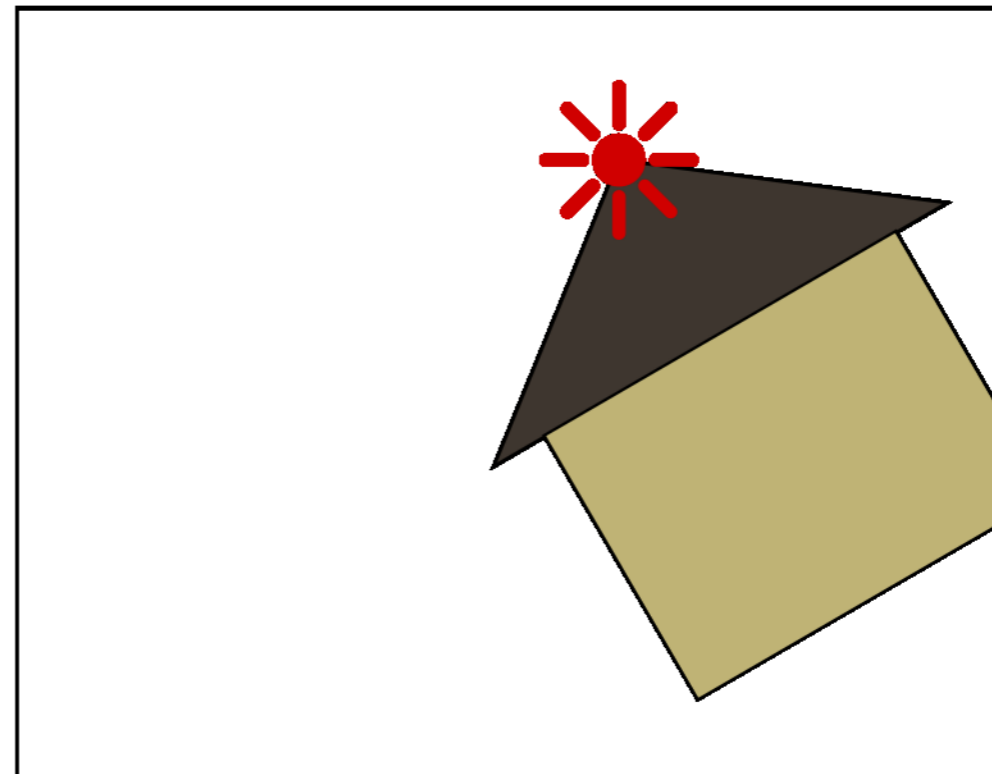
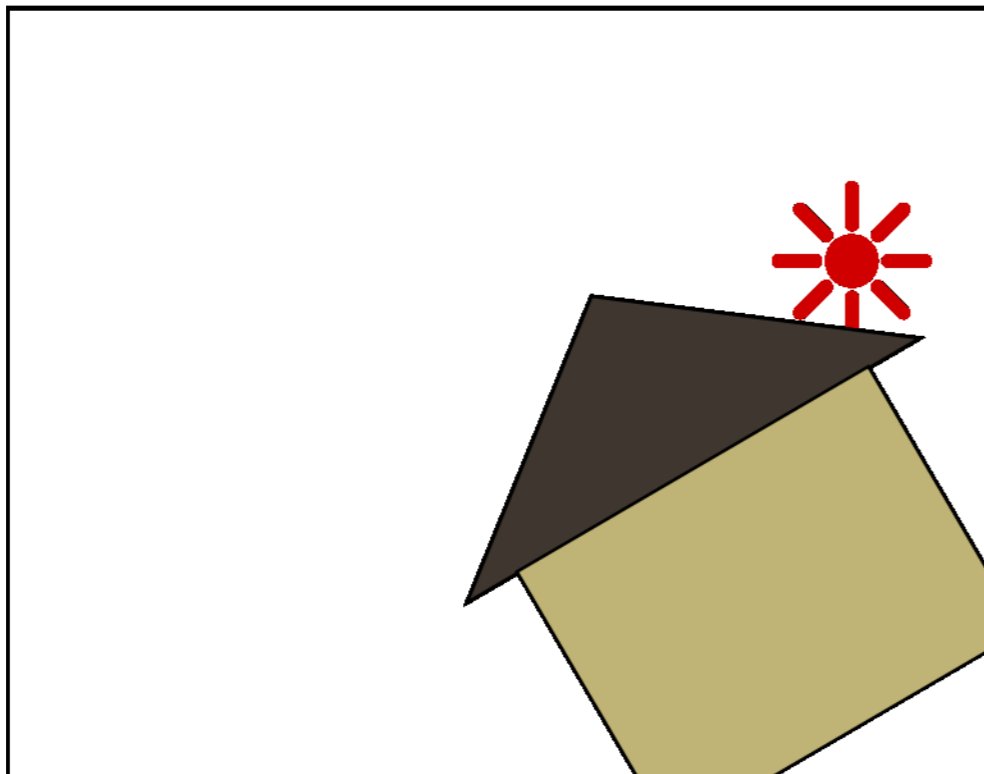
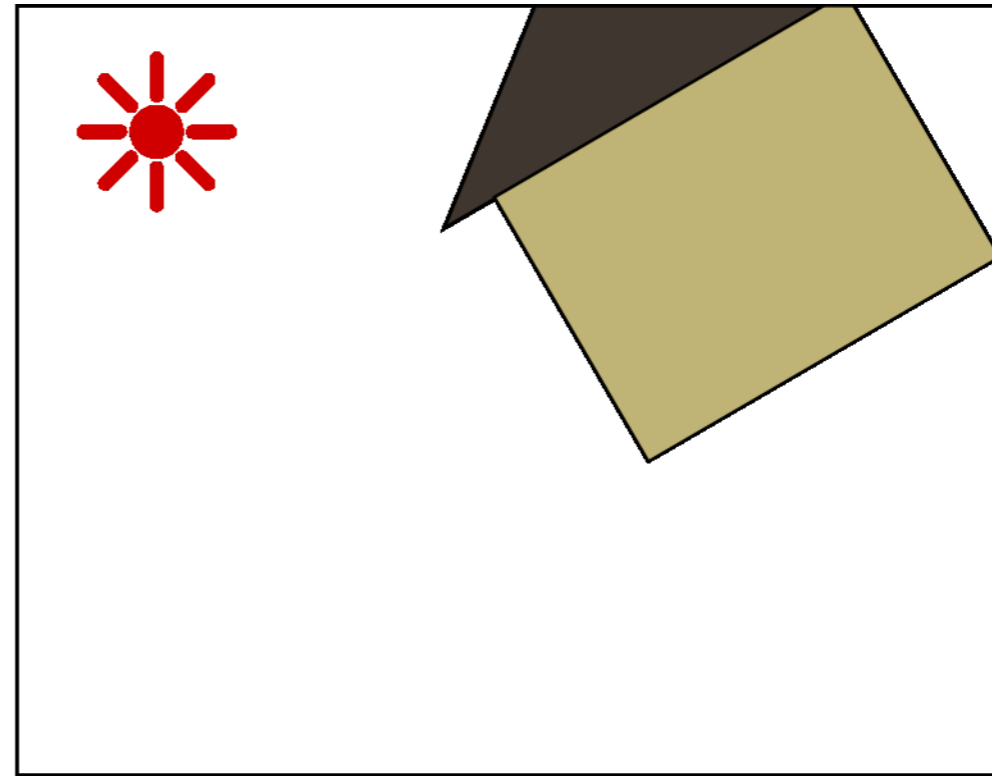
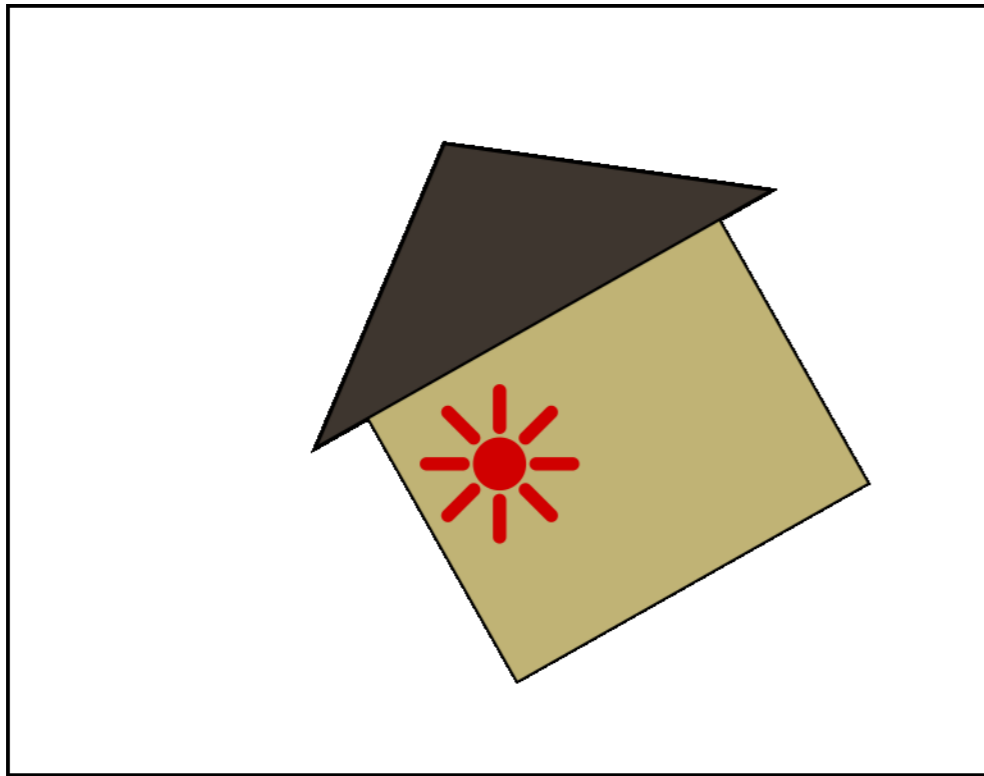
scale(a, b): Scale the current geometric context by ratios a in the x direction and b in the y direction.

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*.

Wherever that is, *right now*.

Order matters!

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

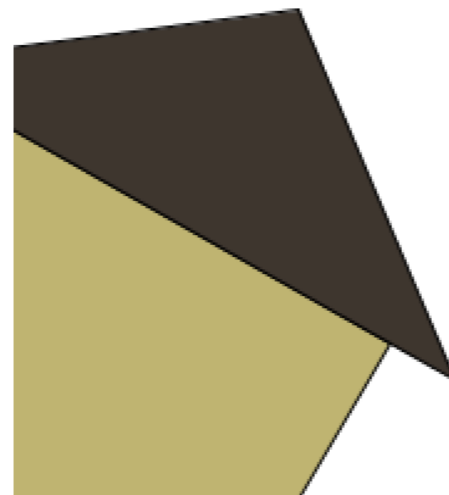
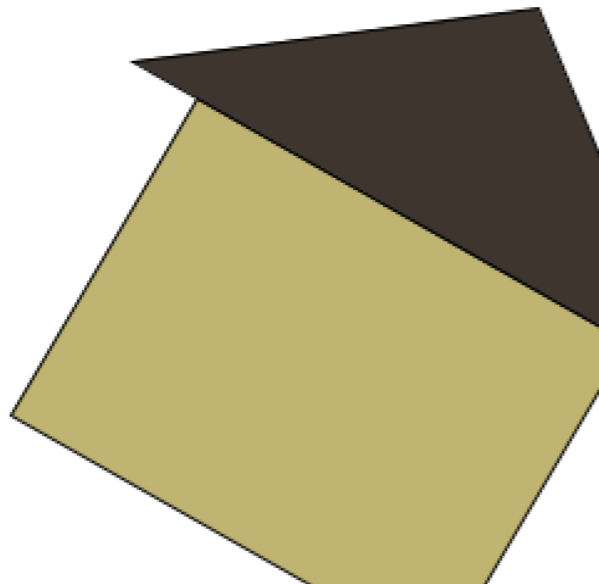
```
  translate( 150, 280 );  
  rotate( radians( 30 ) );  
  drawHouse();
```

```
}
```

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

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

```
}
```



Understanding order, Version 1

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

```
  translate( 150, 280 );
```

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

```
}
```

In this first model, read the drawing command first. Then read the transformations, *backward*, changing your imagined drawing each time.

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

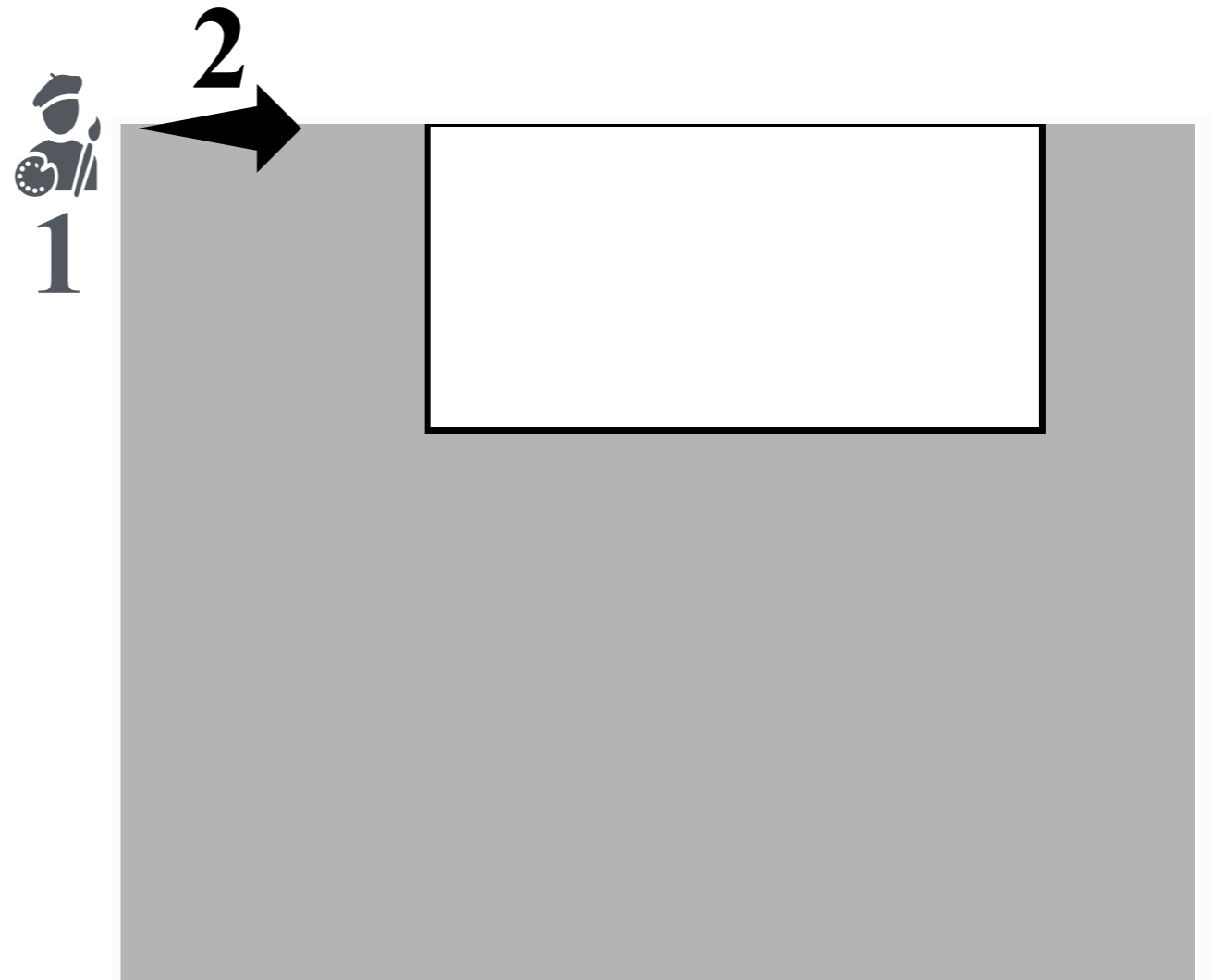
In this first model, read the drawing command first. Then read the transformations, *backward*, changing your imagined drawing each time.

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



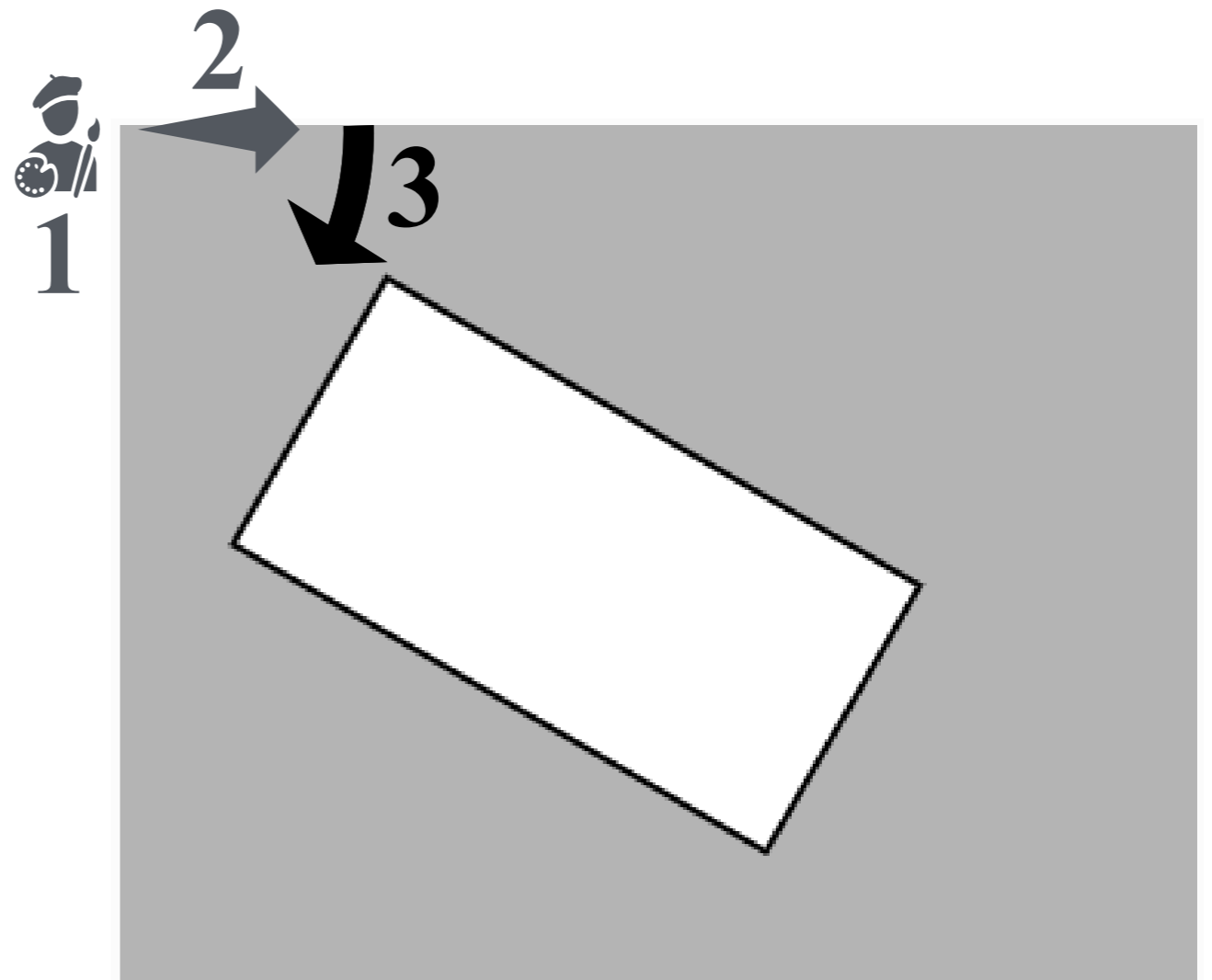
In this first model, read the drawing command first. Then read the transformations, *backward*, changing your imagined drawing each time.

```
// rotate( radians( 30 ) );  
2. translate( 100, 0 );  
1. rect( 0, 0, 200, 100 );
```



In this first model, read the drawing command first. Then read the transformations, *backward*, doing it to your imagined drawing each time.

3. `rotate(radians(30));`
2. `translate(100, 0);`
1. `rect(0, 0, 200, 100);`



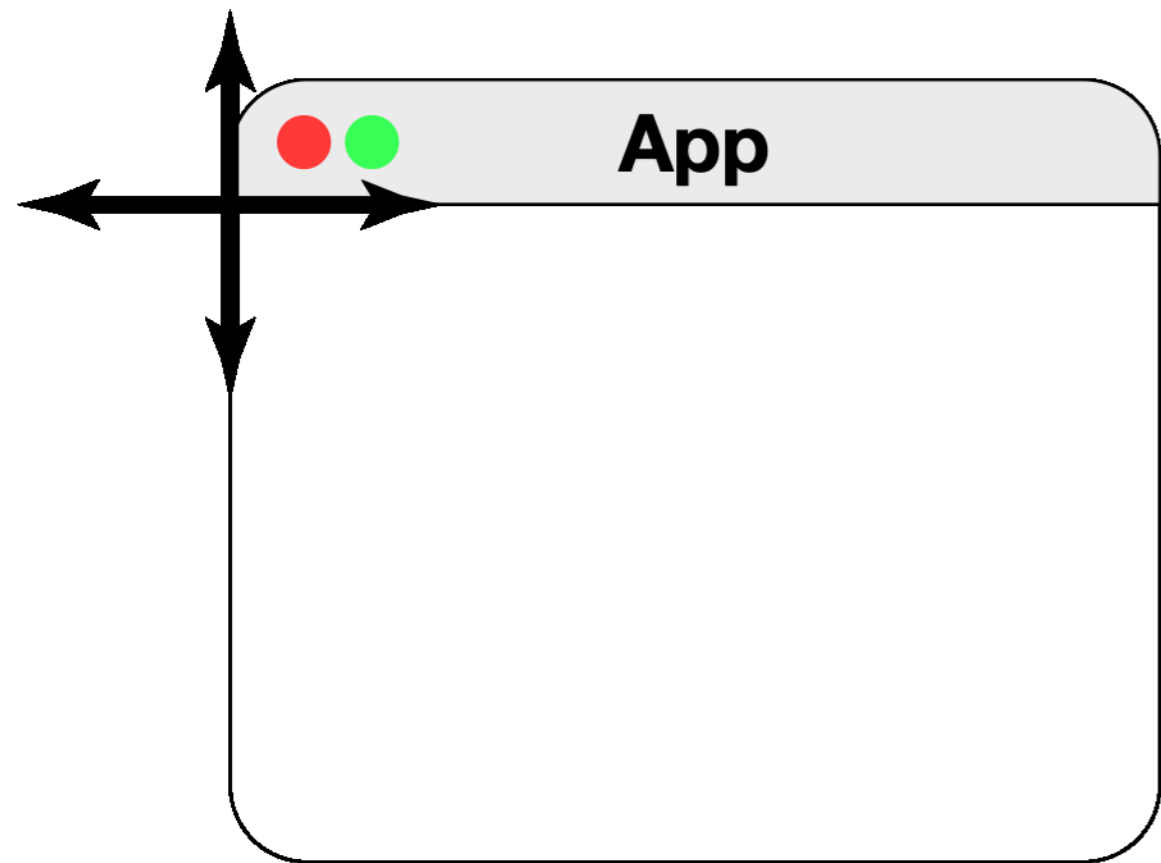
Understanding order, Version 2

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

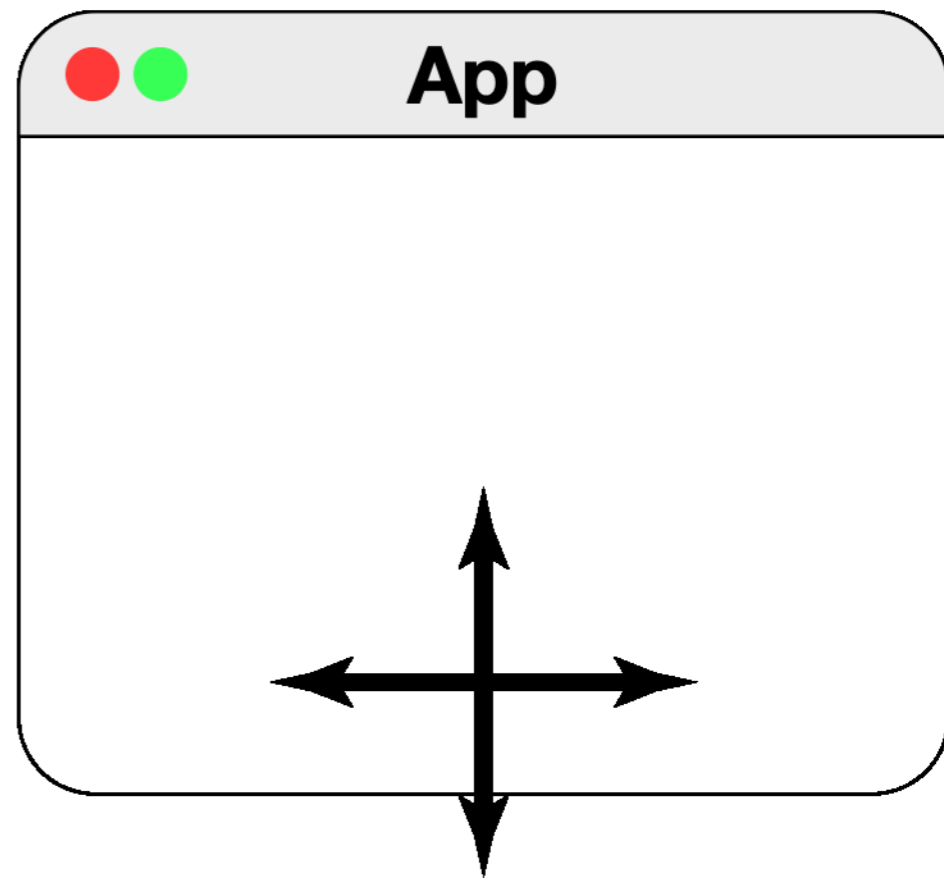
```
  translate( 150, 280 );
```

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

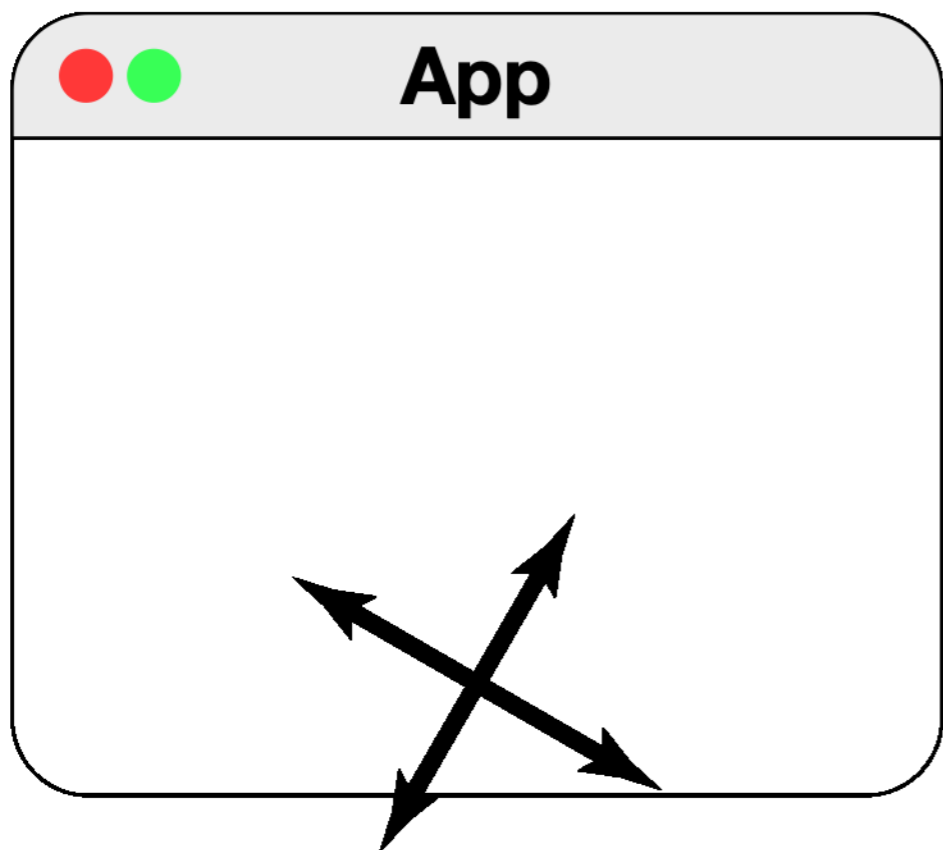
```
}
```



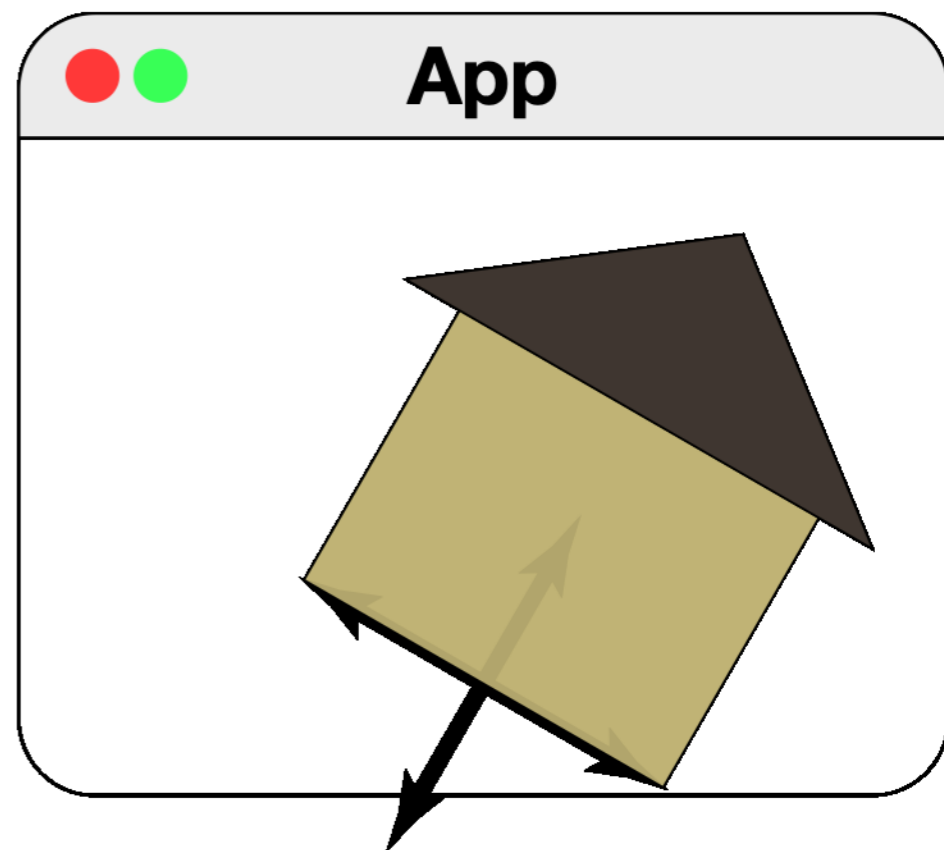
`translate()`



`rotate()`



`drawHouse()`

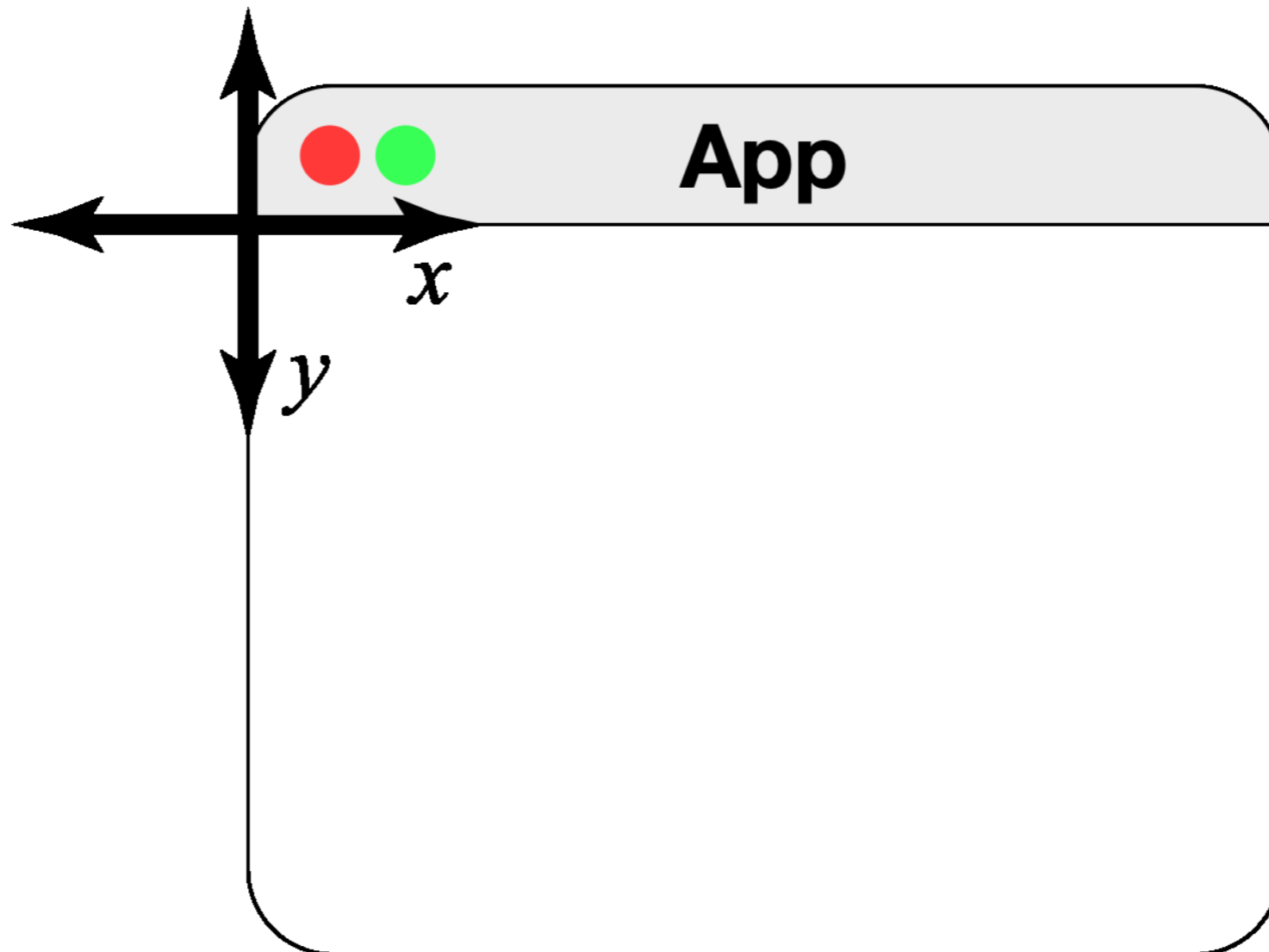


In this second model, each transformation must be applied in the transformed coordinate system that got you there!

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

```
translate( 100, 0 );
```

```
rect( 0, 0, 200, 100 );
```

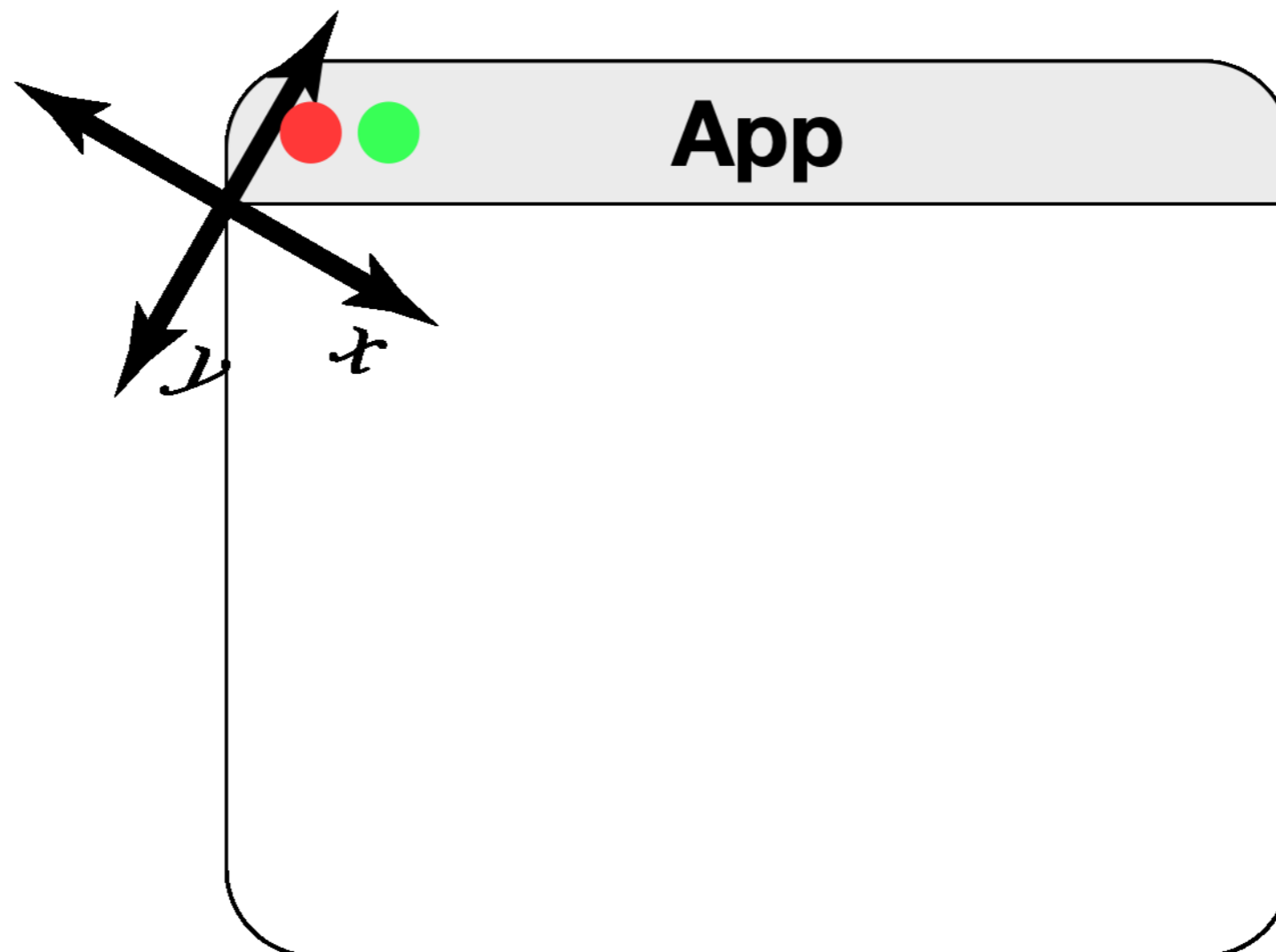


In this second model, each transformation must be applied in the transformed coordinate system that got you there!

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

```
translate( 100, 0 );
```

```
rect( 0, 0, 200, 100 );
```

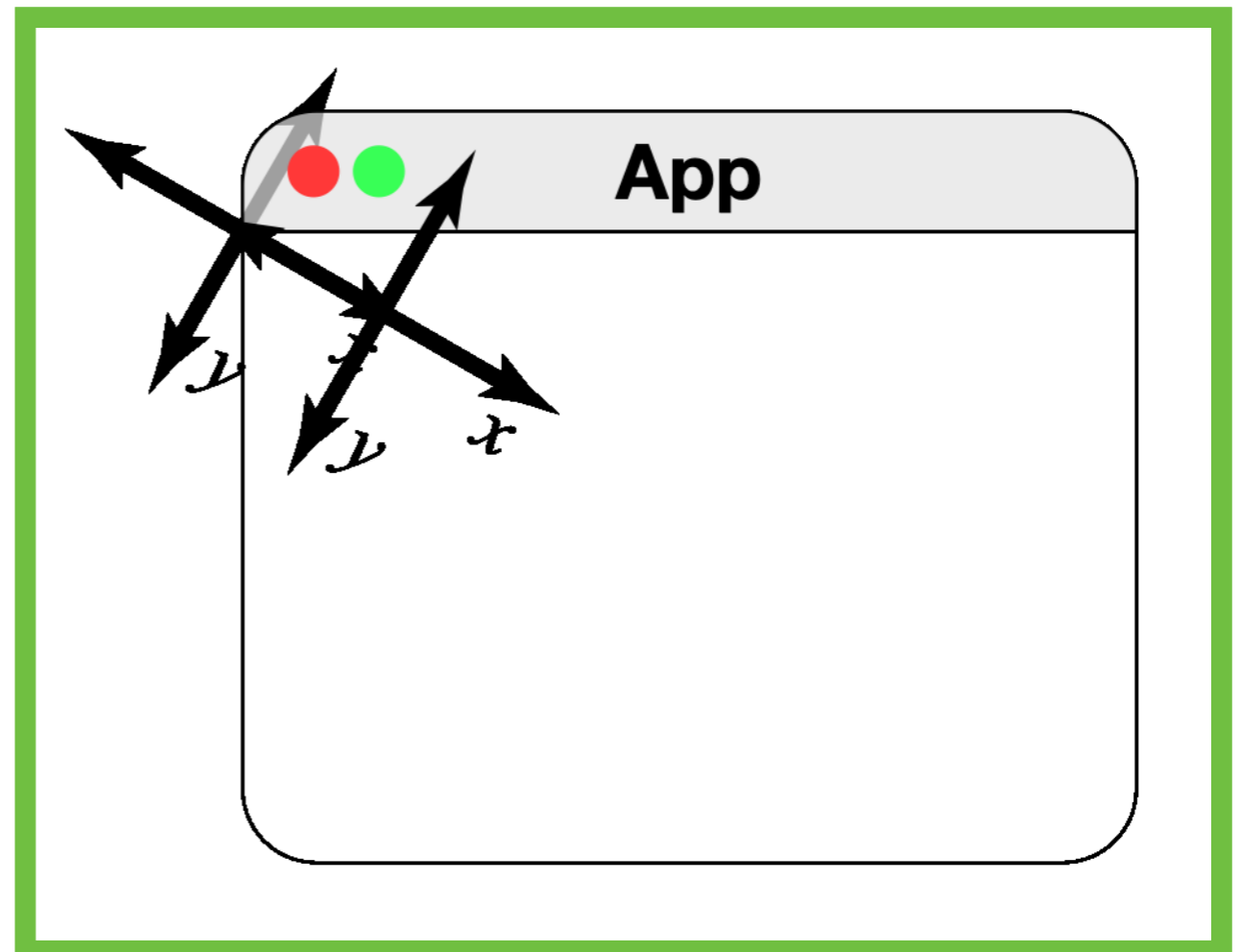
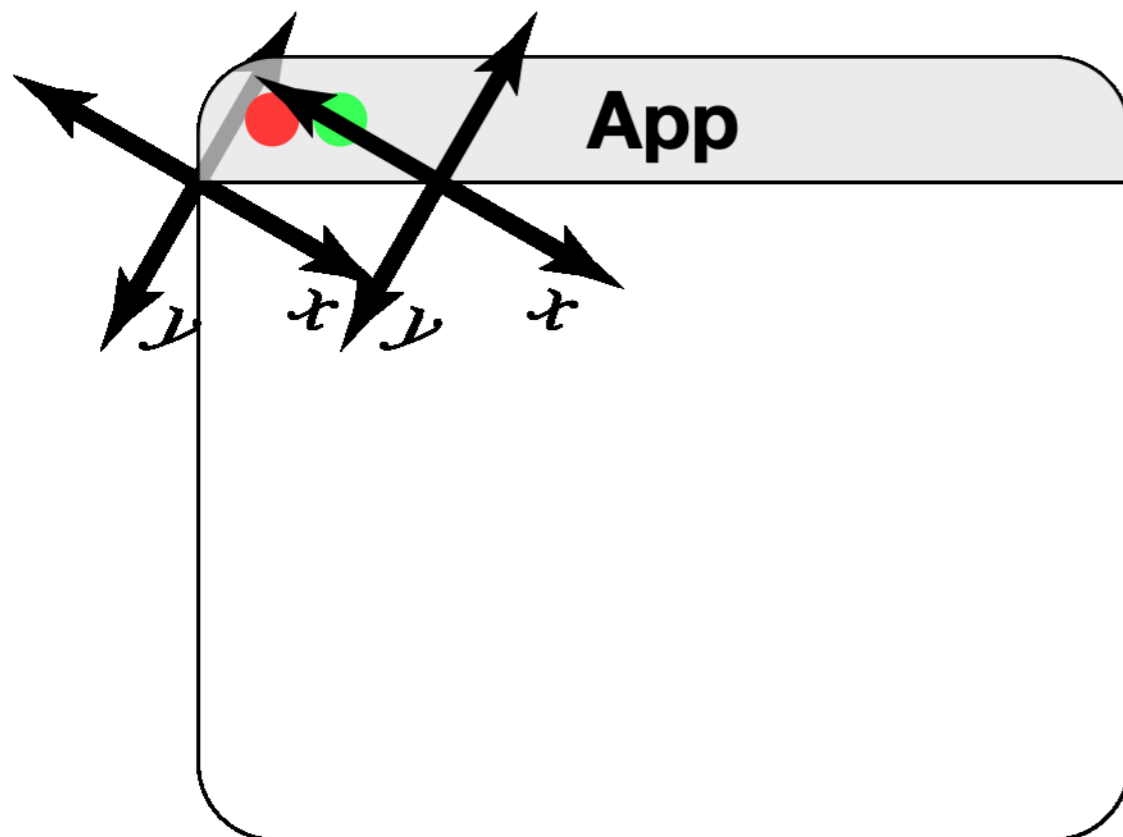


In this second model, each transformation must be applied **in the transformed coordinate system** that got you there!

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

```
translate( 100, 0 );
```

```
rect( 0, 0, 200, 100 );
```

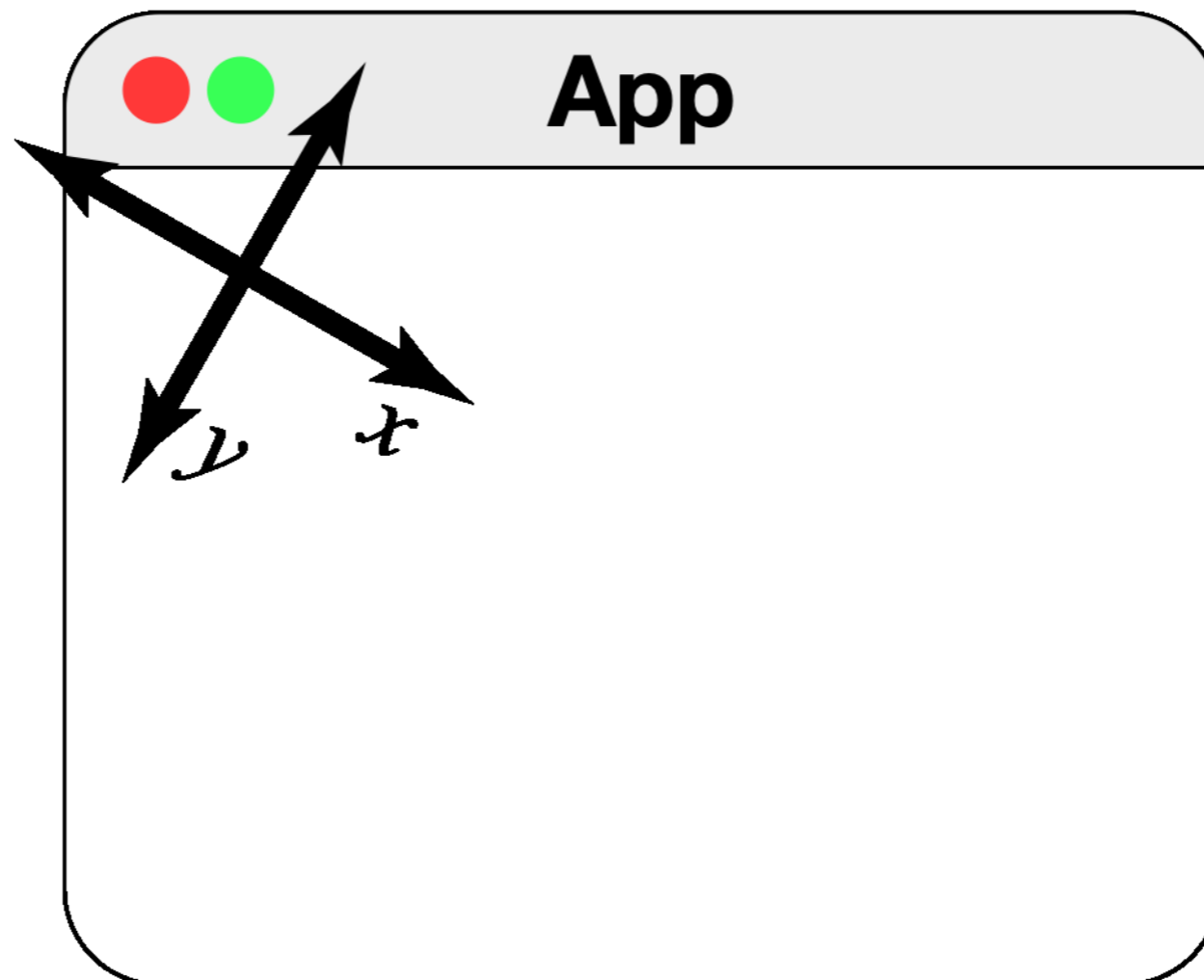


In this second model, each transformation must be applied in the transformed coordinate system that got you there!

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

```
translate( 100, 0 );
```

```
rect( 0, 0, 200, 100 );
```

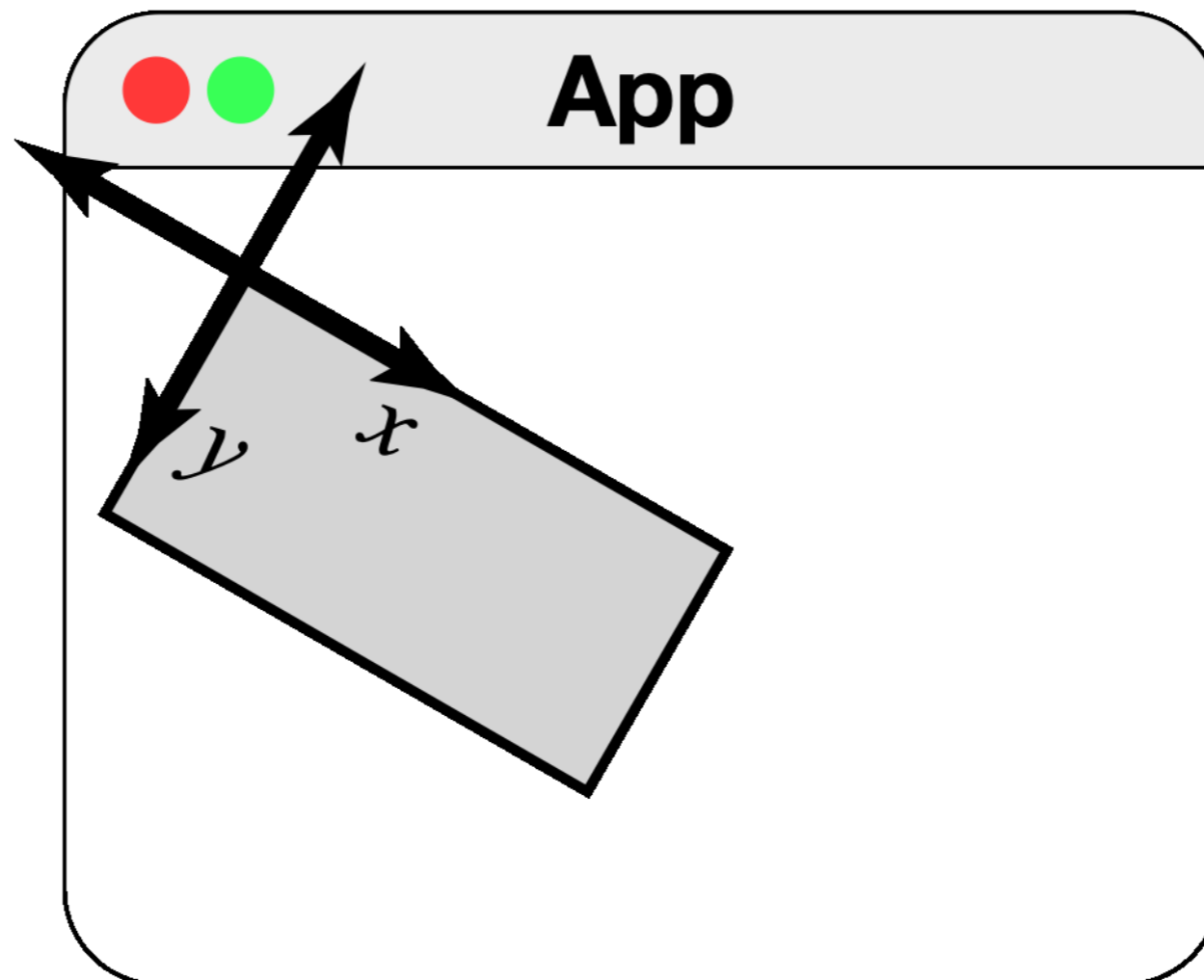


In this second model, each transformation must be applied in the transformed coordinate system that got you there!

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

```
translate( 100, 0 );
```

```
rect( 0, 0, 200, 100 );
```

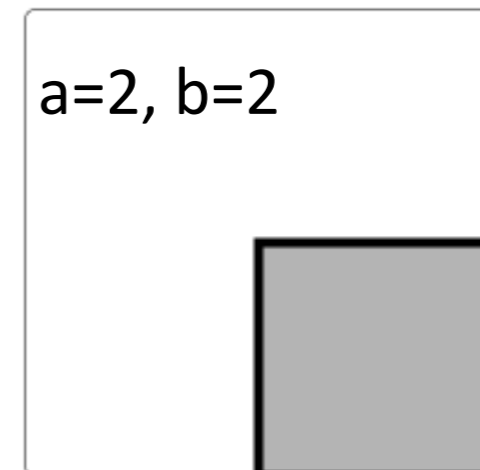
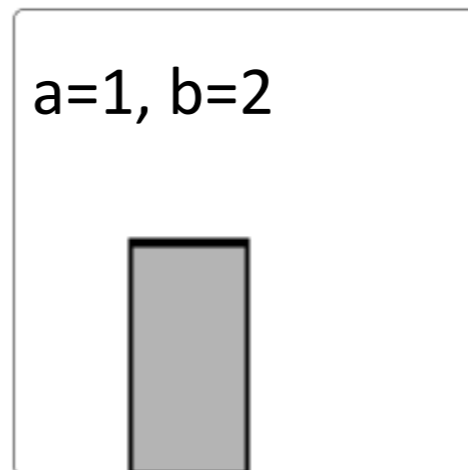
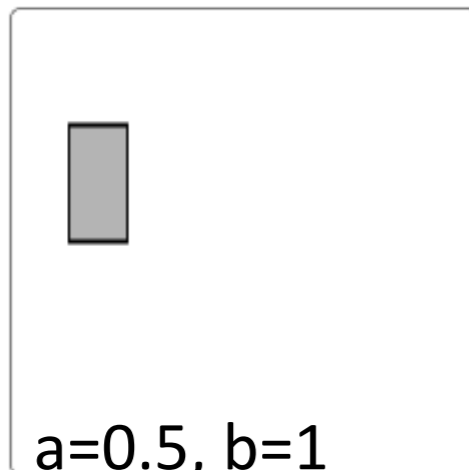
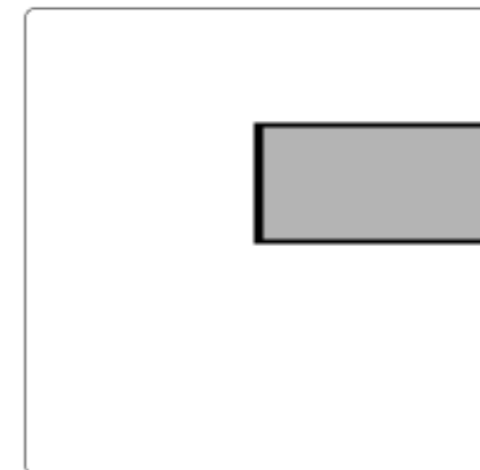
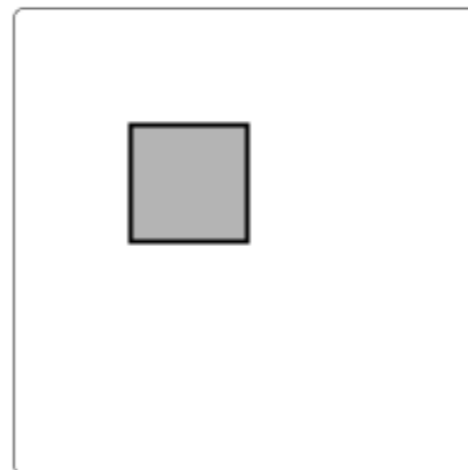
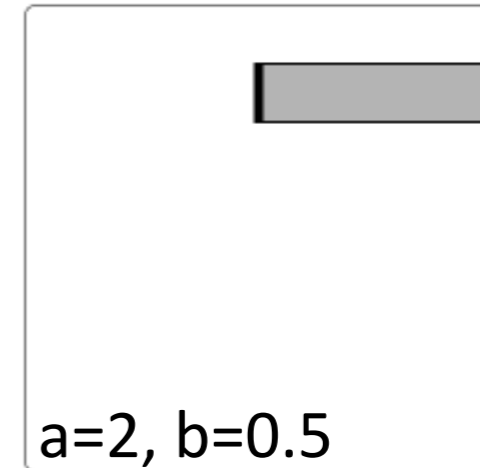
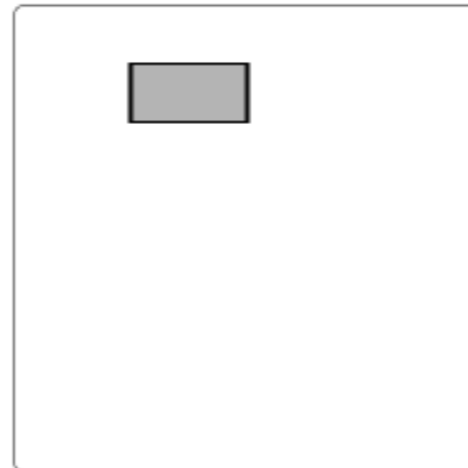
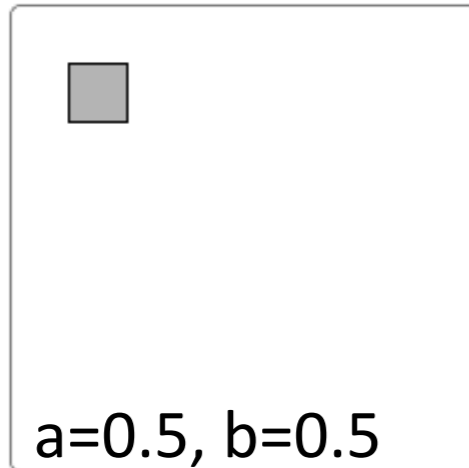


scale(a, b): Scale the current geometric context by ratios a in the x direction and b in the y direction.


```
createCanvas(200, 200);
```

```
scale(a, b);
```

```
rect(50, 50, 50, 50);
```

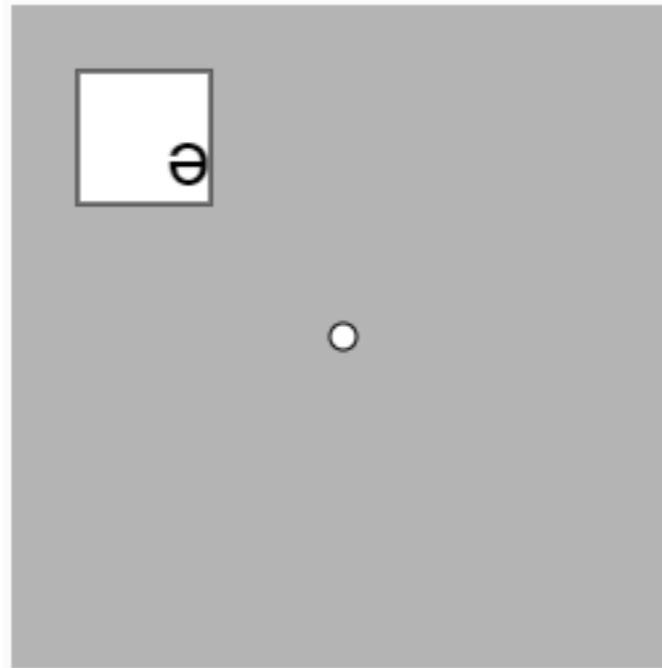


```
textAlign(LEFT, TOP);
textSize(30);
ellipseMode(CENTER);

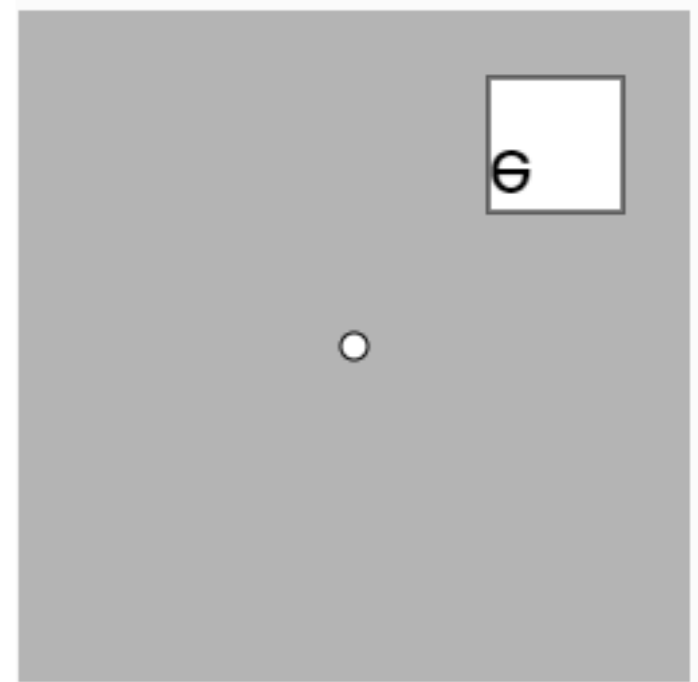
translate(height/2,
          width/2);
ellipse(0, 0, 10, 10);

scale(a, b);
rect(50, 50, 50, 50);
text("e", 50, 50);
```

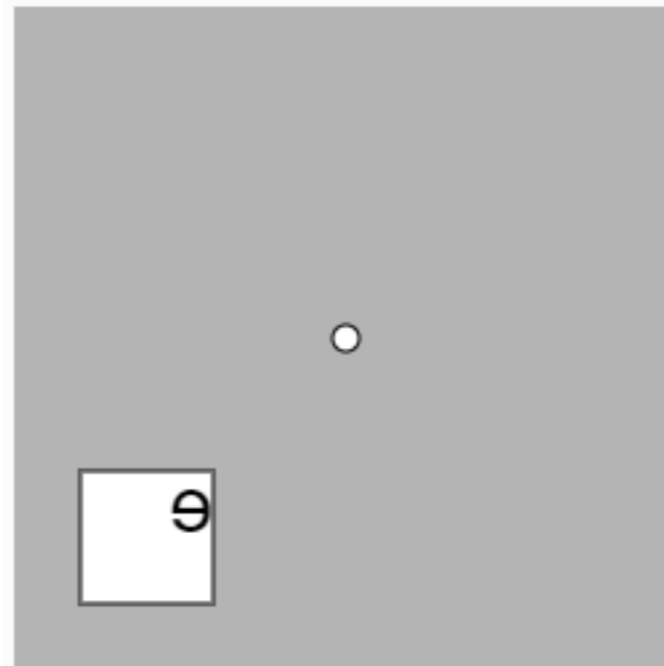
a=-1, b=-1



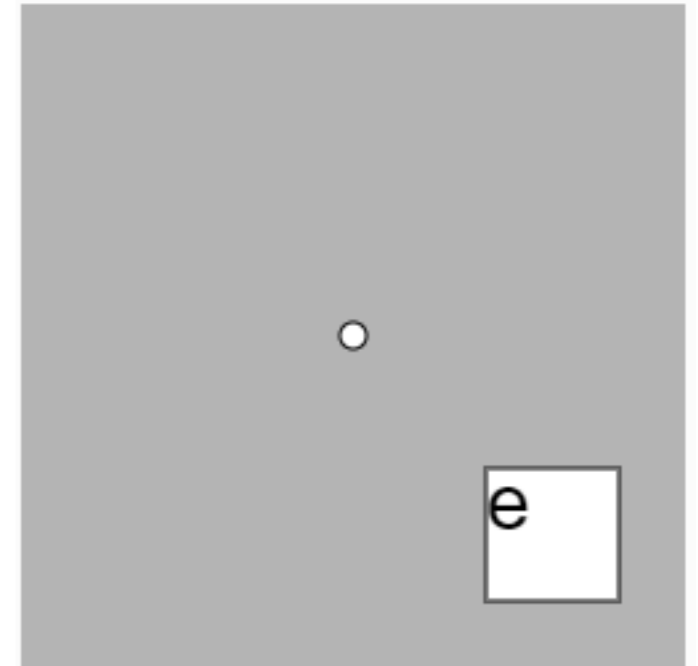
a=1, b=-1



a=-1, b=1

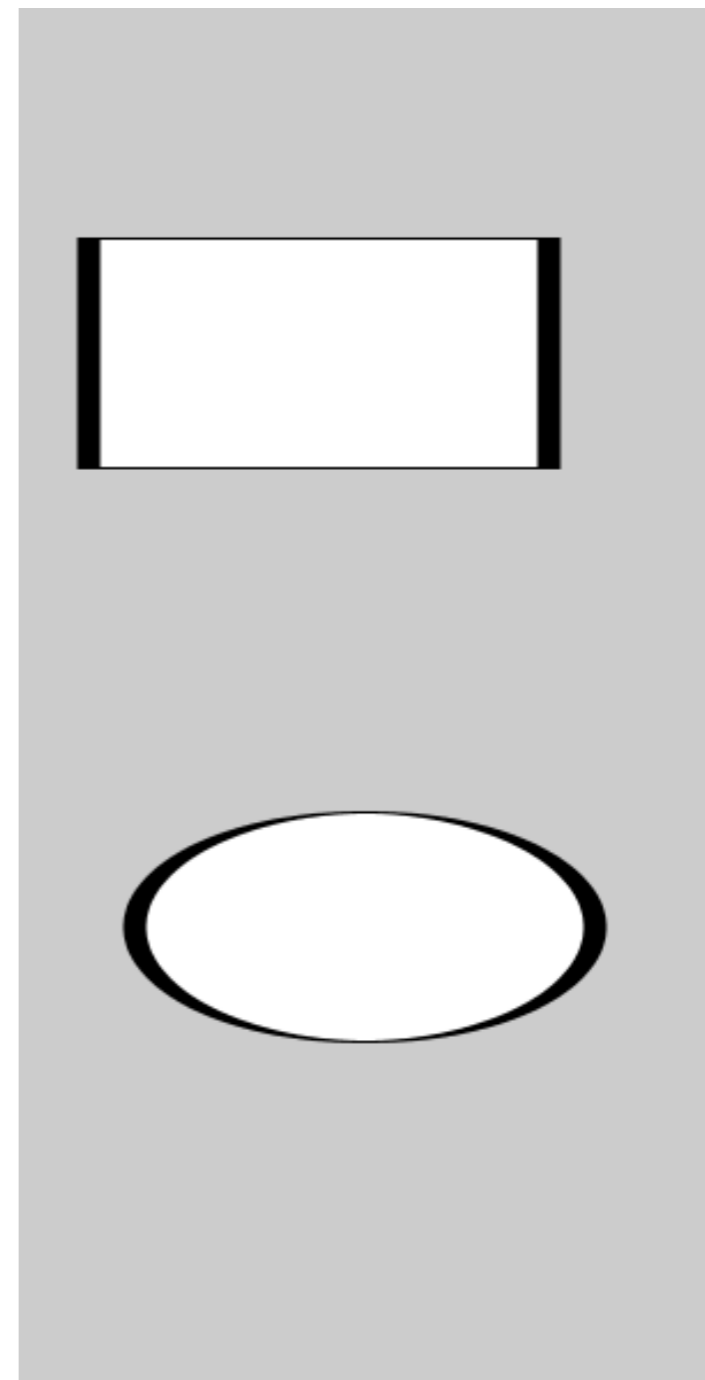


a=1, b=1



Beware: scaling affects strokes too!

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

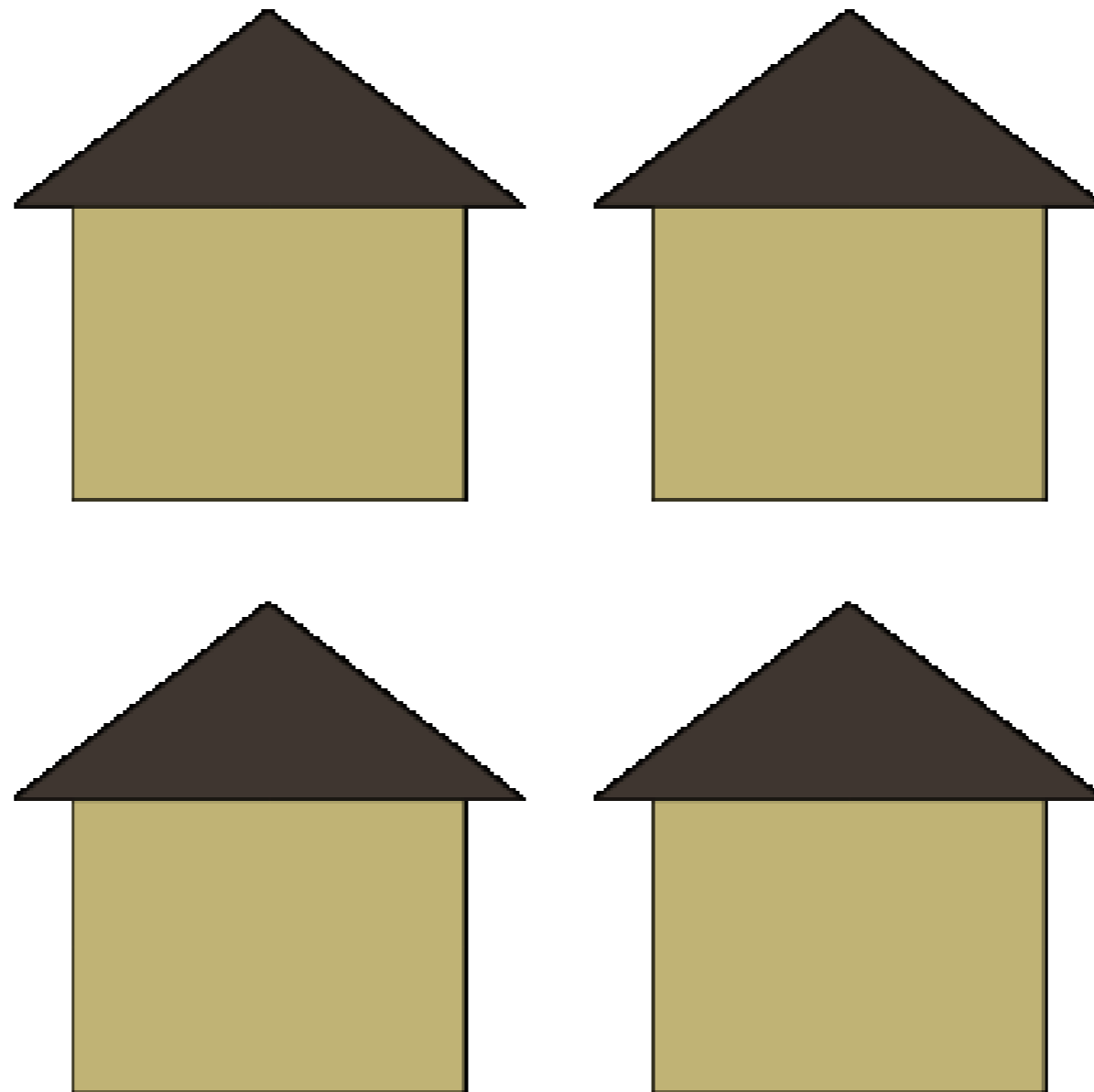


Contexts in a Hierarchy

putting *push* and *pop* to work

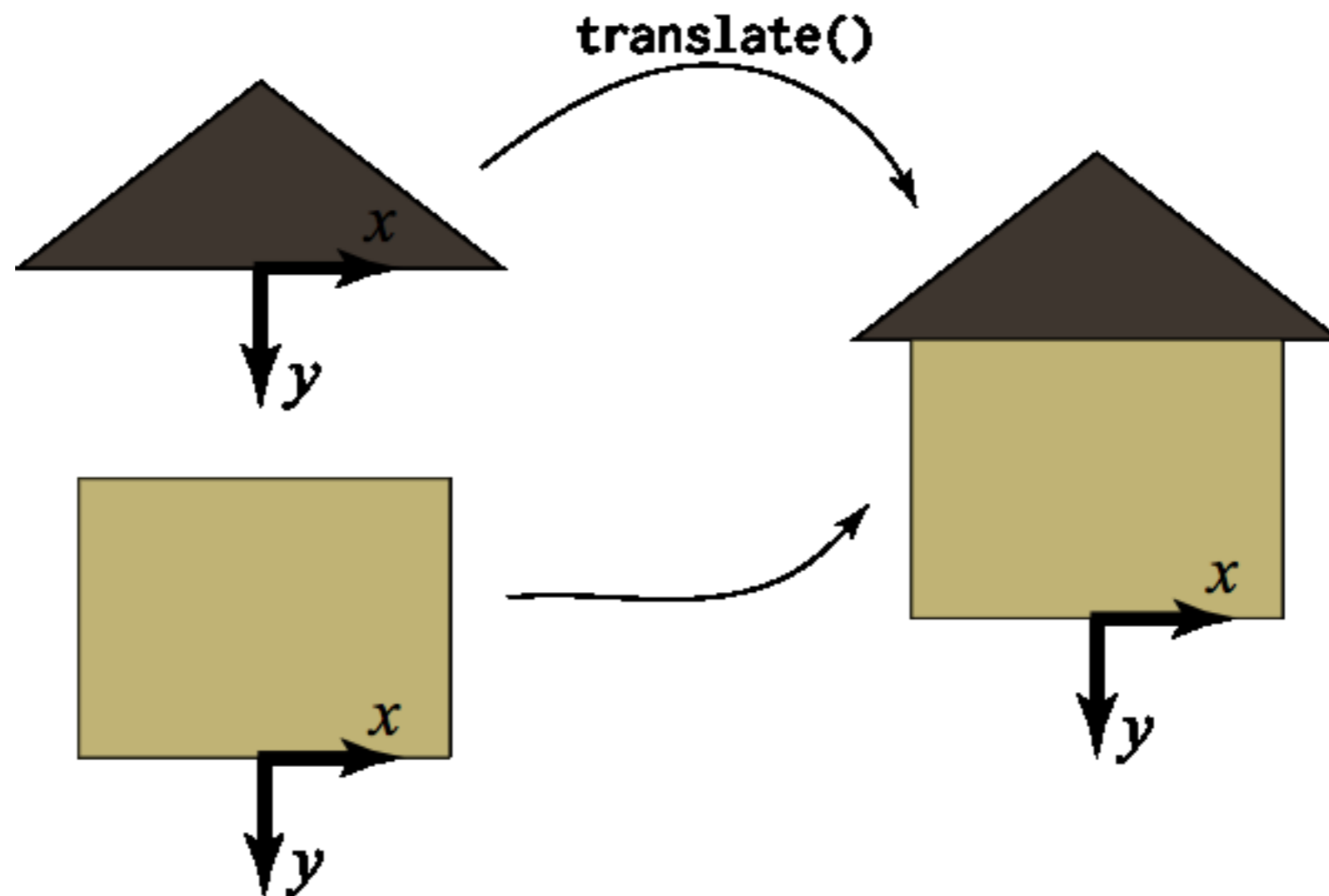
Hierarchical Modelling

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



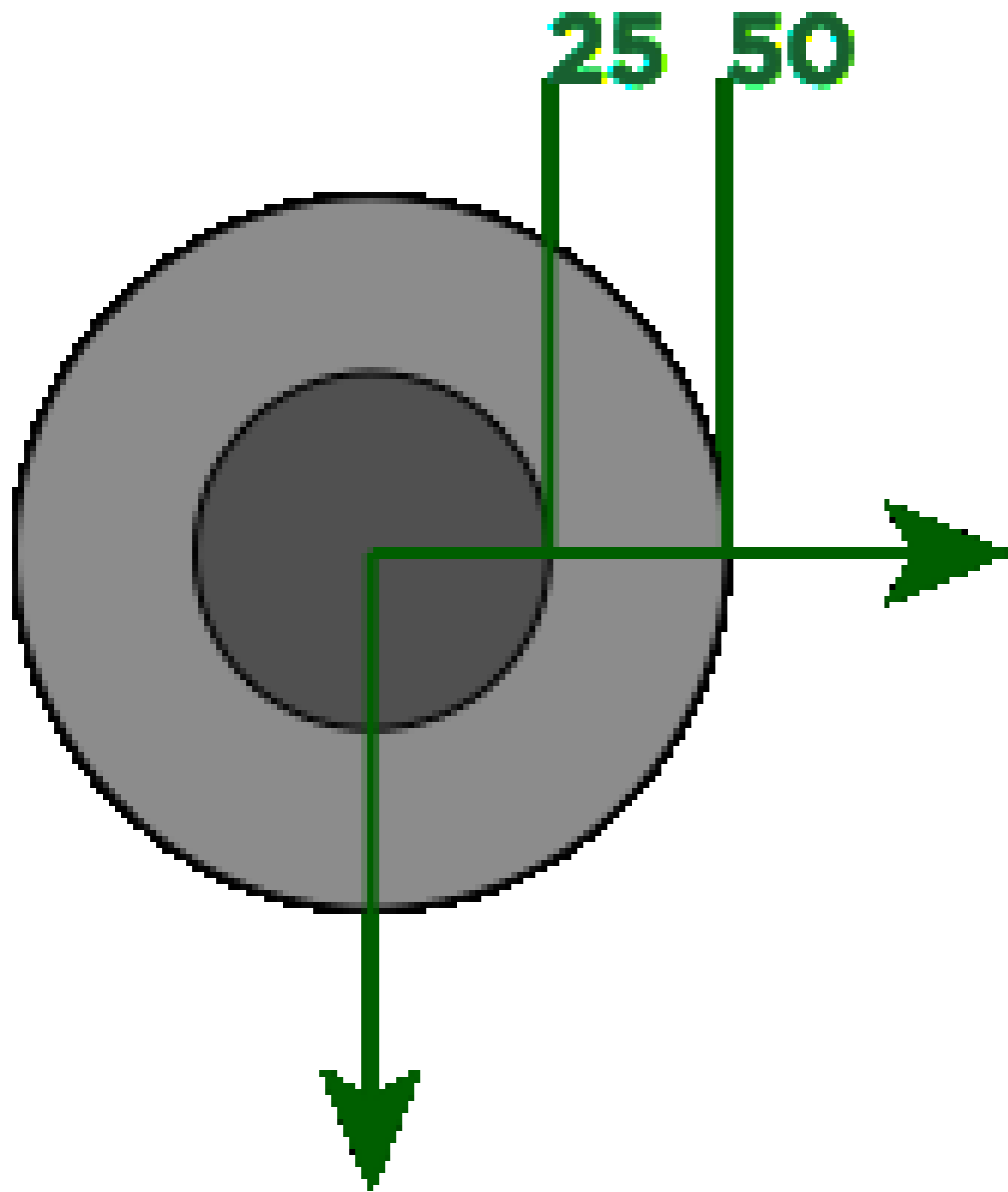
Hierarchical Modelling

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

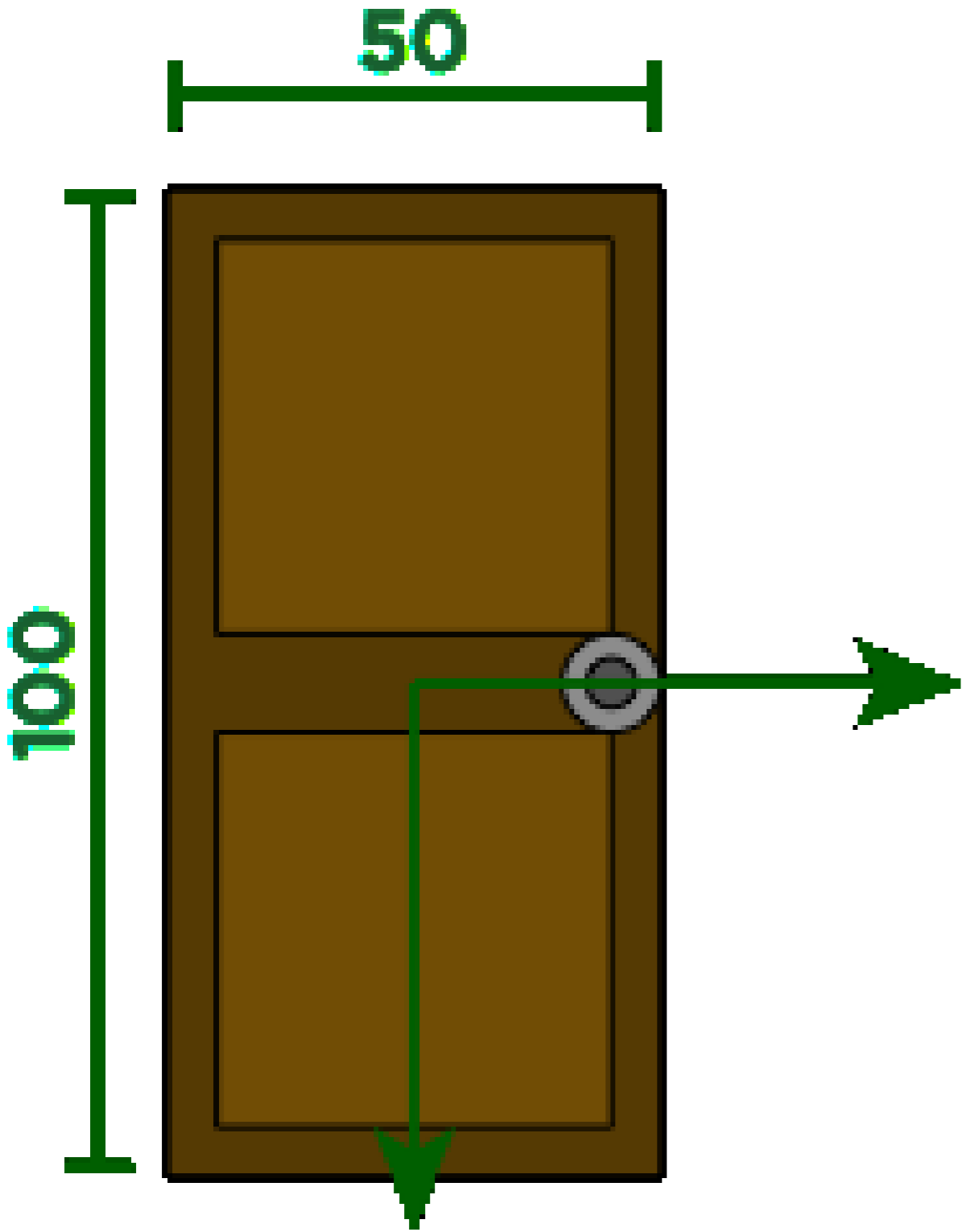


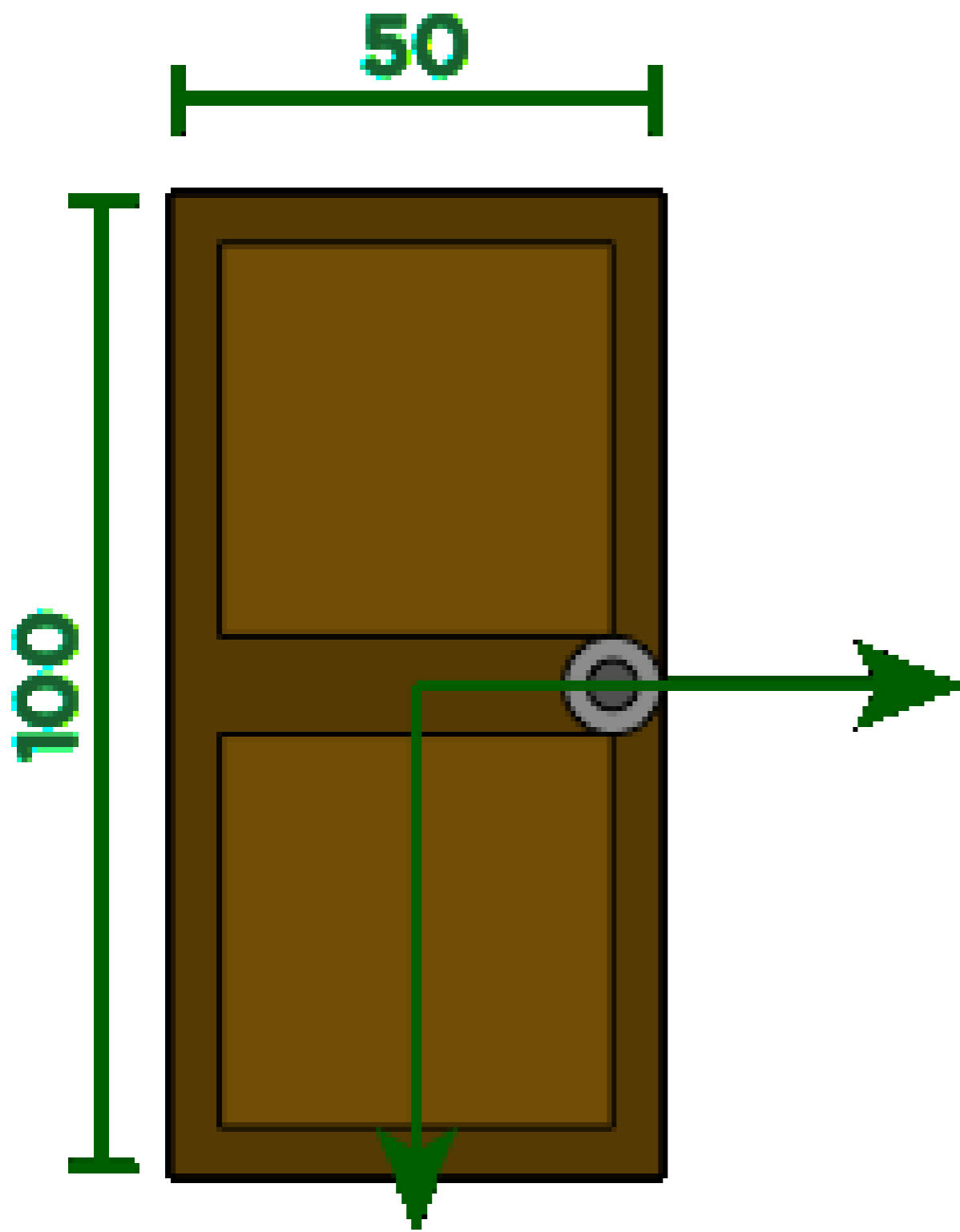
Hierarchical Modelling

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

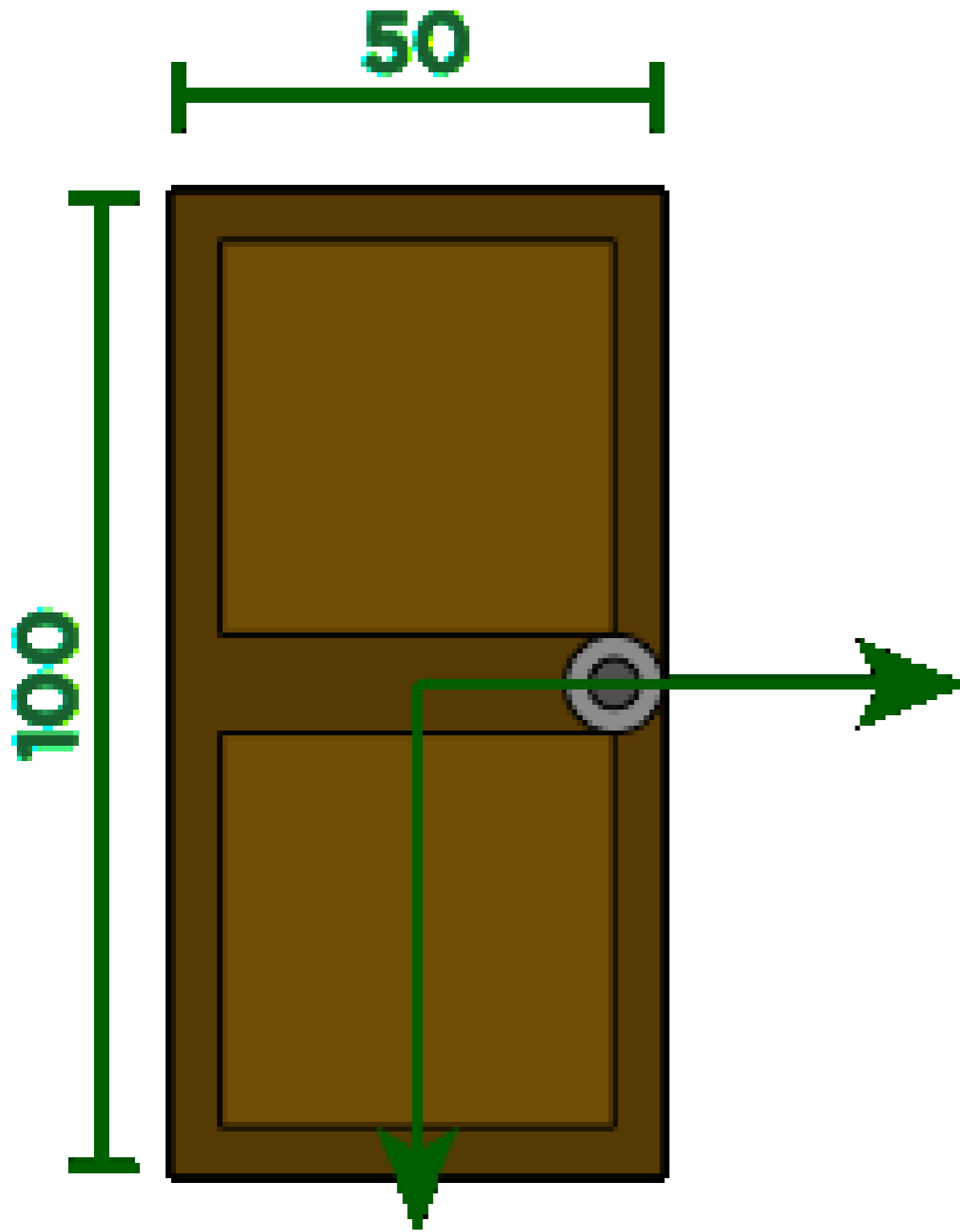


```
function doorknob() {  
  fill(140);  
  ellipse(0, 0, 100, 100);  
  fill(80);  
  ellipse(0, 0, 50, 50);  
}
```

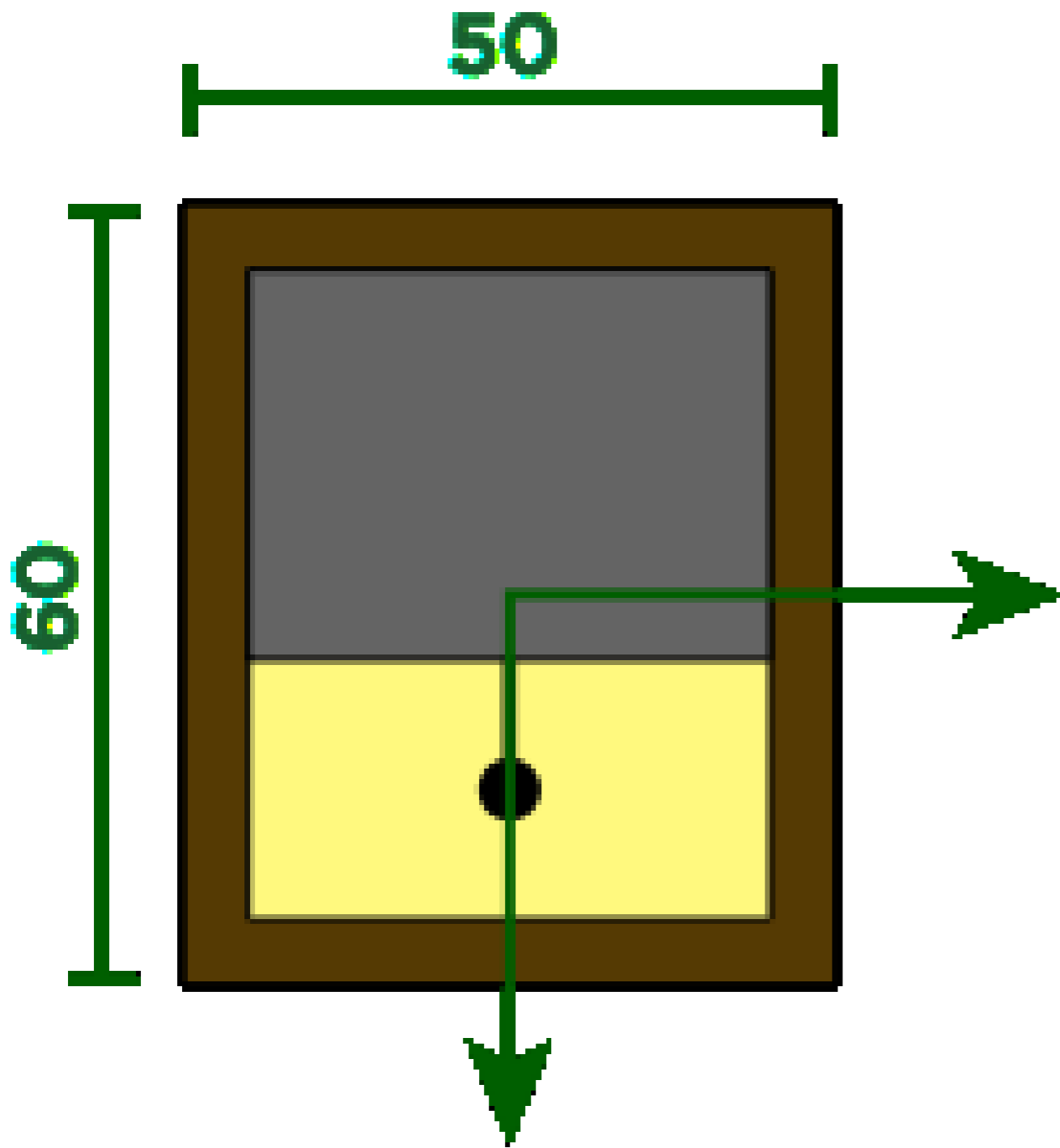





```
function door() {  
  fill("#553A03");  
  rect(-25, -50, 50, 100);  
  fill("#714D05");  
  rect(-20, -45, 40, 40);  
  rect(-20, 5, 40, 40);  
}
```



```
function door() {  
  fill("#553A03");  
  rect(-25, -50, 50, 100);  
  fill("#714D05");  
  rect(-20, -45, 40, 40);  
  rect(-20, 5, 40, 40);  
  
  push();  
  translate(20, 0);  
  scale(0.1);  
  doorknob();  
  pop();  
}
```



```
function 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);  
}
```

