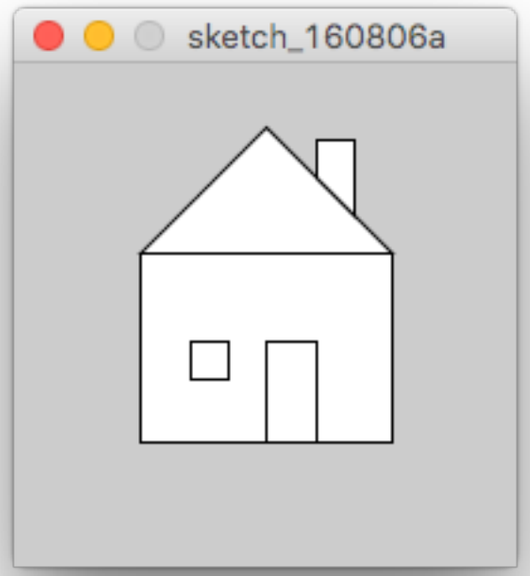
Processing Transformations

Affine Transformations

A house

```
size(200, 200);

rect(50, 75, 100, 75); // (left, top, w, h)
rect(100, 110, 20, 40);   // door
rect(70, 110, 15, 15);    //window
triangle(50, 75, 100, 25, 150, 75); // roof
quad(120, 45, 120, 30, 135, 30, 135, 60); // chimney
```



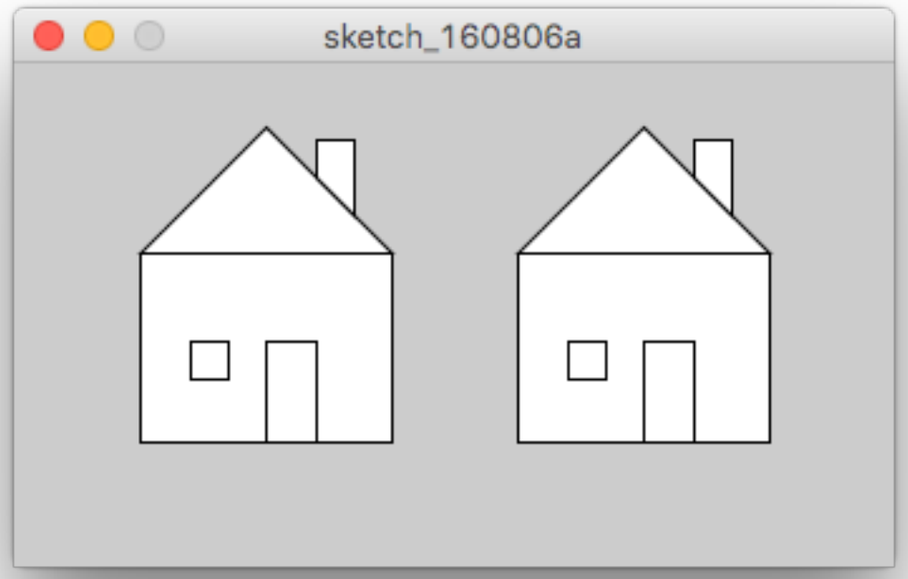
Two houses

```
size(350, 200);

rect(50, 75, 100, 75); // (left, top, w, h)
rect(100, 110, 20, 40);   // door
rect(70, 110, 15, 15);    //window
triangle(50, 75, 100, 25, 150, 75); // roof
quad(120, 45, 120, 30, 135, 30, 135, 60); // chimney

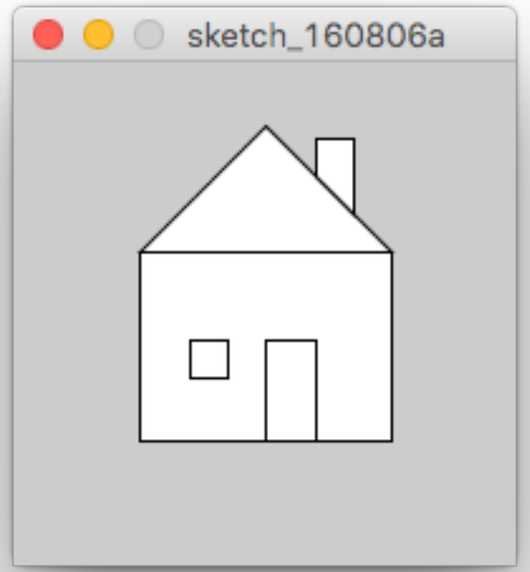
rect(200, 75, 100, 75); // (left, top, w, h)
rect(250, 110, 20, 40);   // door
rect(220, 110, 15, 15);    //window
triangle(200, 75, 250, 25, 300, 75); // roof
quad(270, 45, 270, 30, 285, 30, 285, 60); // chimney
```

- Notice that the numbers in red needed to be changed
- This is annoying and very error-prone
- Wouldn't it be nice if there were a better way?



Methods

- The other way to write a Processing program is as a collection of methods
- There must be a `setup()` method, which is where the program starts
  - The `setup` method typically contains calls to `size`, `background`, and methods that you write

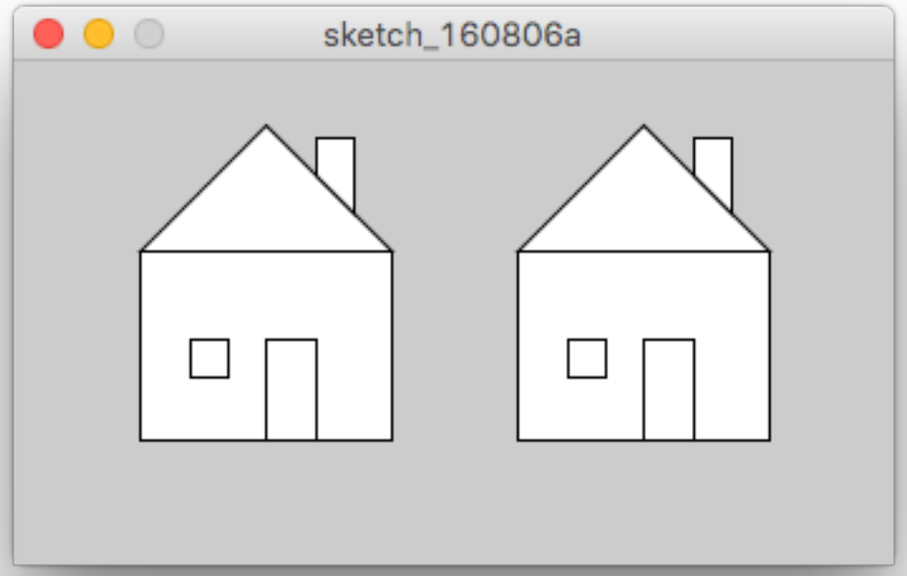



```
void setup() {
  size(200, 200);
  house();
}

void house() {
  rect(50, 75, 100, 75); // (left, top, w, h)
  rect(100, 110, 20, 40);   // door
  rect(70, 110, 15, 15);    //window
  triangle(50, 75, 100, 25, 150, 75); // roof
  quad(120, 45, 120, 30, 135, 30, 135, 60); // chimney
}
```

Translating manually

- **Translation**: the process of moving something from one place to another.
- A Processing program can be written as a collection of methods
- There must be a `setup()` method, which is where the program starts
  - The `setup` method typically contains calls to `size`, `background`, and methods that you write

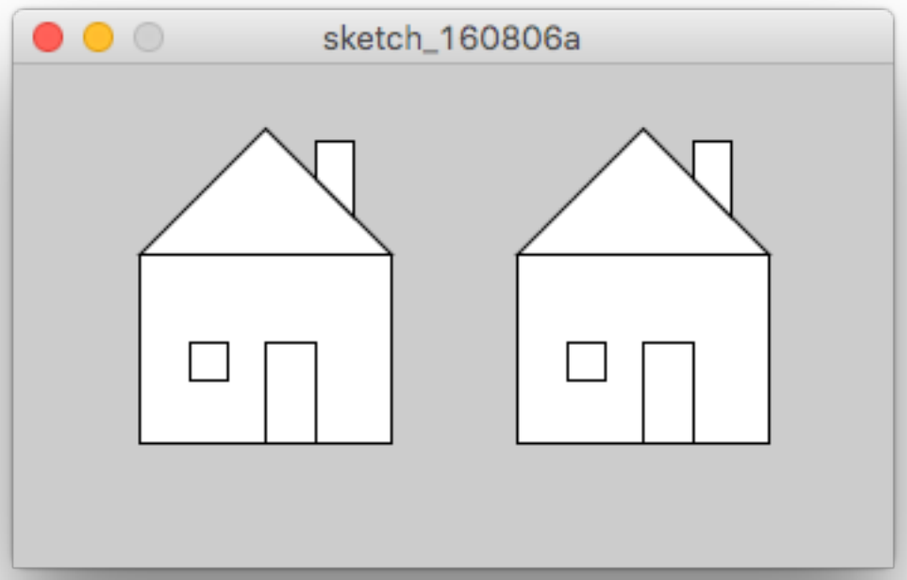



```
void setup() {
  size(350, 200);
  house(0);
  house(150);
}

void house(int dx) {
  rect(50+dx, 75, 100, 75); // (left, top, w, h)
  rect(100+dx, 110, 20, 40);   // door
  rect(70+dx, 110, 15, 15);    //window
  triangle(50+dx, 75, 100+dx, 25, 150+dx, 75); // roof
  quad(120+dx, 45, 120+dx, 30, 135+dx, 30, 135+dx, 60); // chimney
}
```

Translating the easy way

```
void setup() {
  size(350, 200);
  house();
  translate(150, 0); // (x, y)
  house();
}
```

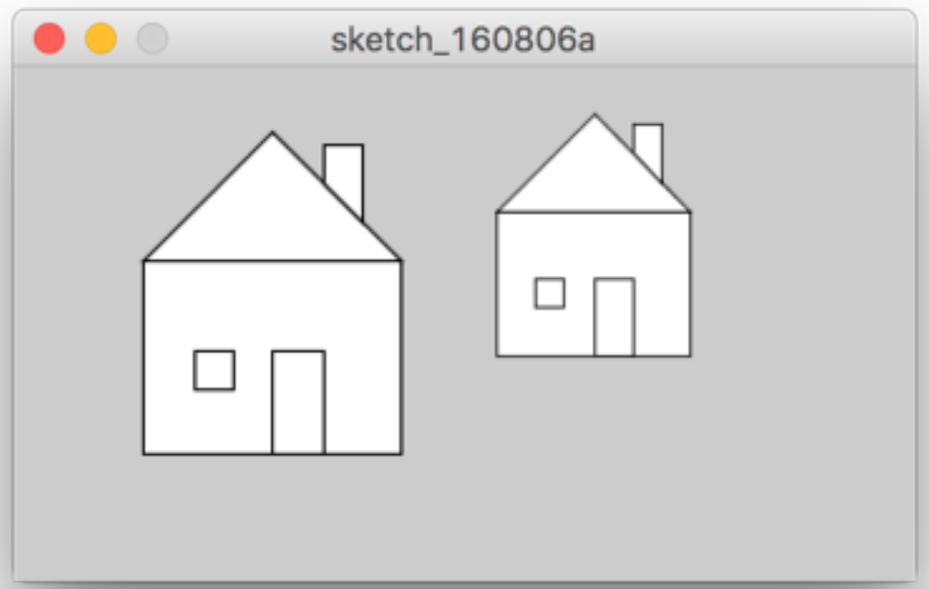



```
void house() { // unchanged from the original
  rect(50, 75, 100, 75); // (left, top, w, h)
  rect(100, 110, 20, 40);   // door
  rect(70, 110, 15, 15);    //window
  triangle(50, 75, 100, 25, 150, 75); // roof
  quad(120, 45, 120, 30, 135, 30, 135, 60); // chimney
}
```

- The `translate` method changes where the **origin** (0, 0) is located

Changing the size

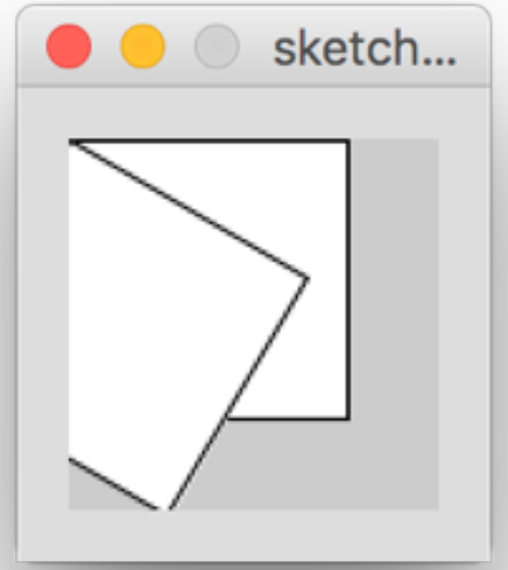
```
void setup() {
  size(350, 200);
  house();
  translate(150, 0);
  scale(0.75);
  house();
}
```




```
void house() { // unchanged from the original
  rect(50, 75, 100, 75); // (left, top, w, h)
  rect(100, 110, 20, 40);   // door
  rect(70, 110, 15, 15);    //window
  triangle(50, 75, 100, 25, 150, 75); // roof
  quad(120, 45, 120, 30, 135, 30, 135, 60); // chimney
}
```

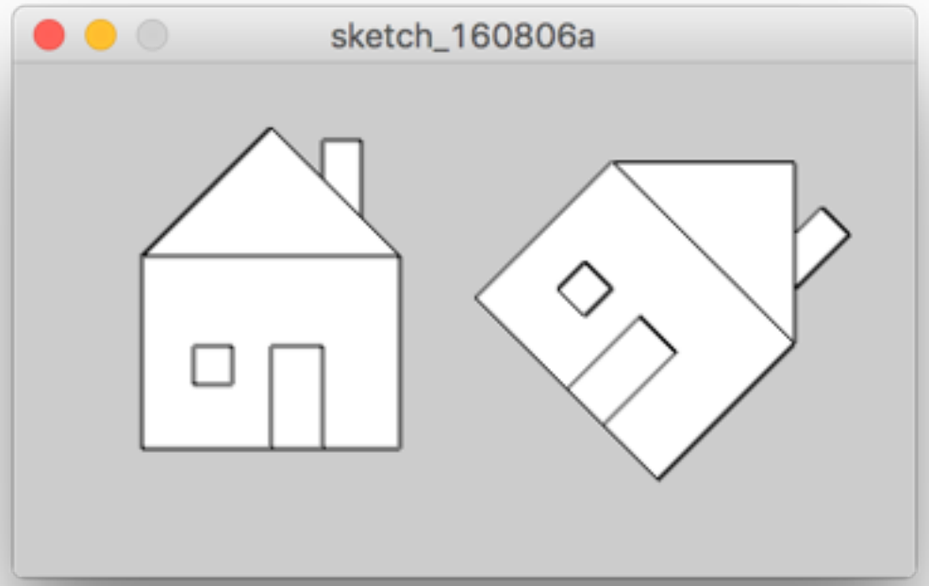
Rotating

- Rotating happens around the **origin**, which is initially the top left corner
- ```
  rect(0, 0, 75, 75);
  rotate(radians(30));
  rect(0, 0, 75, 75);
  ```
- Remember, the `translate` method changes the location of the origin


  ```

Translating and rotating

- Since figures rotate around the origin, you may want to translate the rotated figure



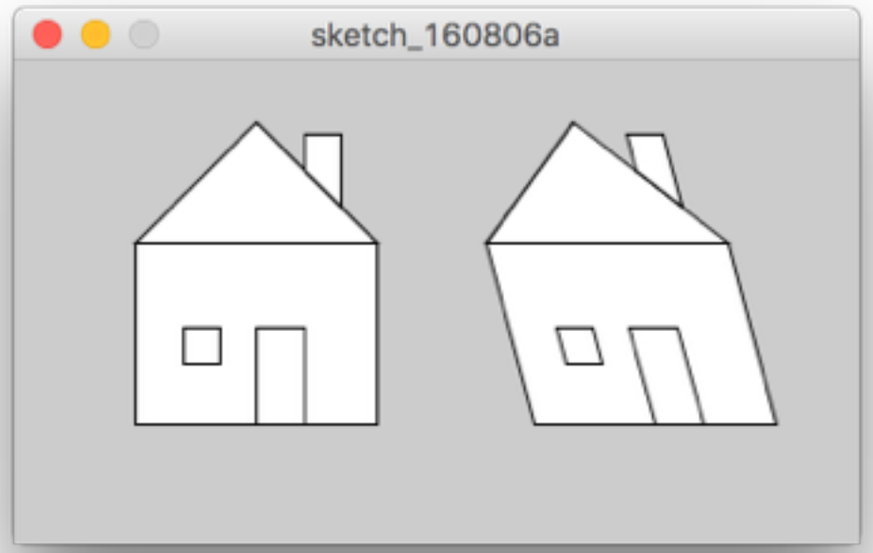

- ```
  void setup() {
    size(350, 200);
    house();
    translate(250, -50);
    rotate(QUARTER_PI);
    house();
  }

  void house() { // unchanged from the original
    rect(50, 75, 100, 75); // (left, top, w, h)
    rect(100, 110, 20, 40);   // door
    rect(70, 110, 15, 15);    //window
    triangle(50, 75, 100, 25, 150, 75); // roof
    quad(120, 45, 120, 30, 135, 30, 135, 60); // chimney
  }
  ```

- Note: `translate` followed by `rotate` will give different results than `rotate` followed by `translate`!
  ```

Shearing

- In addition to translating, scaling, and rotating, you can also shear along either the X or the Y axis




```
void setup() {
  size(350, 200);
  house();
  translate(125, 0);
  shearX(radians(15));
  house();
}

void house() { // unchanged from the original
  rect(50, 75, 100, 75); // (left, top, w, h)
  rect(100, 110, 20, 40);   // door
  rect(70, 110, 15, 15);    //window
  triangle(50, 75, 100, 25, 150, 75); // roof
  quad(120, 45, 120, 30, 135, 30, 135, 60); // chimney
}
```

- Here is an example of converting 15° to radians
- It should be easy to figure out what `shearY` does

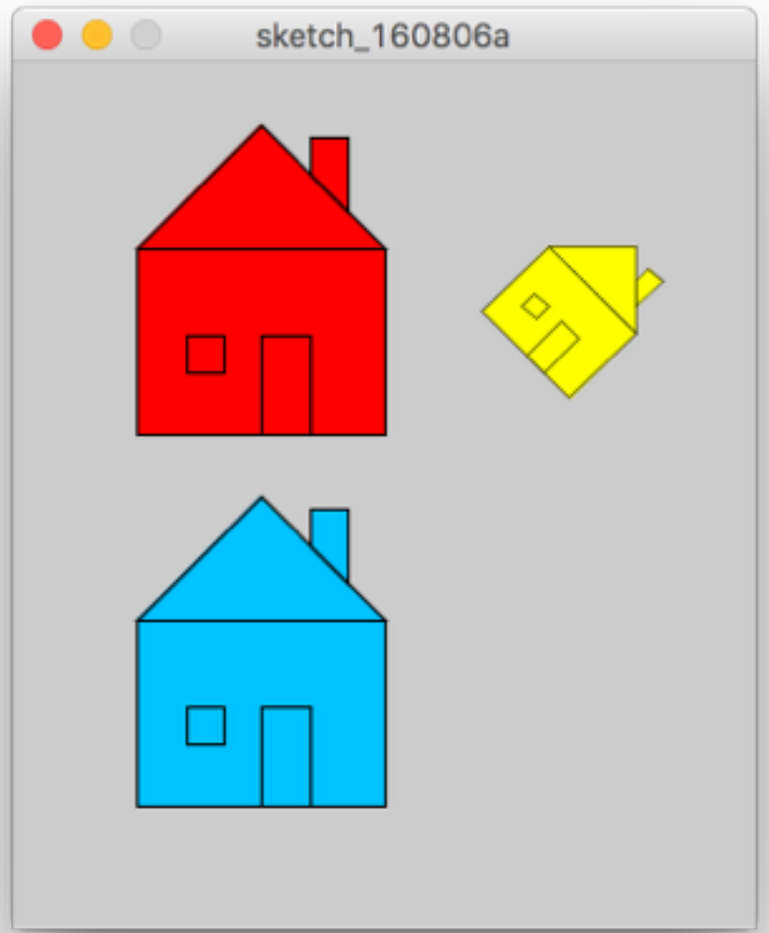
Matrices

- The `translate`, `scale`, `rotate`, `shearX` and `shearY` methods are all matrix transformations
  - Specifically, they are “affine” translations
  - A Processing program starts with an initial default matrix
- You can save the current matrix in a *stack* with the `pushMatrix()` method
  - You can “push” as many things as you like onto a stack; when you “pop” them out, you get the most recently pushed items first
- You can use `popMatrix()` to take the most recently “pushed” matrix from the stack and apply it
- There is also a `resetMatrix()` method to return to the initial default matrix settings

Pushing and popping matrices

```
void setup() {
  size(300, 350);
  fill(255, 0, 0);      // red
  house();
  pushMatrix();
  translate(225, 30);
  scale(0.50);
  rotate(QUARTER_PI);
  fill(255, 255, 0);  // yellow
  house();
  popMatrix();
  translate(0, 150);
  fill(0, 196, 255);  // blue-ish
  house();
}

void house() { // unchanged from the original
  rect(50, 75, 100, 75); // (left, top, w, h)
  rect(100, 110, 20, 40);   // door
  rect(70, 110, 15, 15);    //window
  triangle(50, 75, 100, 25, 150, 75); // roof
  quad(120, 45, 120, 30, 135, 30, 135, 60); // chimney
}
```



Gradients

- Processing does not provide built-in gradients, but with a little effort you can create them
- The `lerpColor(`***color1***`,` ***color2***`,` ***proportion***`)` computes a "weighted average" of two colors
  - If ***proportion*** = 0.0, you get ***color1***; if ***proportion*** = 1.0, you get ***color2***; ***proportion***=0.5 gives you a color halfway between; and so on
- On the next slide, I've provided a `linearGradient` method for filling a rectangle with a gradient
  - It does this by drawing many parallel lines adjacent to each other, and each a slightly different color
  - The first four arguments are the same as for `rect`
  - The next two arguments are the initial and final colors
  - The last argument is the direction of the gradient, one of `"up"`, `"down"`, `"left"`, or `"right"`

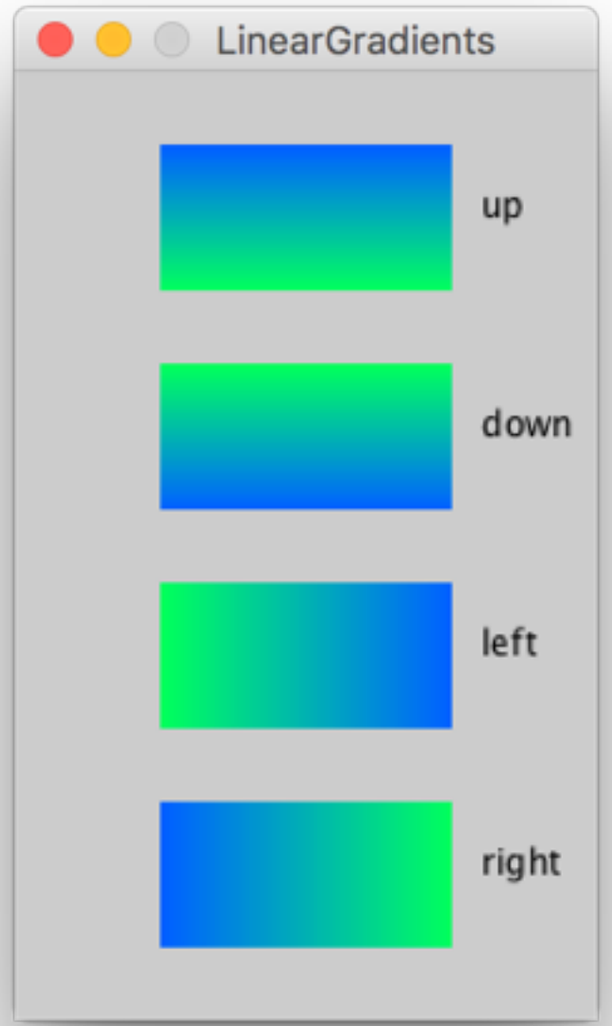
My `linearGradient` method

- ```
  void linearGradient(float x, float y,
                      float w, float h,
                      color c1, color c2,
                      String dir) {
    if (dir.equals("down")) {
      for (float dy = 0; dy < h; dy += 1) {
        stroke(lerpColor(c1, c2, dy / h));
        line(x, y + dy, x + w - 1, y + dy);
      }
    } else if (dir.equals("up")) {
        for (float dy = 0; dy < h; dy += 1) {
          stroke(lerpColor(c2, c1, dy / h));
          line(x, y + dy, x + w - 1, y + dy);
      }
    } else if (dir.equals("left")) {
        for (float dx = 0; dx < w; dx += 1) {
          stroke(lerpColor(c1, c2, dx / w));
          line(x + dx, y, x + dx, y + h - 1);
        }
    } else if (dir.equals("right")) {
        for (float dx = 0; dx < w; dx += 1) {
          stroke(lerpColor(c2, c1, dx / w));
          line(x + dx, y, x + dx, y + h - 1);
      }
    }
  }
  ```
  ```

Using the `linearGradient` method

```
void setup() {
  size(200, 325);
  gradientRect(25, "up");
  gradientRect(100, "down");
  gradientRect(175, "left");
  gradientRect(250, "right");
}

void gradientRect(float y, String dir) {
  color c1 = color(0, 255, 100); //
green to blue
  color c2 = color(0, 100, 255);
  linearGradient(50, y, 100, 50,
                 c1, c2, dir);
  fill(0, 0, 0);
  text(dir, 160, y + 25);
}
```



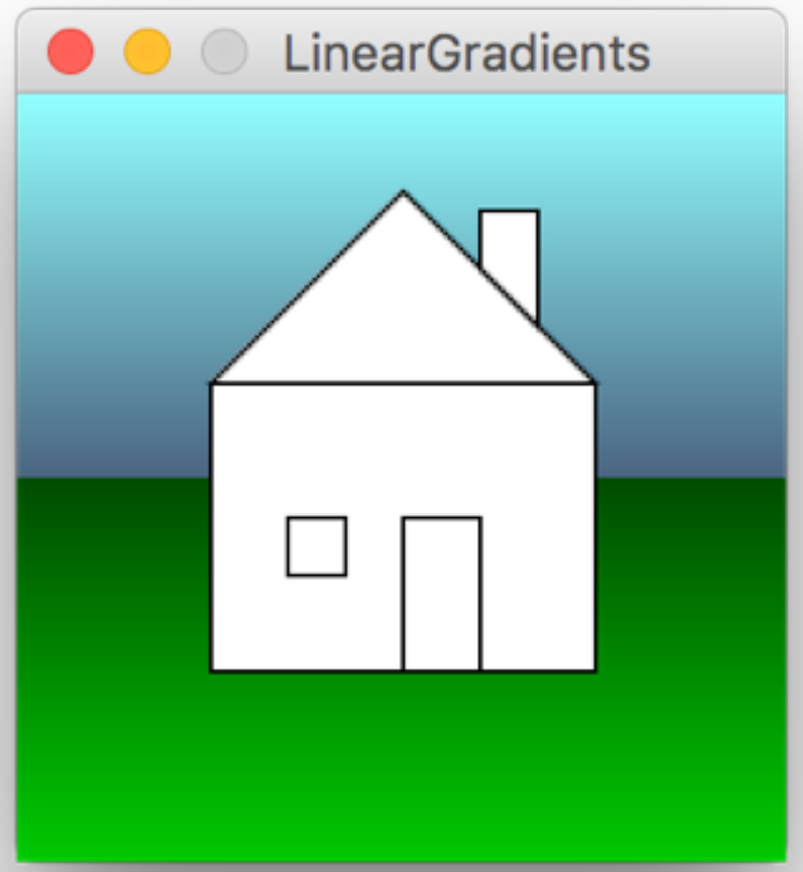
Part of `linearGradient`

```
if (dir.equals("down")) {
  for (float dy = 0; dy < h; dy += 1) {
    stroke(lerpColor(c1, c2, dy / h));
    line(x, y + dy, x + w - 1, y + dy);
  }
}
```

- This draws a gradient from `c1` at the top to `c2` at the bottom, so we draw horizontal lines starting from `y+0` and going to `y+h` (where `h` is the height of the rectangle)
- The corresponding color will go from `0/h` to `h/h`
  - Actually, the color only gets to `(h-1)/h`, which is "close enough"
  - Fixing this without making the rectangle taller is left as an exercise for the student
- The other three directions are very similar

House with gradients

```
void setup() {
  size(200, 200);
  color lightBlue = color(150, 255, 255);
  color darkBlue = color(75, 100, 128);
  linearGradient(0, 0, width, height / 2,
                 lightBlue, darkBlue, "down");
  color lightGreen = color(0, 200, 0);
  color darkGreen = color(0, 75, 0);
  linearGradient(0, height / 2, width,
                 height / 2, lightGreen,
                 darkGreen, "up");
  stroke(0);
  house();
}
```



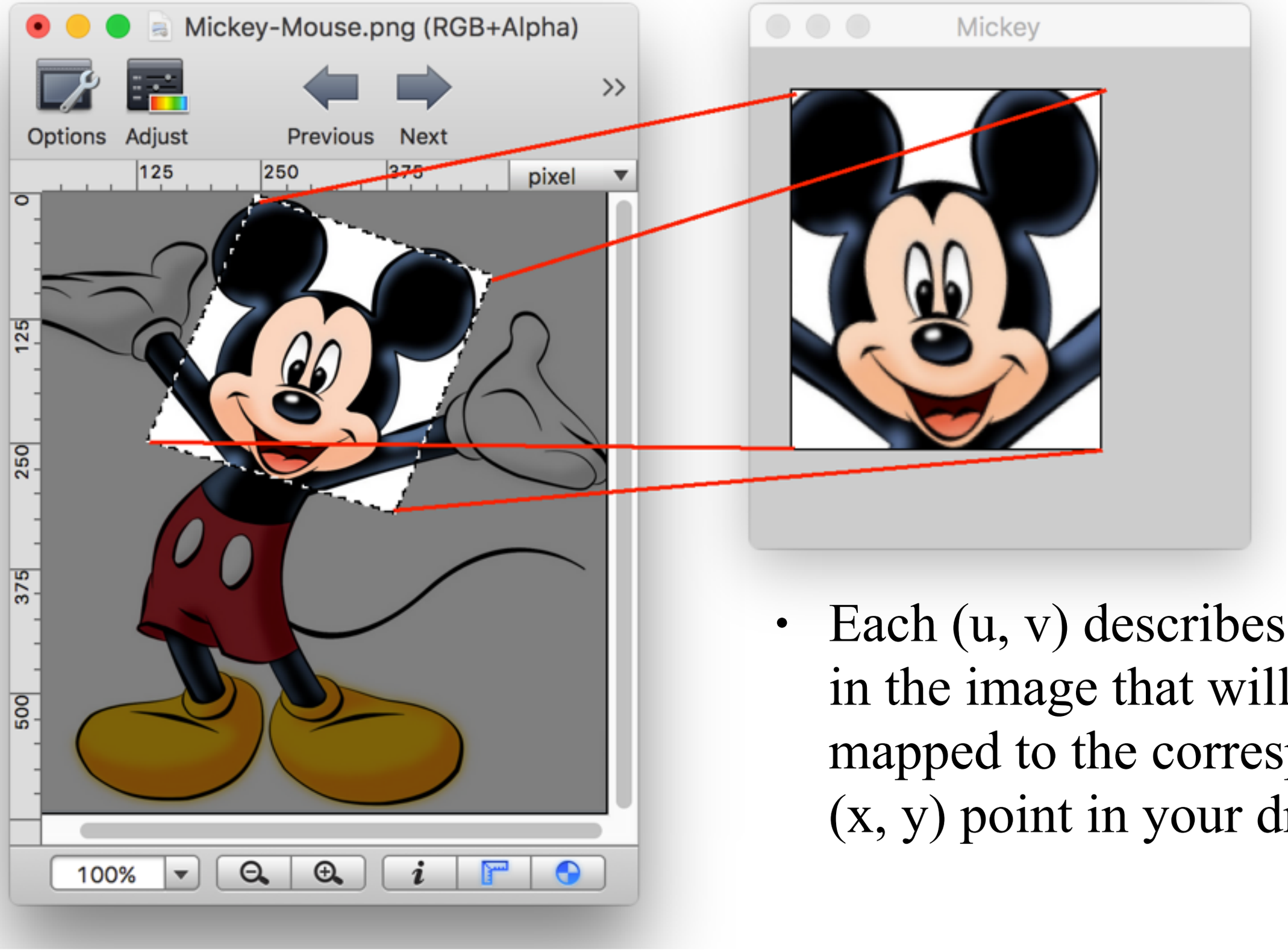
Textures I

- You can add textures to a shape:
  - Add `P2D` to the call to `size`:

    ```
    size(400, 400, P2D);
    ```

  - Use `loadImage(`*path*`)` to get the image
  - As the first thing within a `beginShape...endShape`, put a call to `texture(`*image*`)`
- Add two arguments, `u` and `v`, to each call to `vector` inside the shape, for example, `vertex(20, 20, 245, 0);`
  - Each `u` and `v` coordinate pair in the image are mapped to the corresponding `x` and `y` coordinates of your drawing
  - To use the entire image, use *image*`.width` and *image*`.height`
  - The image will be stretched and sheared to fit the area you are putting it in

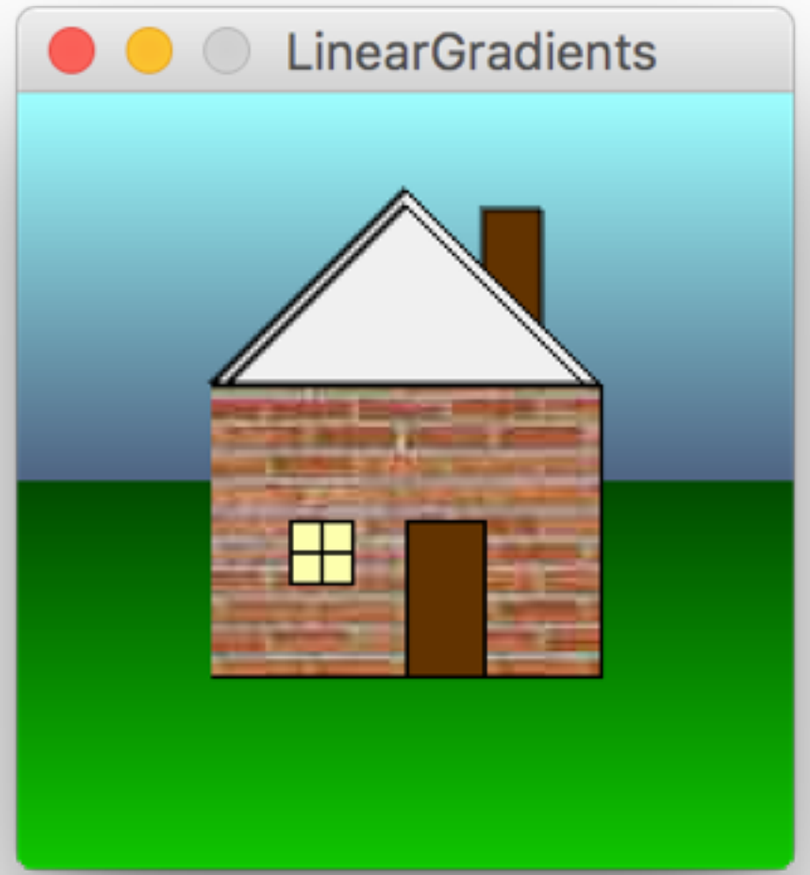
Textures II



- Each (u, v) describes a point in the image that will be mapped to the corresponding (x, y) point in your drawing

House with texture

```
void house() {
  rect(50, 75, 100, 75); // (left, top, w, h)
  PImage img = loadImage("/Users/dave/bricks.jpeg");
  beginShape();
    texture(img);
    vertex(50, 75, 50, 75);
    vertex(150, 75, 150, 75);
    vertex(150, 150, 150, 150);
    vertex(50, 150, 50, 150);
  endShape();
  fill(100, 50, 0);
  rect(100, 110, 20, 40);   // door
  fill(255, 255, 180);
  rect(70, 110, 16, 16);    //window
  fill(240, 240, 240);
  line(70, 118, 86, 118);
  line(78, 110, 78, 126);
  triangle(50, 75, 100, 25, 150, 75); // roof
  triangle(54, 75, 100, 29, 146, 75);
  fill(100, 50, 0);
  quad(120, 45, 120, 30, 135, 30, 135, 60); // chimney
}
```



The End