”I try to apply colors like words that shape poems, like notes that shape music.”

Table of Contents

Everything in this book has been generated by code.

Every instance of this book is unique.

Chapter 00: Introduction

Color, "the property possessed by an object of producing different sensations on the eye as a result of the way the object reflects or emits light"--according to the Dictionary app on my laptop. Here's another: "one, or any mixture, of the constituents into which light can be separated in a spectrum or rainbow, sometimes including (loosely) black and white". Whatever that means.

It is believed that color 'ought' to be separate from language, and that the more we try to define or describe color the further we are from truly experiencing it. When I started writing this book it was never my intention to emerge with a definition of what color is--rather, I was interested in understanding color and seeing whether the things we see with our eyes can be reproduced systematically. Methodically. Algorithmically. But the more I tried to develop these algorithms, the more I realized that describing color was exactly what I was trying to do.

I am at a stage in my life where I am still trying to understand how things in the world work, still trying to figure out who I am, still suffering from the occasional existential crisis. Since I am probably unqualified to articulate something on behalf of other people, this book is not about how color works for everyone. This is about how color works for me.

This is The Colorist's Cookbook.

Chapter 01: 3 Colors Into 4 Part 1

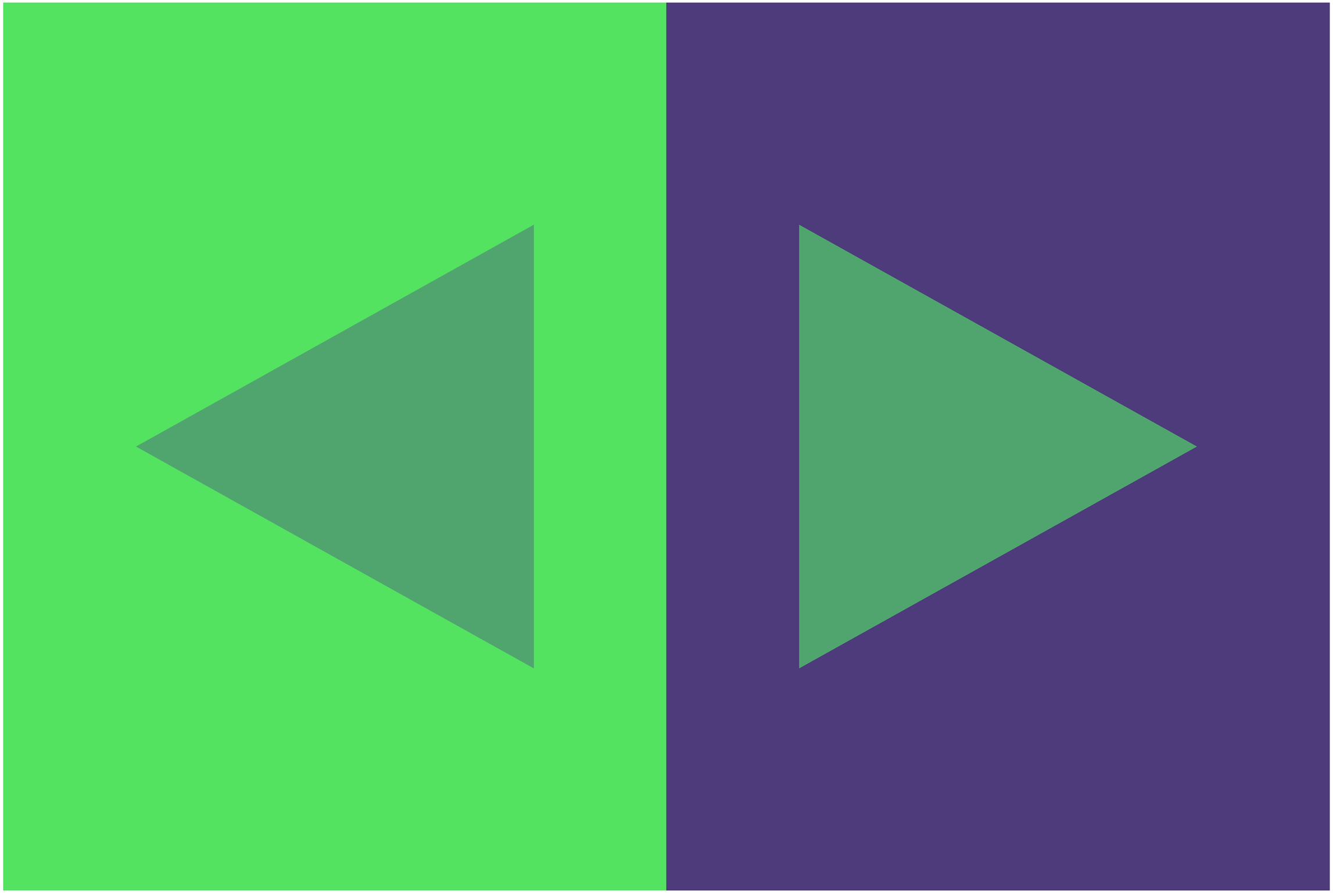
Remember the dress that turned into a viral phenomenon? Not that thing Lady Gaga wore to the VMA's that one year--but the one that got famous just because people couldn't decide whether it was black and blue or white and gold? In February 2015, a woman took a picture of a dress she wanted to wear to a wedding, and released the photo on Tumblr after many of her friends disagreed over the color. The picture spread quickly across the Internet, and sparked an intense public debate about the color of the dress. It turns out that this phenomenon can be explained by the pure science of color vision. When scientists pitched in to provide insight into what was going on they found that if the dress was shown in yellow lighting the majority of people would see it as blue and black, while lighting with a blue bias caused people to view the dress as white and gold.*

According to color theory, having a color in the background removes that color from the foreground--this is why the dress shown against yellow lighting would make it look less yellow and more blue. I won't get into the specifics of the science behind this because I'm running out of space on this page and frankly I don't fully understand it myself--but this is a very important and interesting concept in color theory that is the basis of many optical illusions. (It also really f**ked me up and made me question whether I was worthy of having perfect color vision.) This chapter is about using this very concept to make 3 colors look like 4. I hope it causes you to lose as much faith in your eyeballs as I did.

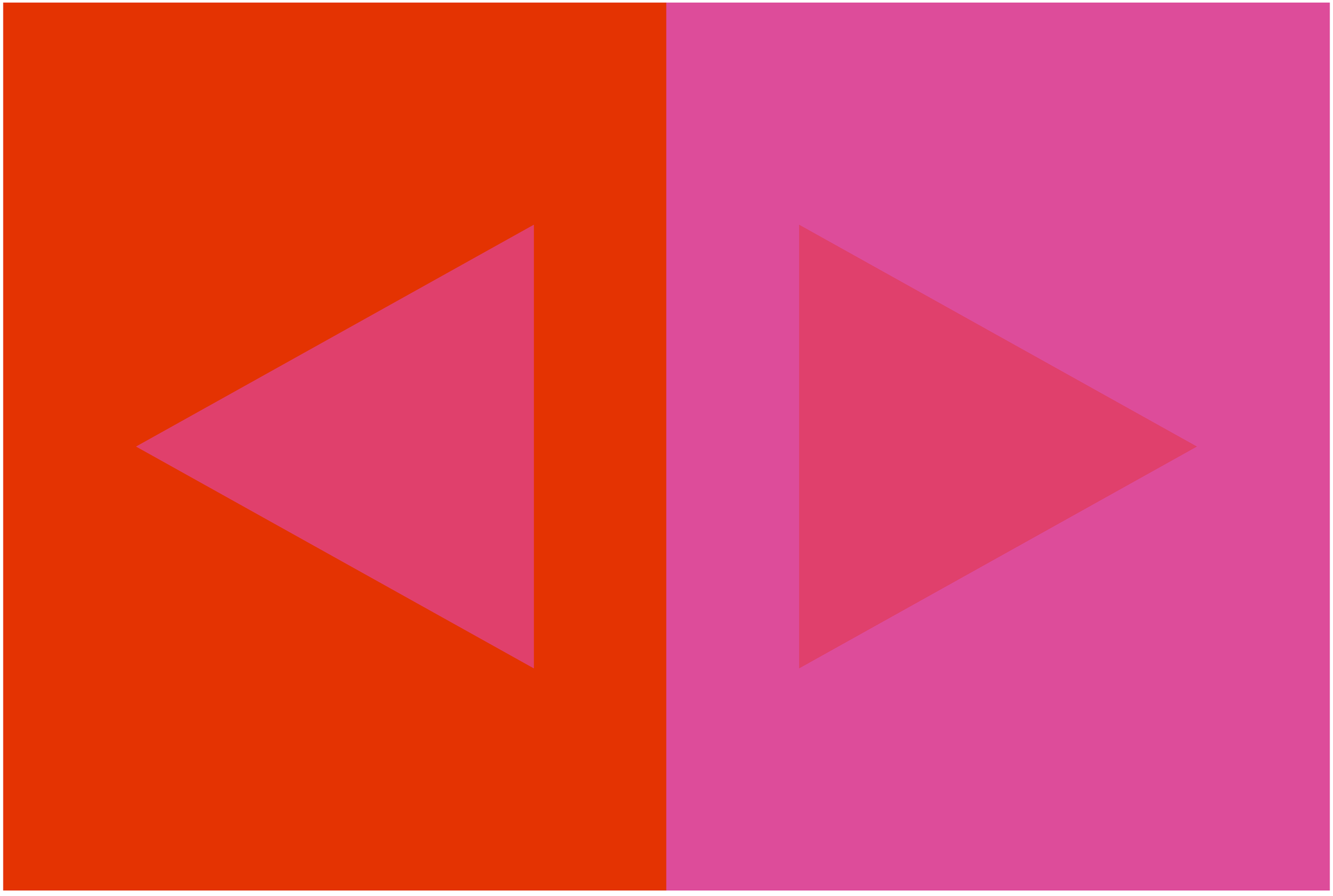
*Look up #thedress or Dressgate if my summary displeased you and you want to find out more

0xFFEBE054 0xFFC5F0F5 0xFFD8E8B7

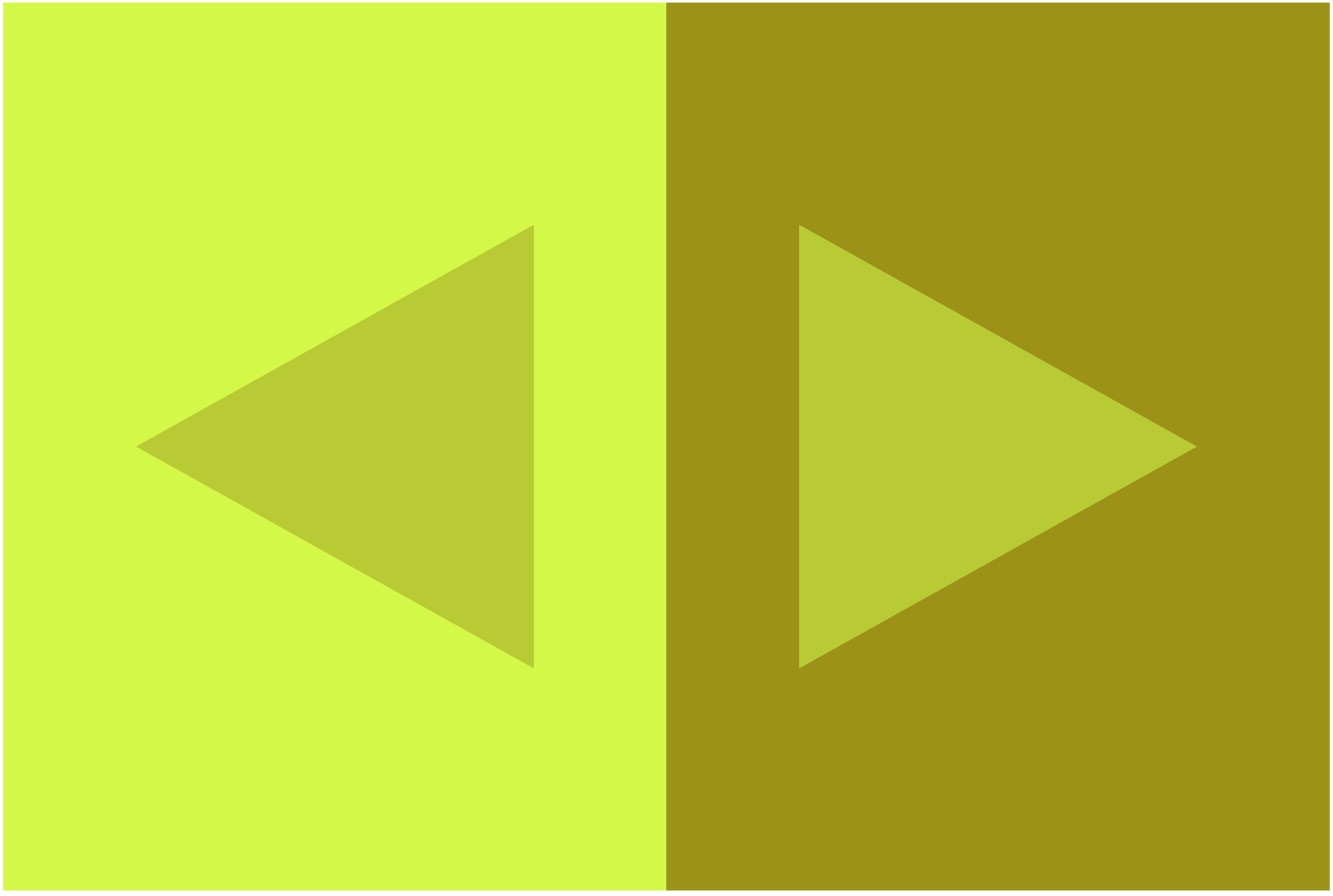
0xFF54E261 0xFF4D3B7B 0xFF50A56E



0xFFE43302 0xFFDD4C9A 0xFFE0406C



0xFFD4F847 0xFF9B9217 0xFFB9CB34



```
// recipe for making 3 colors look like 4

// prepare the first color
SecureRandom random = new SecureRandom();

int min = 0;
int max = 255;
int r1 = random.nextInt(max-min+1)+min;
int g1 = random.nextInt(max-min+1)+min;
int b1 = random.nextInt(max-min+1)+min;
color col1 = color(r1, g1, b1);

// then prepare the second color
int r2 = random.nextInt(max-min+1)+min;
int g2 = random.nextInt(max-min+1)+min;
int b2 = random.nextInt(max-min+1)+min;
color col2 = color(r2, g2, b2);
// evenly mix the first two colors to create
// the 'middle' color
float mixedred = sqrt((sq(red(col1))*0.5 +sq(red(col2))*0.5));
float mixedgreen = sqrt((sq(green(col1))*0.5 +sq(green(col2))*0.5));
float mixedblue = sqrt((sq(blue(col1))*0.5 +sq(blue(col2))*0.5));

color mid = color(mixedred, mixedgreen, mixedblue);
// pre-translate the transformation matrix
// to the size of your margins
pushMatrix();
translate(margin, margin);
```

```
// arrange the first two colors beside each other
fill(col1);
stroke(col1);
rect(0,0,pgwidth/2f,pgheight);
fill(col2);
stroke(col2);
rect(pgwidth/2f,0,pgwidth/2f,pgheight);

// top with the middle color
fill(mid);
noStroke();

triangle(pgwidth*0.1,pgheight/2f,pgwidth*0.4,pgheight/4f, pgwidth*0.4,pgheight*0.75);
triangle(pgwidth*0.9,pgheight/2f,pgwidth*0.6,pgheight/4f, pgwidth*0.6,pgheight*0.75);
  popMatrix();
```

Chapter 02: 3 Colors Into 4 Part II

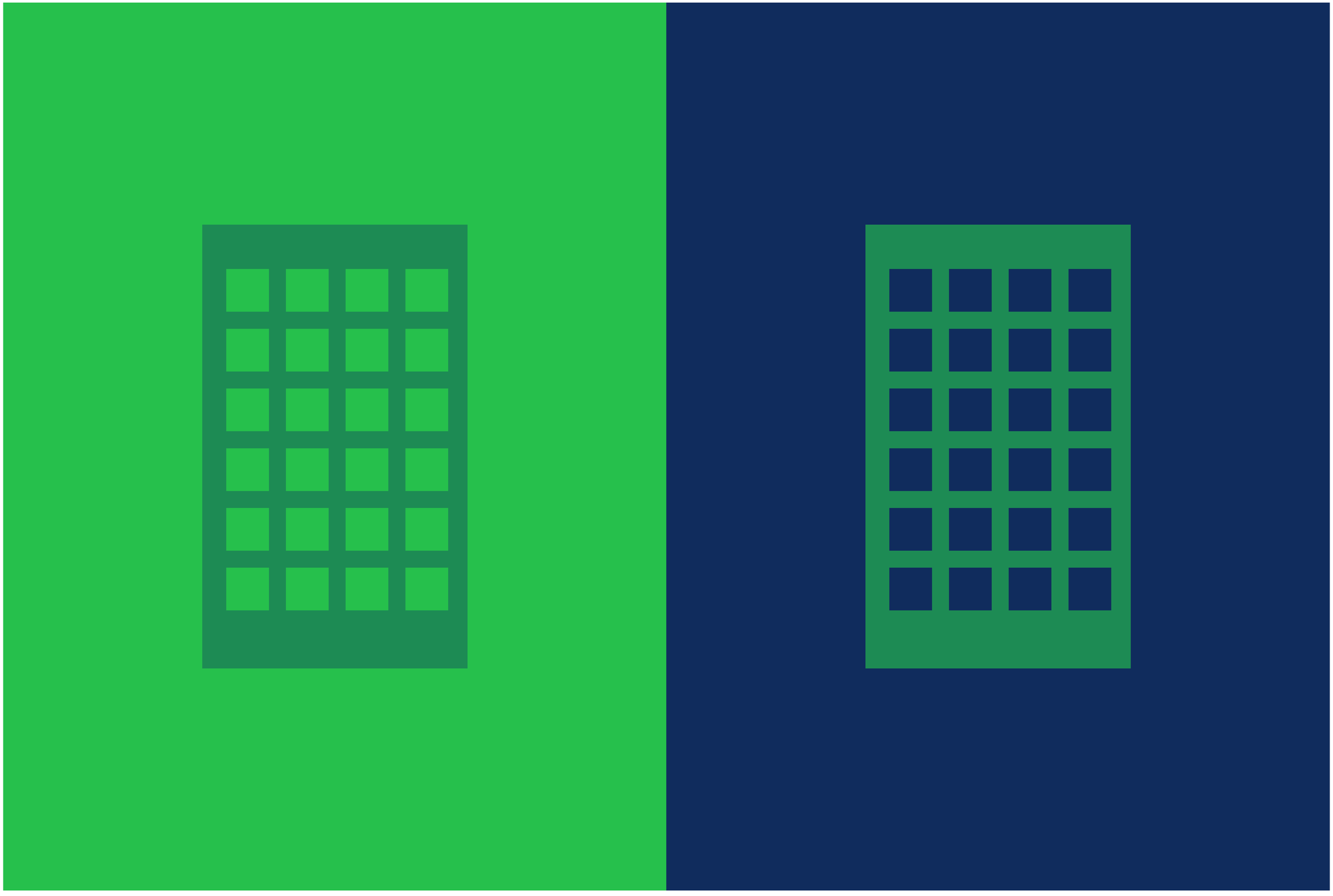
A guy named Pablo Picasso once said, "Why do two colors, put one next to the other, sing? Can one really explain this? No. Just as one can never learn how to paint." I'm sorry Pab, but I'm going to have to disagree with you on this one because:

1) I started learning how to paint the moment I taught myself how to hold a brush. I'm still learning.

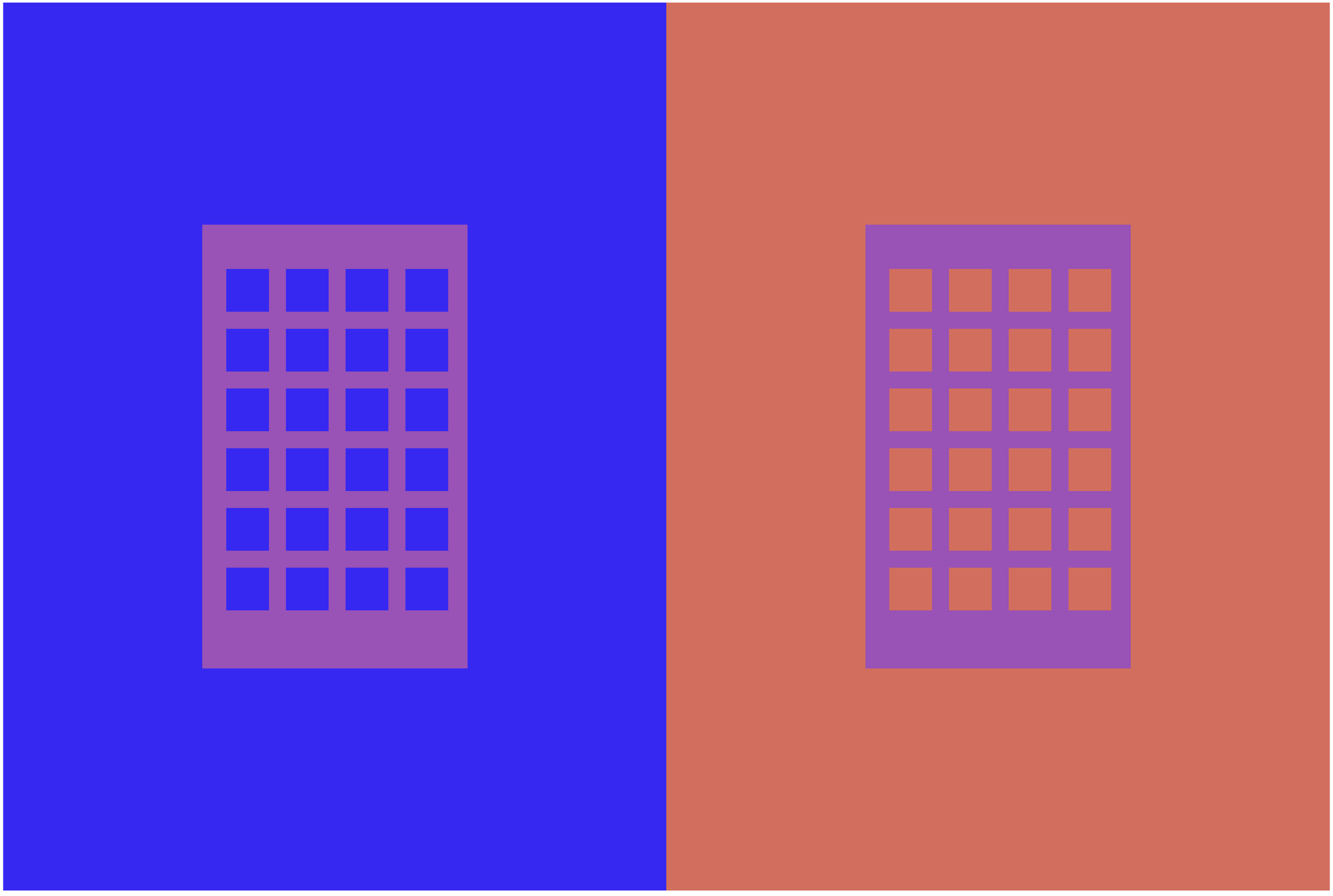
2) Although it's true that color harmony is complex and often tricky to articulate, it is not entirely inexplicable. Color theorists are better equipped to answer this than I am, but one reason why two colors may 'sing' when placed next to each other is because aesthetic responses can be evoked by complementary hues that are juxtaposed against each other, or analogous hues that are meshed with each other.

The logician in me believes that we can always find a reasonable explanation for almost all observable phenomena. The philosopher in me believes that, without the aid of some kind of 'oracle', there is no way for us to really confirm that our explanations are reasonable. I guess this is why I decided to write this book--because I wanted to see if I could prove my selves wrong (and therefore right) by turning the logician into a philosopher, and the philosopher into a logician.

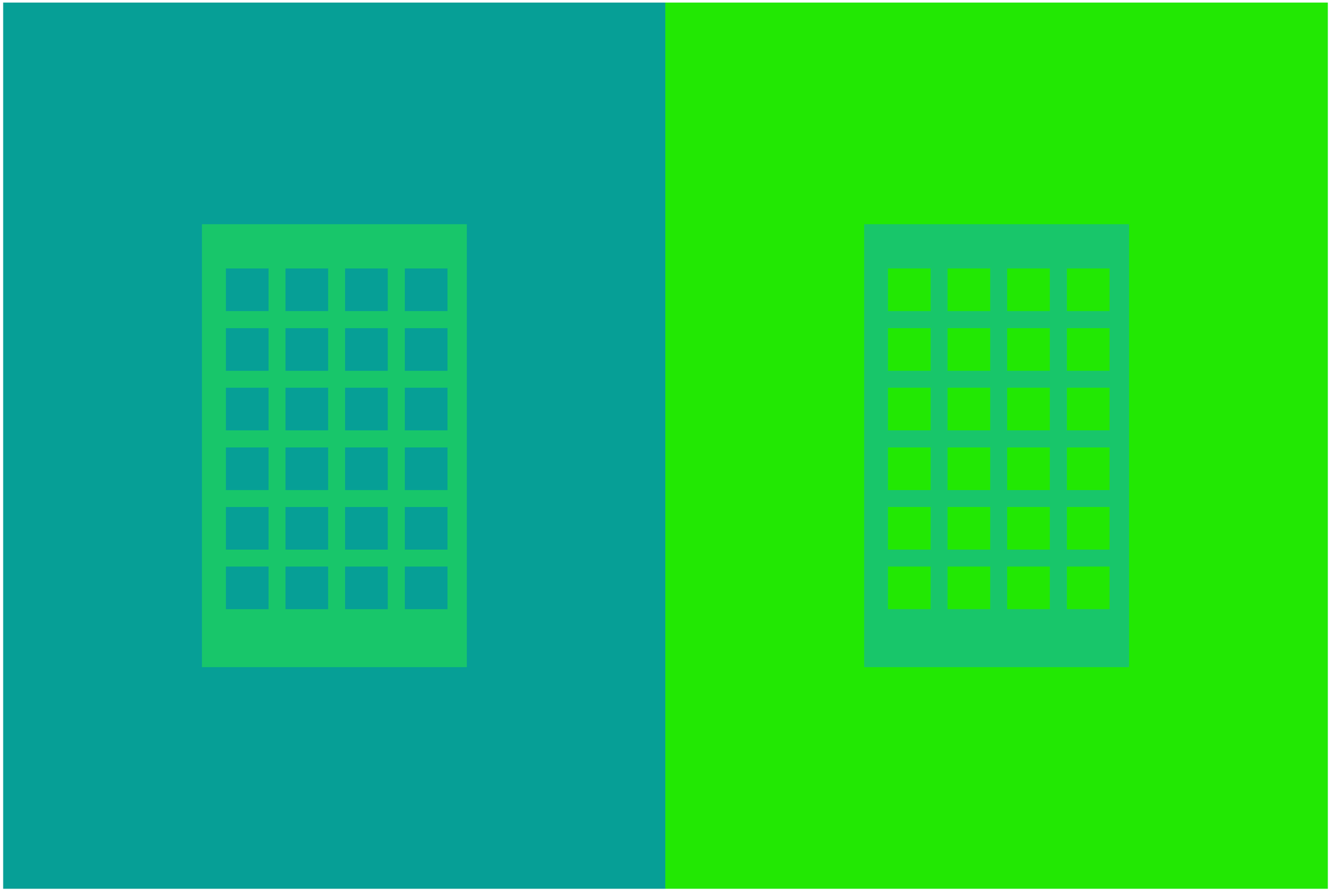
0xFF26C04C 0xFF102C5D 0xFF1D8B54



0xFF3628F1 0xFFD26E5E 0xFF9952B6



0xFF069F96 0xFF22E803 0xFF18C66A



```
// recipe for making 3 colors look like 4...with holes!

// prepare the first color
SecureRandom random = new SecureRandom();

int min = 0;
int max = 255;
int r1 = random.nextInt(max-min+1)+min;
int g1 = random.nextInt(max-min+1)+min;
int b1 = random.nextInt(max-min+1)+min;
color col1 = color(r1, g1, b1);

// then prepare the second color
int r2 = random.nextInt(max-min+1)+min;
int g2 = random.nextInt(max-min+1)+min;
int b2 = random.nextInt(max-min+1)+min;
color col2 = color(r2, g2, b2);
// evenly mix the first two colors to create
// the 'middle' color
float mixedred = sqrt((sq(red(col1))*0.5 +sq(red(col2))*0.5));
float mixedgreen = sqrt((sq(green(col1))*0.5 +sq(green(col2))*0.5));
float mixedblue = sqrt((sq(blue(col1))*0.5 +sq(blue(col2))*0.5));

color mid = color(mixedred, mixedgreen, mixedblue);

// pre-translate the transformation matrix
// to the size of your margins
pushMatrix();
translate(margin, margin);
```

```
rectMode(CORNER);

// arrange first two colors beside each other
fill(col1);
stroke(col1);
rect(0,0,pgwidth/2f,pgheight);
fill(col2);
stroke(col2);
rect(pgwidth/2f,0,pgwidth/2f,pgheight);

// top with the middle color
rectMode(CENTER);
fill(mid);
noStroke();
rect((pgwidth)/4f,pgheight/2f,pgwidth/5f,pgheight/2f);
rect((pgwidth*3f)/4f,pgheight/2f,pgwidth/5f,pgheight/2f);

// slice holes in the middle for a more dramatic effect
// waffle aesthetic
rectMode(CORNER);
fill(col1);
for(int rows = 0; rows < 6; rows++) {
  for(int cols = 0; cols < 4; cols++) {
    rect(pgwidth * 0.168 + 140*cols, pgheight * 0.3 + 140*rows,100,100);
  }
}

fill(col2);
float rand3 = random(-350,350);
```

```
pushMatrix();

for(int rows = 0; rows < 6; rows++) {
  for(int cols = 0; cols < 4; cols++) {
    rect(pgwidth * 0.668 + 140*cols, pgheight * 0.3 + 140*rows,100,100);
  }
}
popMatrix();

popMatrix();
```

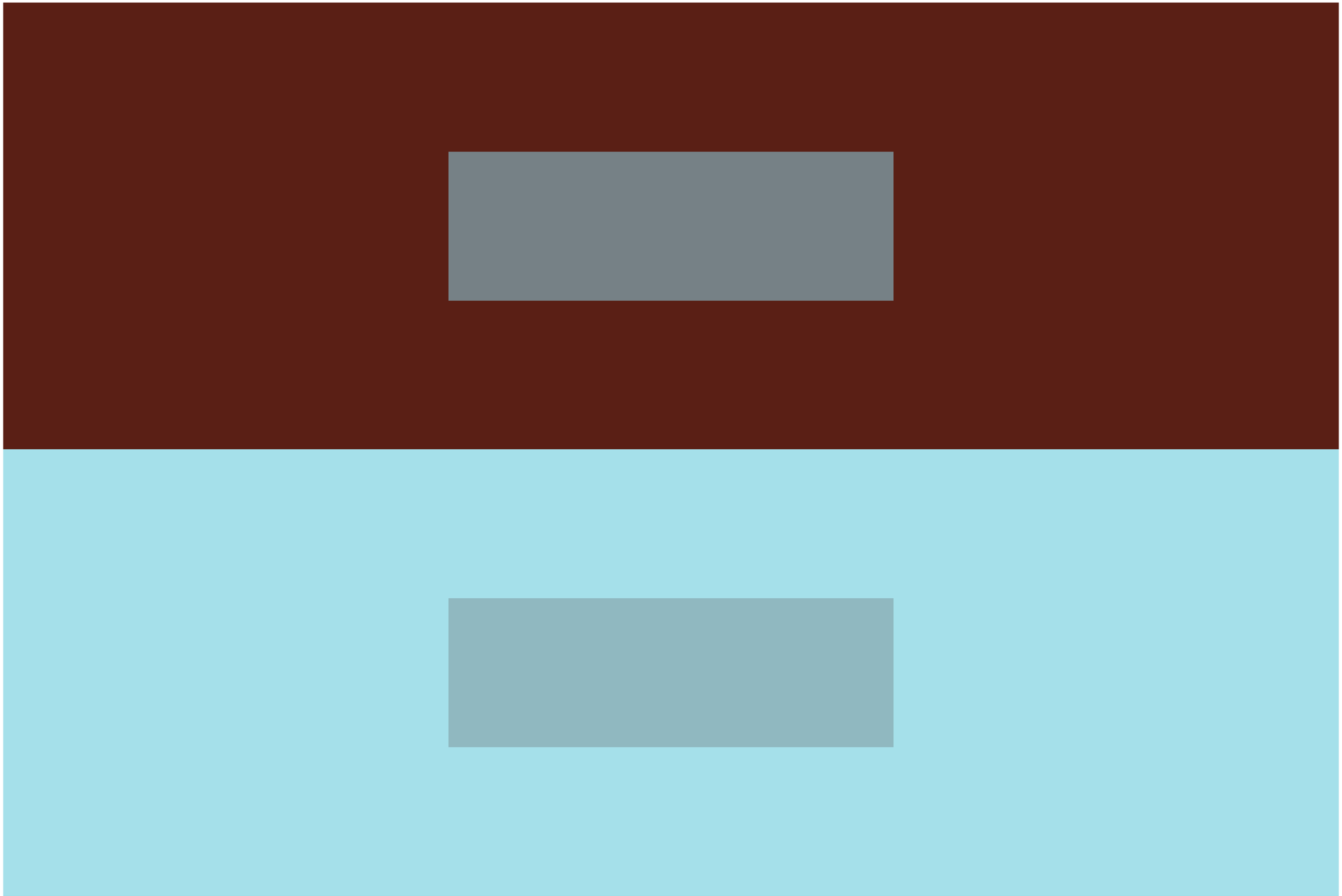
Chapter 03: 4 Colors Into 3

If you've been keeping up with the code (which you should--I painstakingly included it for a reason!), you could probably observe that everything boils down to just a bunch of math. This is because in computerland, colors (and all other things) are just numbers. Consider this: since every color is expressed as a number, then for an encoding scheme with 6 hexadecimal* numbers the value of colors ranges from 0 to 16,777,215--that gives us a total of 16,777,216 possible colors! Does that mean there are 16 million colors that exist in the world?

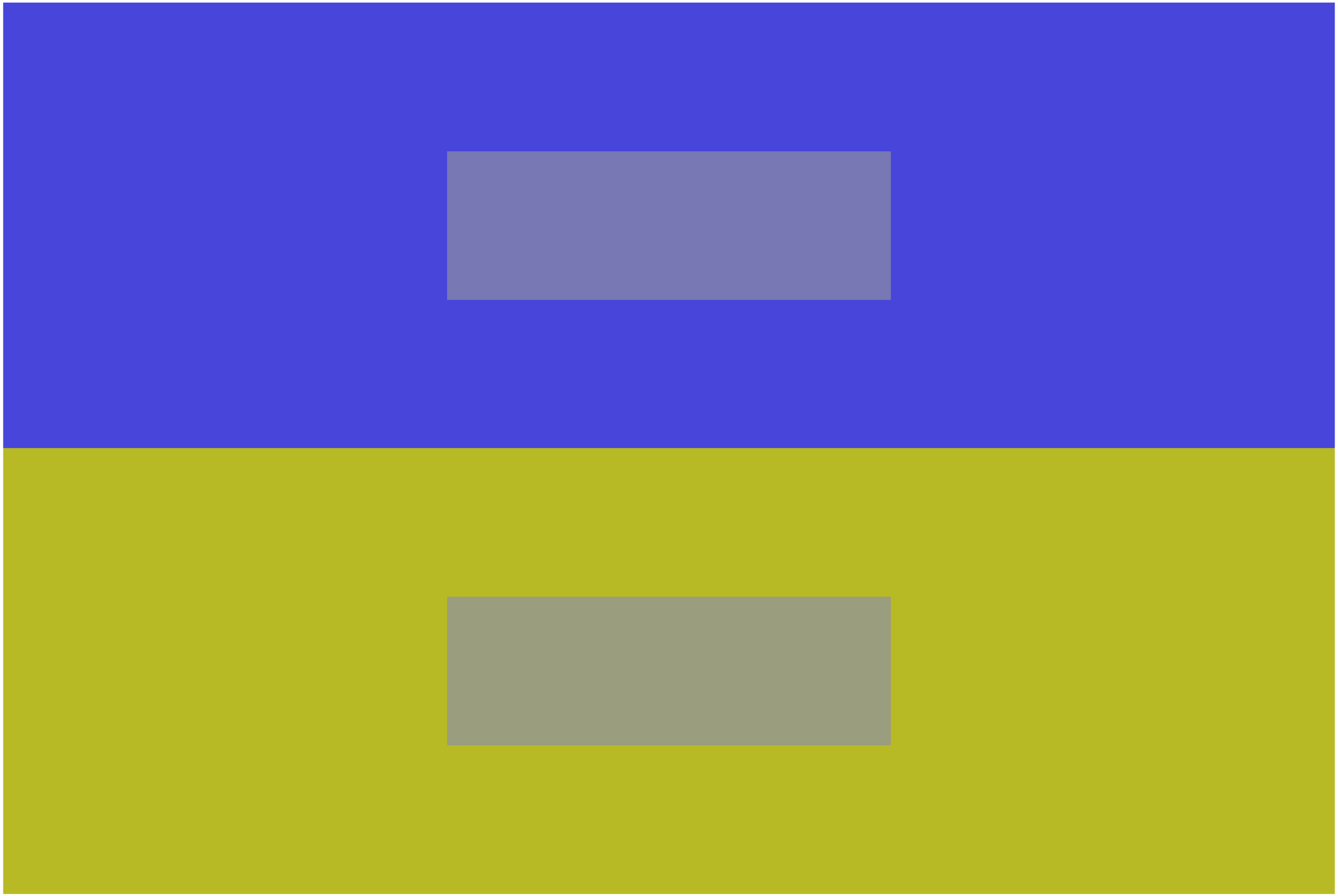
Well...no. It turns out that humans, on average, can only see about 10 million colors in a single viewing condition--so computers really encode 6 million more colors than necessary. So is the answer 10 million, then? Remember that the 10 million refers to the number of colors we can see in a SINGLE viewing condition. How many possible viewing conditions are there? And how do we take into account that color perception can differ from one person to another? That means the real answer is infinity. Now the question is: is the number of colors that exist in the world countably infinite, or uncountably infinite?** I suck at discrete math, so maybe someone else can answer this question for me.

*this is just a way of saying base-16, which is a very common numerical system for computers. Normal people think in base-10.
**for those who are not familiar with these terms: in mathland there are two infinities, the uncountable and countable. If that isn't weird enough, it has also been proven that the uncountable infinity is 'bigger' than the countable infinity.

0xFF5A1F15 0xFFA5E0EA 0xFF768186 0xFF90B8C0



0xFF4845DB 0xFFB7BA24 0xFF7778B4 0xFF9B9D7F



0xFF60A16F 0xFF9F5E90 0xFF778E7A 0xFF8D7785

```
// recipe for making 4 colors look like 3
// prepare the first and second colors
SecureRandom random = new SecureRandom();

int min = 0;
int max = 255;
int r1 = random.nextInt(max-min+1)+min;
int g1 = random.nextInt(max-min+1)+min;
int b1 = random.nextInt(max-min+1)+min;
color col1 = color(r1, g1, b1);

// the second color is opposite from the first color
color col2 = color(255 - red(col1),
                   255 - green(col1),
                   255 - blue(col1));

// evenly mix the first two colors to create
// the 'middle' color
float mixedred = sqrt((sq(red(col1))*0.5 +sq(red(col2))*0.5));
float mixedgreen = sqrt((sq(green(col1))*0.5 +sq(green(col2))*0.5));
float mixedblue = sqrt((sq(blue(col1))*0.5 +sq(blue(col2))*0.5));

color mid = color(mixedred, mixedgreen, mixedblue);

float c1_weight = 0.35;
float c2_weight = 0.65;

// calculate a weighted average of the first color and middle color
mixedred = sqrt((sq(red(col1))*c1_weight +sq(red(mid))*c2_weight));
```

```
mixedblue = sqrt((sq(blue(col1))*c1_weight +sq(blue(mid))*c2_weight));

color mixed1 = color(mixedred, mixedgreen, mixedblue);

// calculate a weighted average of the second color and middle color
mixedred = sqrt((sq(red(col2))*c1_weight +sq(red(mid))*c2_weight));
mixedgreen = sqrt((sq(green(col2))*c1_weight +sq(green(mid))*c2_weight));
mixedblue = sqrt((sq(blue(col2))*c1_weight +sq(blue(mid))*c2_weight));

color mixed2 = color(mixedred, mixedgreen, mixedblue);

// pre-translate the transformation matrix
// to the size of your margins
pushMatrix();
translate(margin, margin);
rectMode(CORNER);

// arrange opposing colors beside each other
fill(col1);
stroke(col1);
rect(0,0,pgwidth,pgheight/2);
fill(col2);
stroke(col2);
rect(0,pgheight/2,pgwidth,pgheight/2);

// top with the middle colors
fill(mixed1);
noStroke();
```

```
fill(mixed2);
rect((pgwidth)/3,  (pgheight*2)/3, pgwidth/3, pgheight/6);


popMatrix();
```

Chapter 04: 5 Colors Into 3

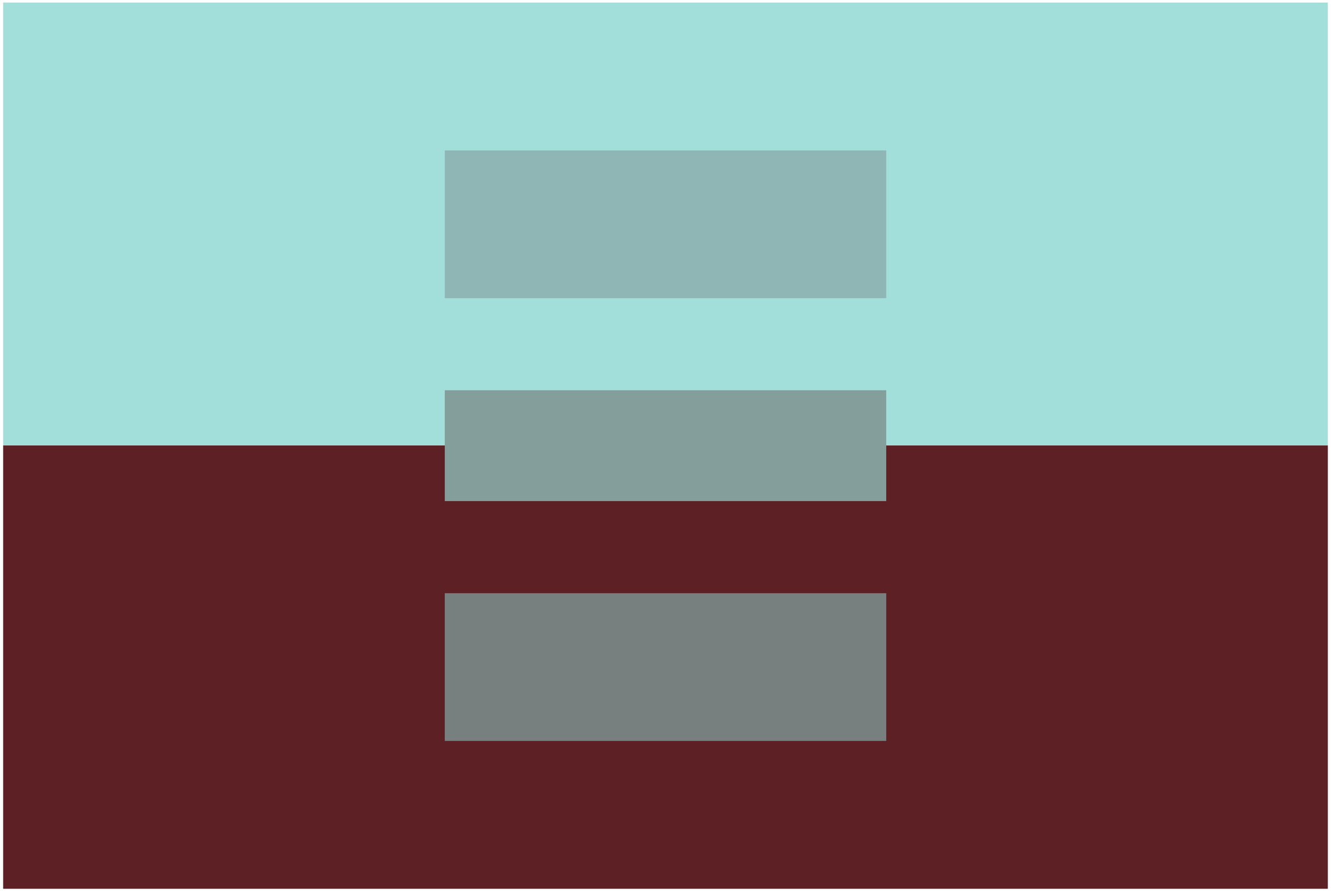
So I probably should be saying something about the next chapter, but I want to tell you a story instead. Last Tuesday, I had to go to my professor's office hours to prepare for an exam. My friend and I made our way to the back corner of the room and sat down, taking out our notes and note-taking instruments and whatever else people possess when they need to study. As more students shuffled into the small office with questions about the material, the afternoon proceeded unremarkably. Some time passed, and as the questions began to slow down I started to have a fairly unremarkable conversation with my friend about Windows computers:

"Oh, you're a Windows person?" I asked him.
"Yeah," he replied.
"Okay, I'm not judging," I lied.
And then:
"I am," chimed in my professor. "Macs are better."

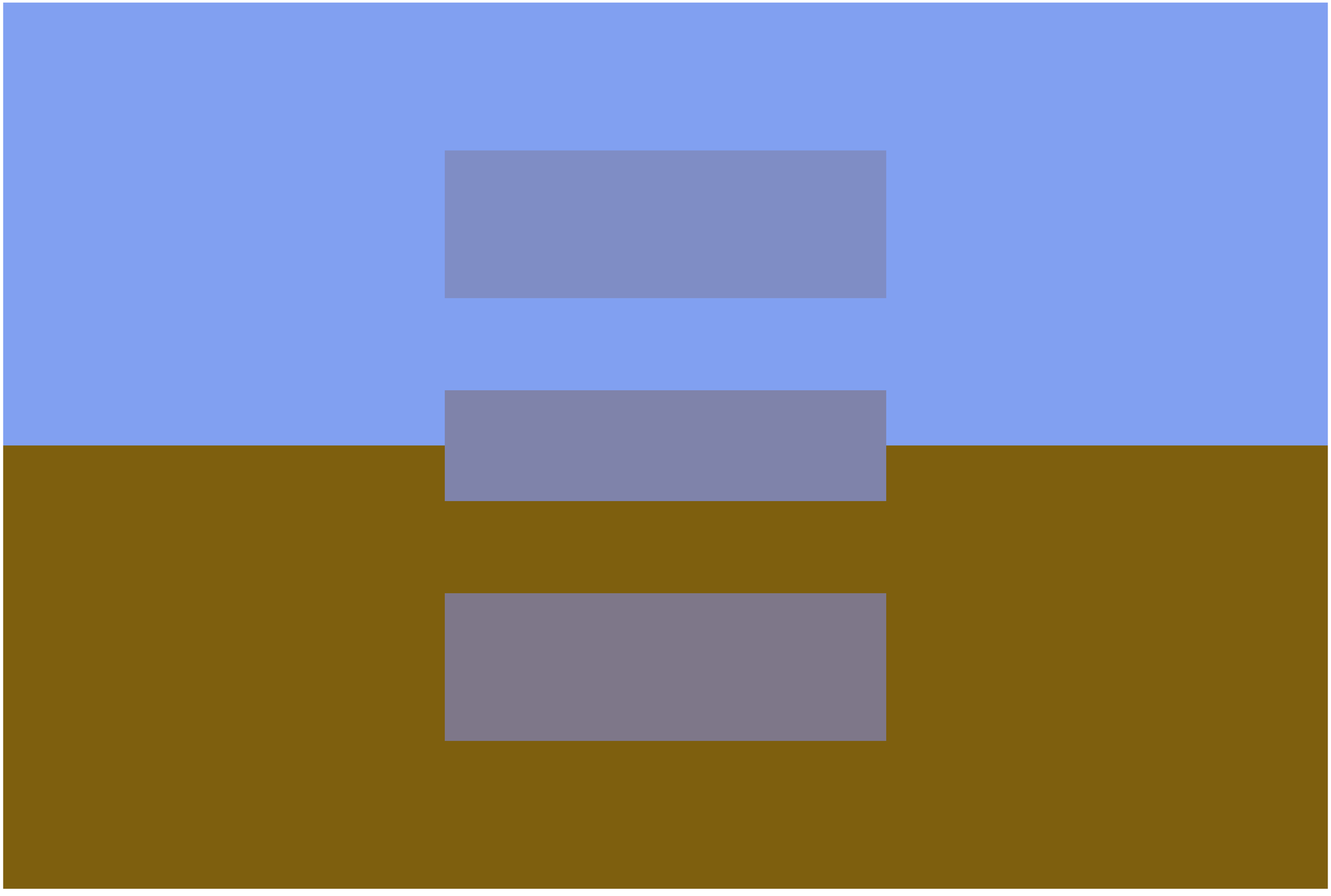
And that's when the room was thrown into a passive-aggressive debate about whether Macs were better than Windows machines (or rather, why Windows machines were better than Macs).

Now I have a confession to make: for the majority of public situations in which I am surrounded by people who are some combination of scientists/engineers/mathematicians, I generally avoid declaring myself as an art student--not out of shame but as a precautionary measure to avoid hostility and having metaphorical daggers thrown at my back. But for whatever reason, on that unremarkable Tuesday, during that unremarkable conversation, it was exactly what I did.

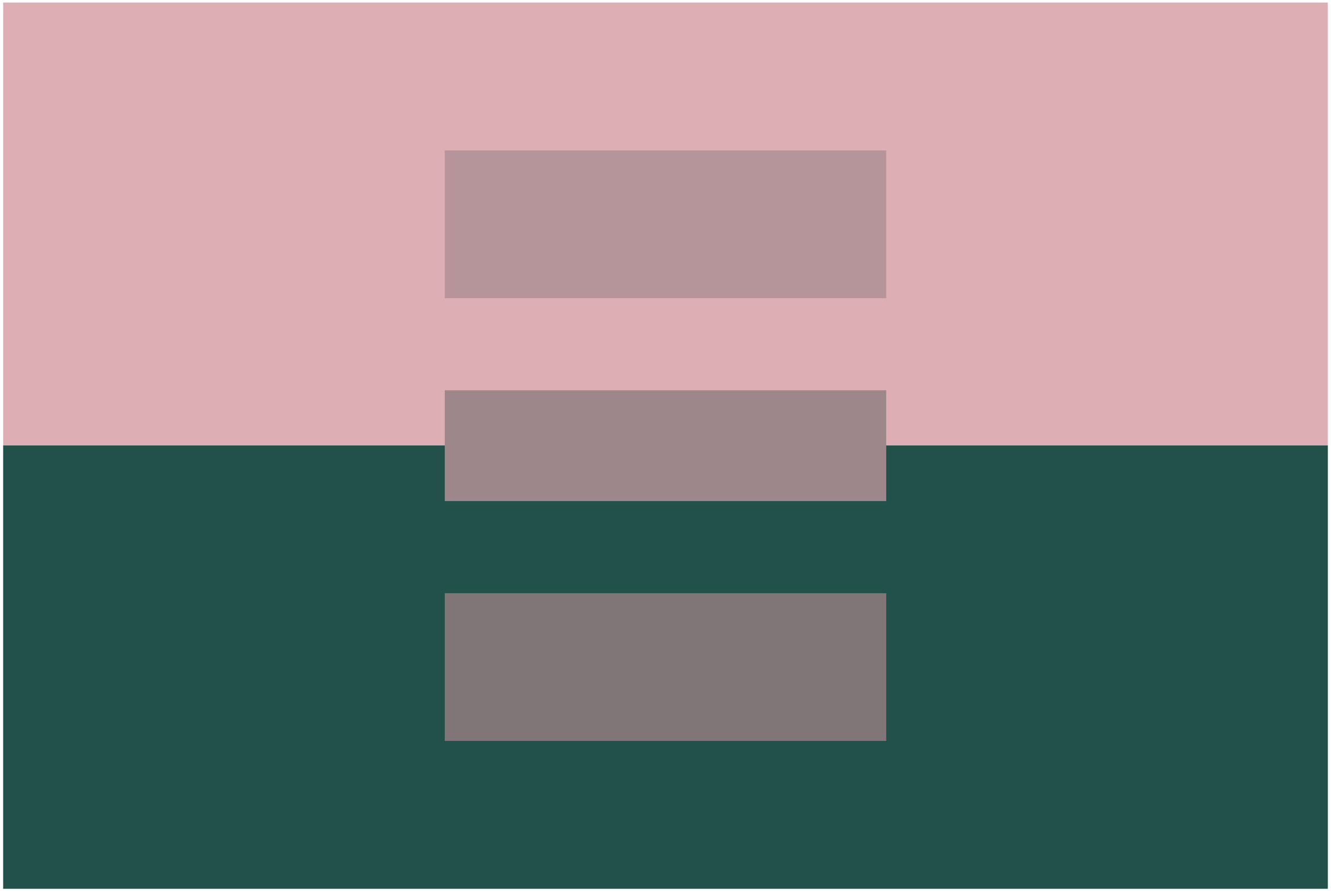
0xFFA2DEDA 0xFF5D2125 0xFF8FB6B4 0xFF849E9C 0xFF77807F



0xFF81A0F1 0xFF7E5F0E 0xFF7F8DC5 0xFF7F83AA 0xFF7E7789



0xFFDEAEB5 0xFF21514A 0xFFB6959A 0xFF9E878A 0xFF807677



```
// recipe for making 5 colors look like 3
// prepare the first and second colors
SecureRandom random = new SecureRandom();

int min = 0;
int max = 255;
int r1 = random.nextInt(max-min+1)+min;
int g1 = random.nextInt(max-min+1)+min;
int b1 = random.nextInt(max-min+1)+min;
color col1 = color(r1, g1, b1);

// the second color is opposite from the first color
color col2 = color(255 - red(col1),
                   255 - green(col1),
                   255 - blue(col1));

// evenly mix the first two colors to create
// the 'middle' color
float mixedred = sqrt((sq(red(col1))*0.5 +sq(red(col2))*0.5));
float mixedgreen = sqrt((sq(green(col1))*0.5 +sq(green(col2))*0.5));
float mixedblue = sqrt((sq(blue(col1))*0.5 +sq(blue(col2))*0.5));

color mid = color(mixedred, mixedgreen, mixedblue);

float c1_weight = 0.35;
float c2_weight = 0.65;

// calculate a weighted average of the first color and middle color
mixedred = sqrt((sq(red(col1))*c1_weight +sq(red(mid))*c2_weight));
```

```
mixedblue = sqrt((sq(blue(col1))*c1_weight +sq(blue(mid))*c2_weight));

color mixed1 = color(mixedred, mixedgreen, mixedblue);

// calculate a weighted average of the second color and middle color
mixedred = sqrt((sq(red(col2))*c1_weight +sq(red(mid))*c2_weight));
mixedgreen = sqrt((sq(green(col2))*c1_weight +sq(green(mid))*c2_weight));
mixedblue = sqrt((sq(blue(col2))*c1_weight +sq(blue(mid))*c2_weight));

color mixed2 = color(mixedred, mixedgreen, mixedblue);

// pre-translate the transformation matrix
// to the size of your margins
pushMatrix();
translate(margin, margin);
rectMode(CORNER);

// arrange opposing colors beside each other
fill(col1);
stroke(col1);
rect(0,0,pgwidth,pgheight/2);
fill(col2);
stroke(col2);
rect(0,pgheight/2,pgwidth,pgheight/2);

// top with the middle colors
fill(mixed1);
noStroke();
```

```
fill(mixed2);
rect((pgwidth)/3,  (pgheight*2)/3, pgwidth/3, pgheight/6);

fill(mid);
rect(pgwidth/3, pgheight*0.4375, pgwidth/3, pgheight/8);

popMatrix();
```

Chapter 05: Color Modulation

"I think as an art person," I started boldly. "I am naturally inclined to like Macs more," I could feel the daggers digging into my spine. "It's just--" as the daggers made their way deeper into my skin, I found myself helplessly fumbling for words--"I don't know, the colors are just...nicer?"

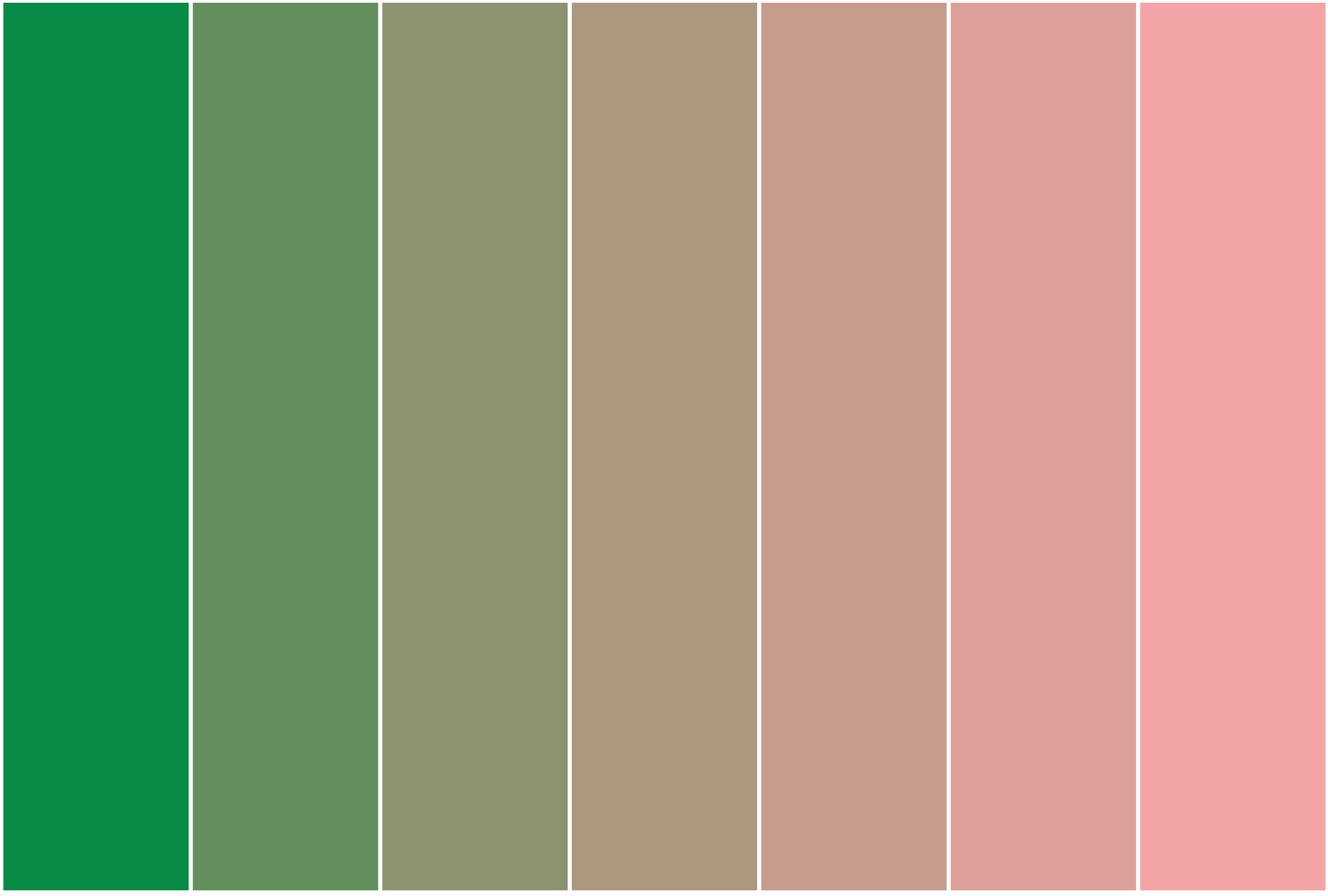
A heavy silence.

And then: "That's not a valid reason," argued the guy across the table.

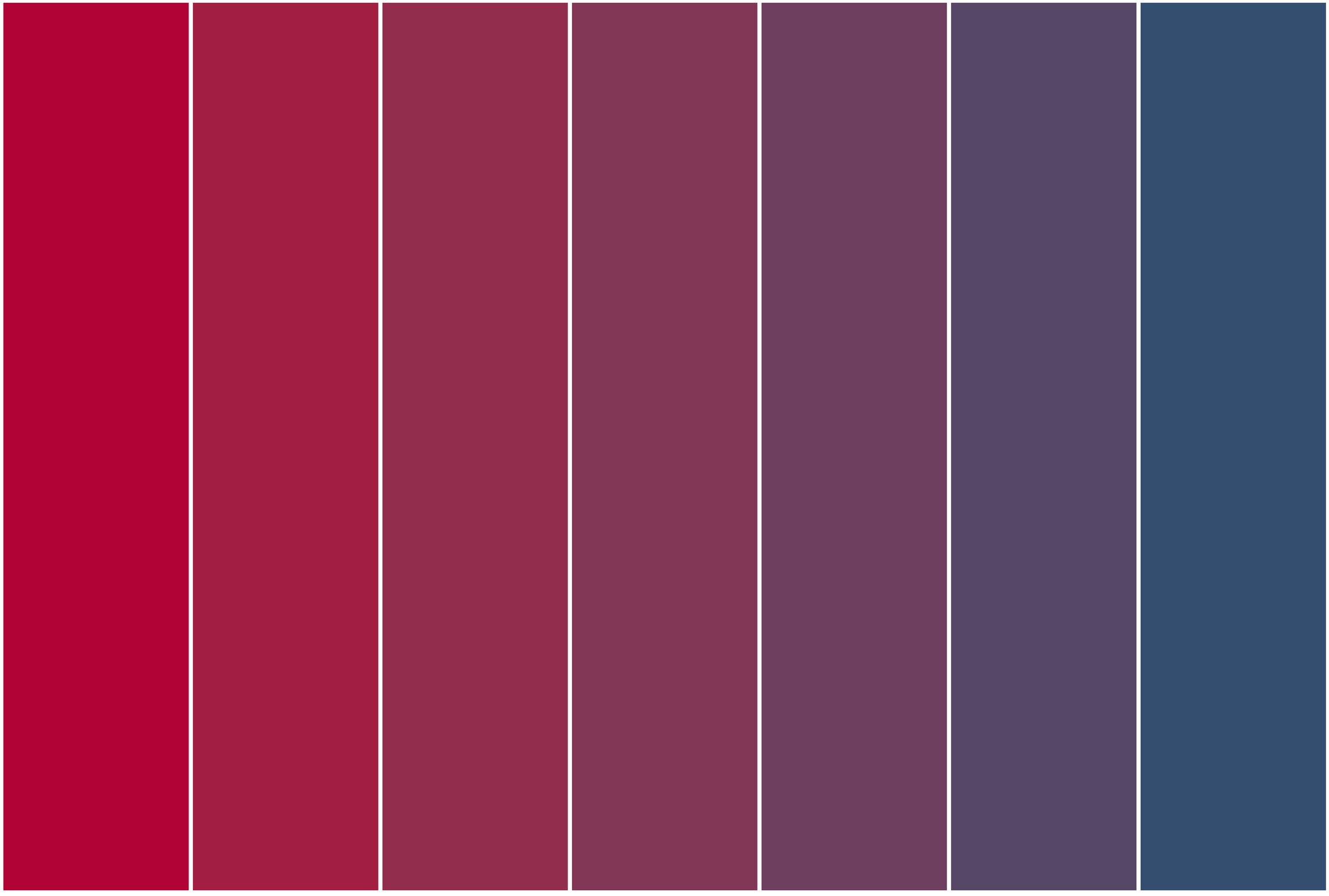
Is it?

When were colors ever 'a valid reason'?

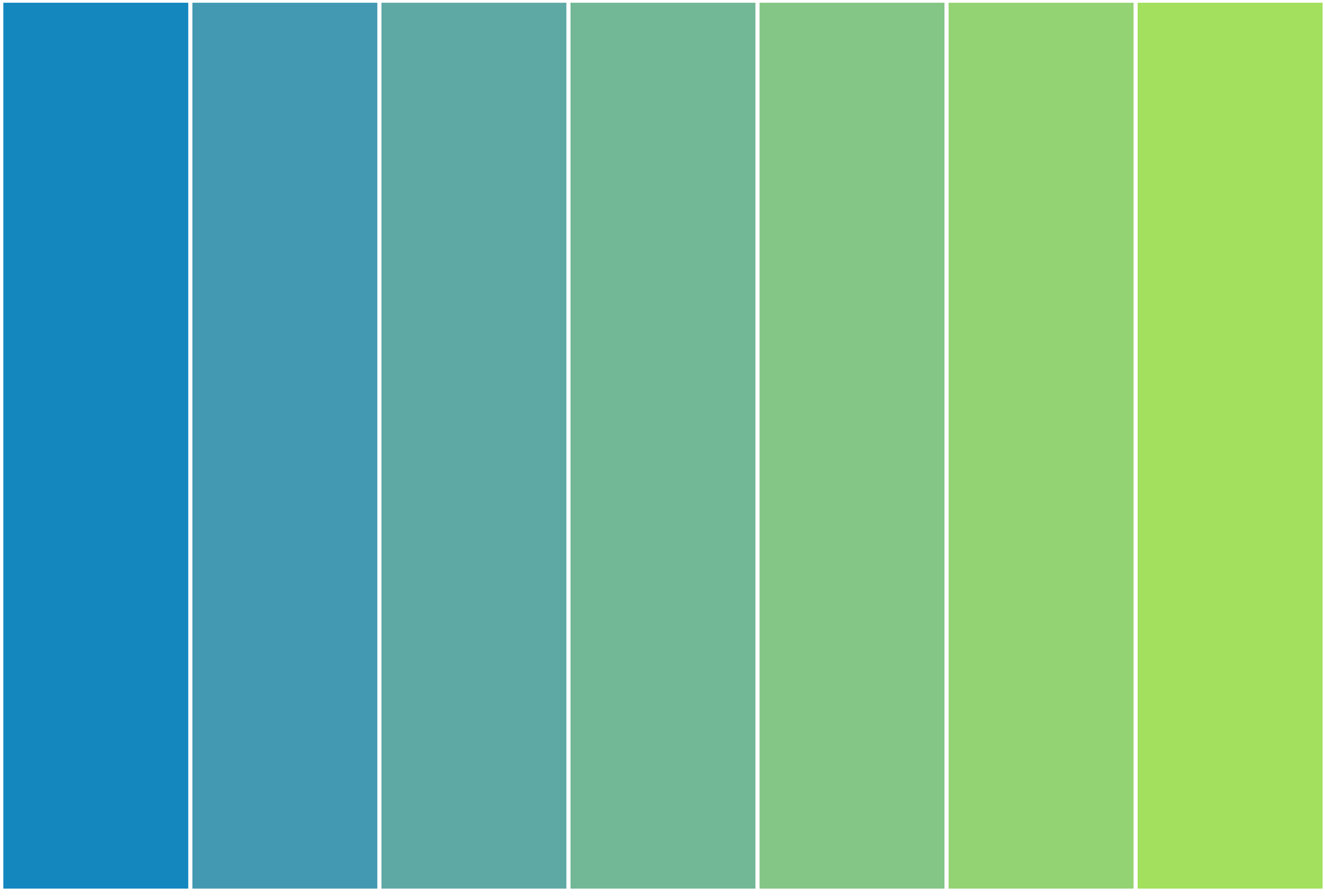
0xFF088B46 0xFF638F5D 0xFF8C936F 0xFFAB987F 0xFFC69C8D 0xFFDDA09A 0xFFF3A4A6



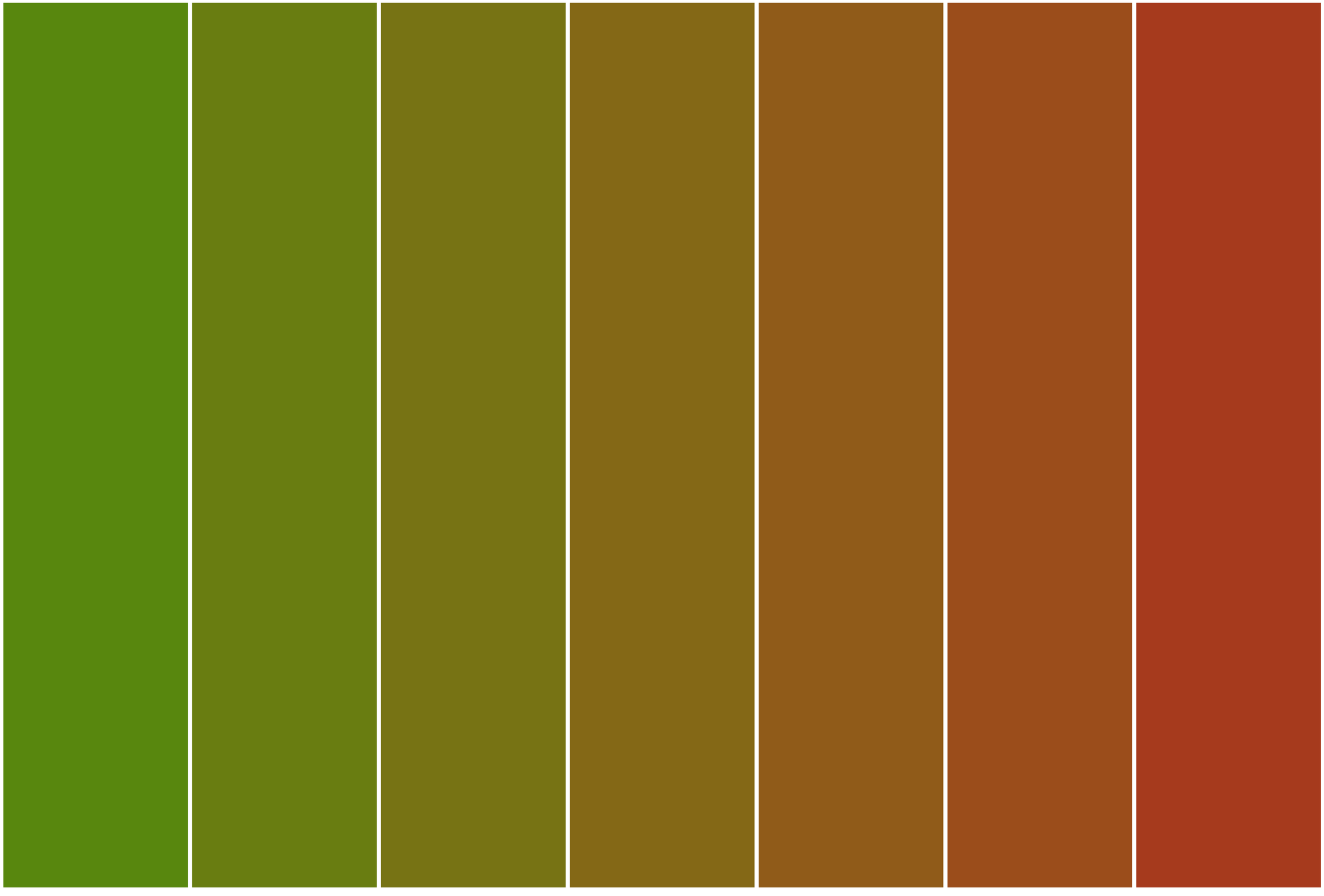
0xFFB10336 0xFFA21F42 0xFF932D4D 0xFF823757 0xFF6E3F5F 0xFF564767 0xFF334E6F



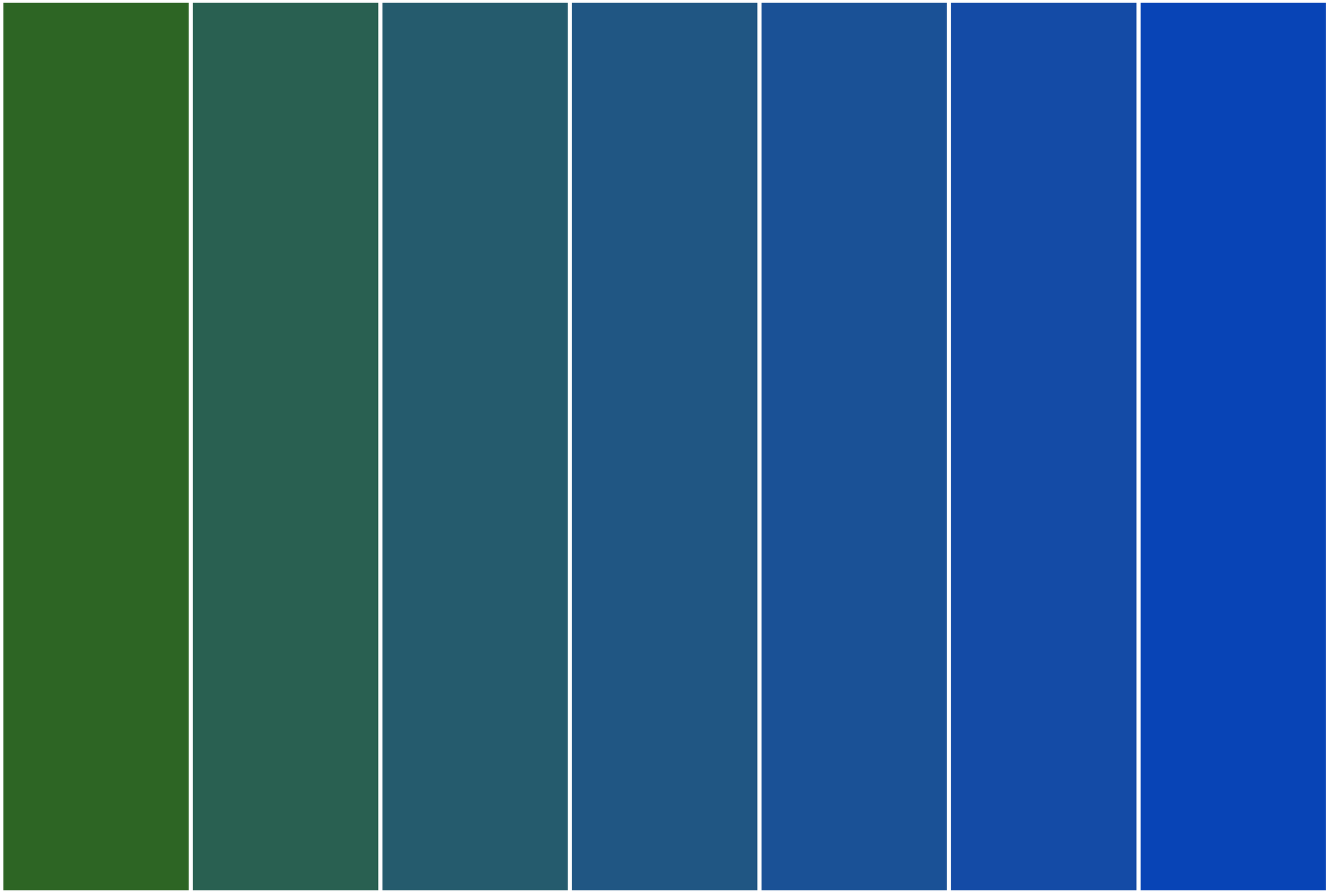
0xFF1587BF 0xFF4499B2 0xFF5FA9A5 0xFF73B896 0xFF84C686 0xFF94D374 0xFFA2E05E



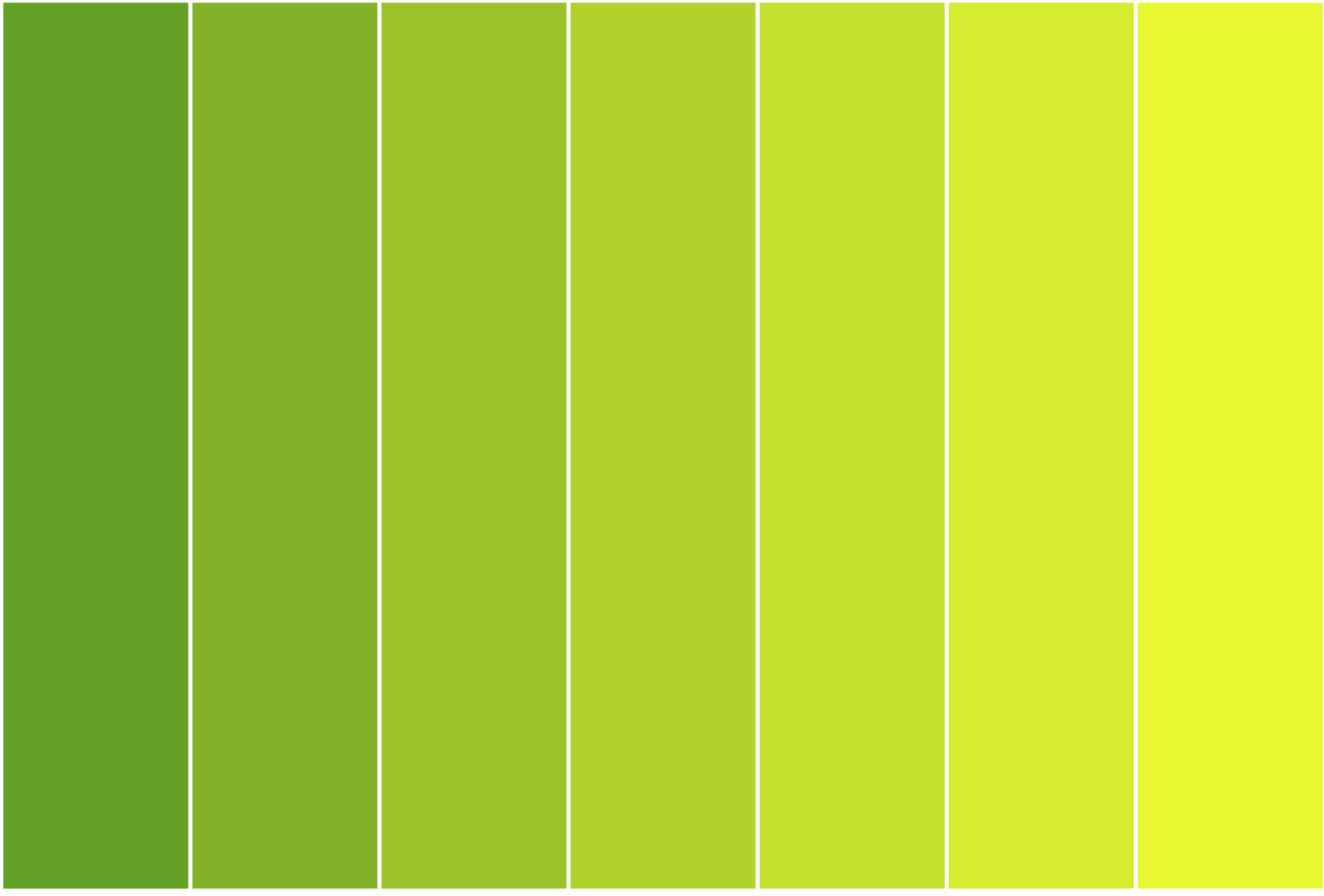
0xFF58870E 0xFF697D11 0xFF777314 0xFF846816 0xFF905B19 0xFF9B4D1B 0xFFA63A1D



0xFF2D6524 0xFF296051 0xFF255B6D 0xFF205683 0xFF1A5196 0xFF144BA6 0xFF0844B6



0xFF63A126 0xFF82B228 0xFF9BC22A 0xFFB1D12C 0xFFC5DF2E 0xFFD6EC30 0xFFE7F831



```
// recipe for color modulation

SecureRandom random = new SecureRandom();

// prepare the first color
int min = 0;
int max = 255;
int r1 = random.nextInt(max-min+1)+min;
int g1 = random.nextInt(max-min+1)+min;
int b1 = random.nextInt(max-min+1)+min;
color col1 = color(r1, g1, b1);

// then prepare the second color
int r2 = random.nextInt(max-min+1)+min;
int g2 = random.nextInt(max-min+1)+min;
int b2 = random.nextInt(max-min+1)+min;
color col2 = color(r2, g2, b2);

// make preliminary calculations
float numsteps = 7;
float step =  (1.0)/(numsteps-1);
float c1_weight = 1; // the starting weight for the first color
float c2_weight = 0;  // the starting weight for the second color
float gapsize = 10;
float rectwidth = pgwidth/numsteps - gapsize;

pushMatrix();
translate(margin, margin);
noStroke();
```

```
for(int i = 0; i < numsteps; i++) {
  // get a weighted average of the red, green, blue channels
  float mixedred = sqrt((sq(red(col1))*c1_weight +sq(red(col2))*c2_weight));
  float mixedgreen = sqrt((sq(green(col1))*c1_weight +sq(green(col2))*c2_weight));
  float mixedblue = sqrt((sq(blue(col1))*c1_weight +sq(blue(col2))*c2_weight));

  // prepare the color strip
  color stripcol = color(mixedred, mixedgreen, mixedblue);
  fill(stripcol);
  float posx = (rectwidth + gapsize)*i;

  // lay down the color strip
  rect(posx, 0, rectwidth, pgheight);

  text("0x"+hex(stripcol), posx, height*0.05 - margin);
  c1_weight -= step;
  c2_weight += step;

}

popMatrix();
```

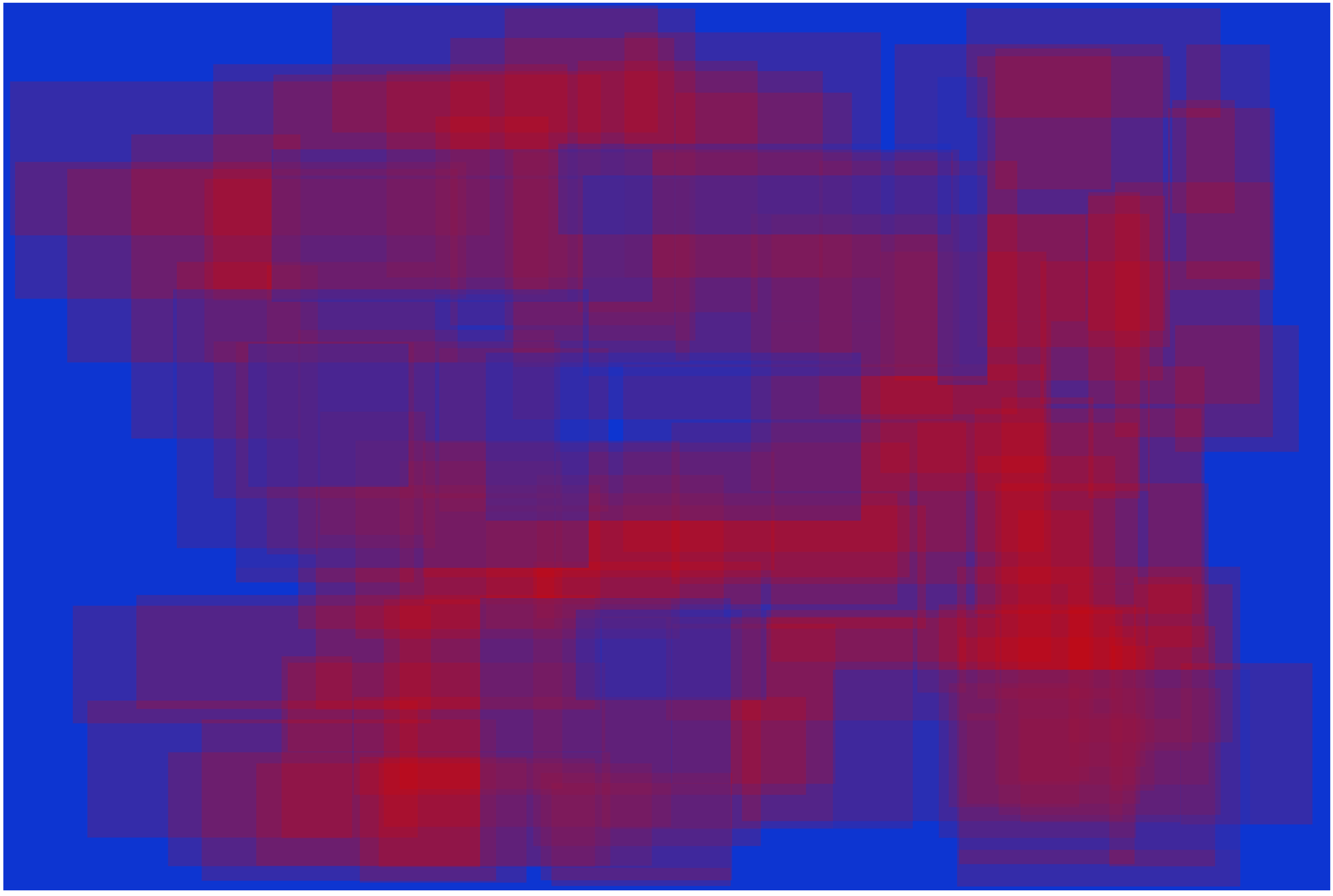
Chapter 06: Bridging Colors

I had vodka the other day (it was in a mixed drink). It was not unlike drinking the liquid embodiment of a migraine. Within a few gulps I felt my head spinning, as the world around me swirled together into a conglomerate of colors and lights. Before I knew it I had my elbows on the table, holding my forehead in my hands while squinting at my dad seated on the other side.

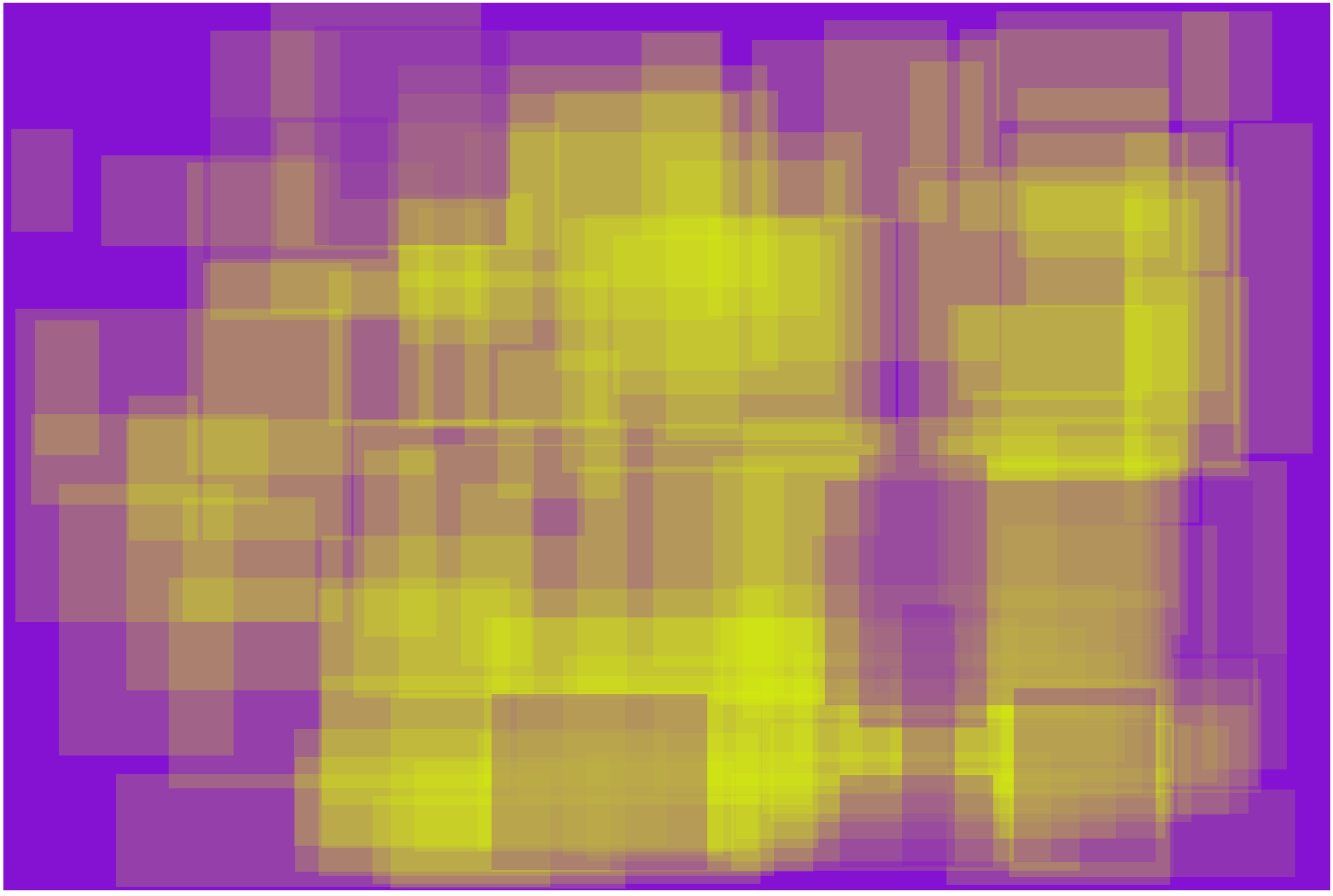
"So yeah, I think in general my alcohol tolerance is pretty good," I began. A few moments later: "I'm getting kinda sleepy."

If my experience that day could be somehow represented as an infographic, it would look exactly like the next few studies.

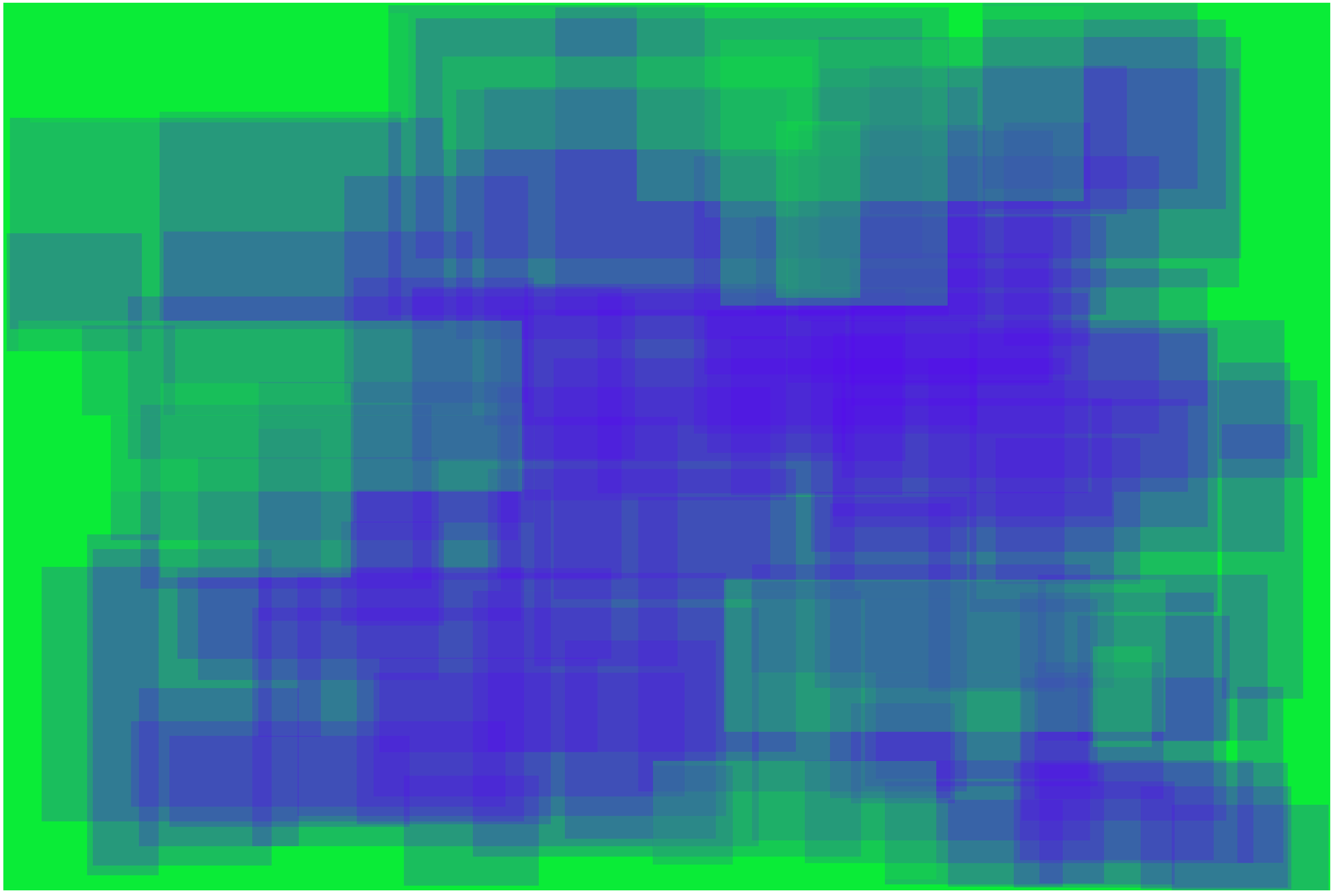
0xFFD60805 0xFF0D35D1



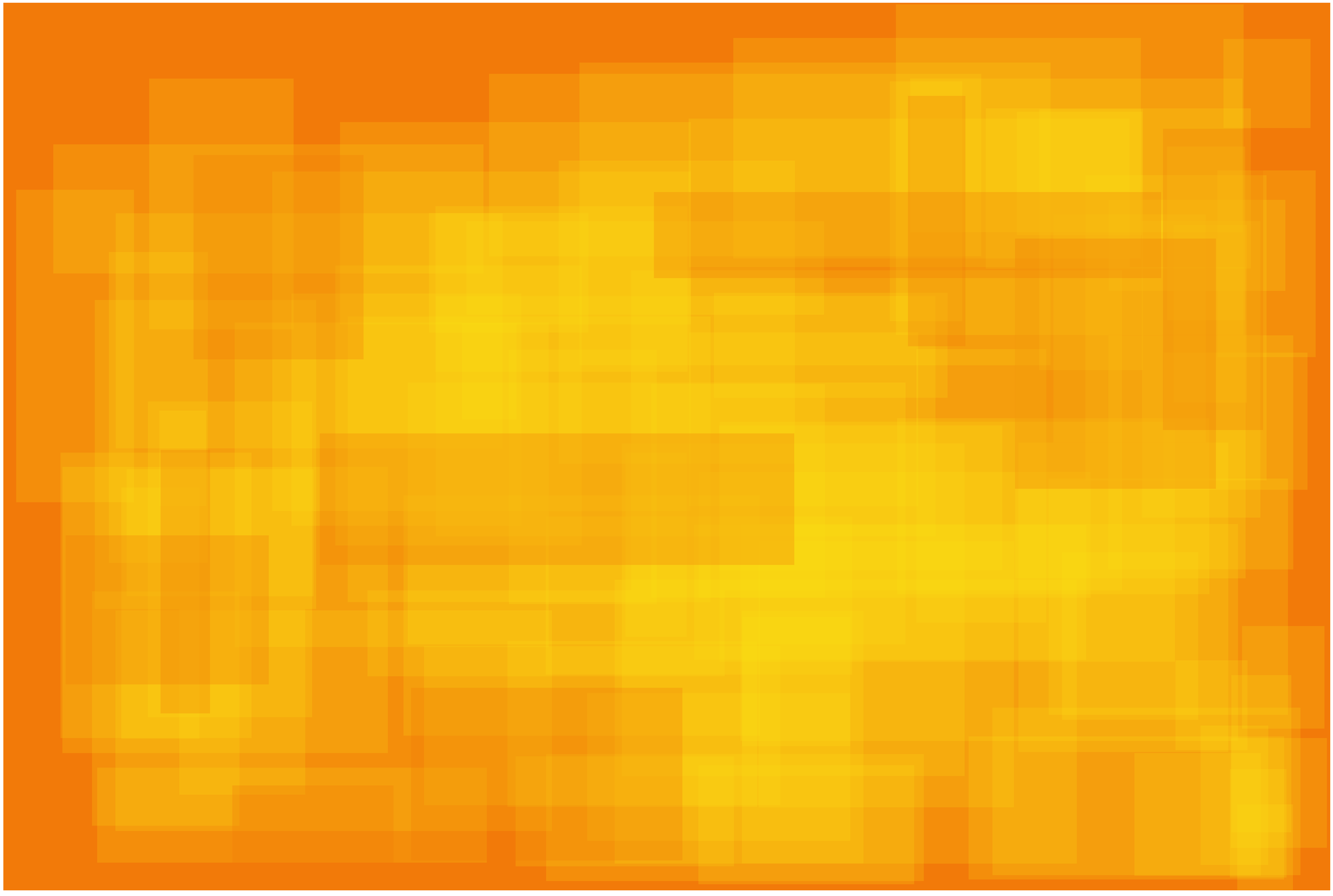
0xFFD9F906 0xFF8512D2



0xFF5C02F9 0xFF0AEC37



0xFFFEE317 0xFFF27A09



```
// recipe for using bridging colors

// switch working color mode to HSB
// before preparing the foreground color
colorMode(HSB,360,100,100);
SecureRandom random = new SecureRandom();

// prepare essential random ingredients
int min = 0;
int max = 360;
int h1 = random.nextInt(max-min+1)+min;

int min2 = 0;
int max2 = 180;
int h2 = random.nextInt(max2-min2+1)+min2;

color col1 = color(h1, random(90,100), random(80,100));

rectMode(CORNER);
pushMatrix();
translate(margin, margin);
noStroke();

// glaze with the background color, which is opposite from the foreground color
// with a touch of randomness for enhanced flavor
color bg = color((hue(col1)+180+h2)%360, random(90,100), random(80,100));

fill(bg);
rect(0, 0, pgwidth,pgheight);
```

```
// randomly add 90 slices of the foreground color
for(int i = 0; i < 90; i++) {
  float posX = random(0, pgwidth-200);
  float posY = random(0, pgheight-200);

  float rectw = random(100, min(1200, pgwidth-posX));
  float recth = random(200, min(800, pgheight-posY));

  // the slices can overlap with each other,
  // but they must be thin
  // like prosciutto
  fill(col1, 50);
  rect(posX, posY, rectw,recth);

}
```

```
// for balance, top with a few thin slices of the background color
for(int i = 0; i < 10; i++) {
  float posX = random(0, pgwidth-200);
  float posY = random(0, pgheight-200);

  float rectw = random(100, min(1200, pgwidth-posX));
  float recth = random(200, min(800, pgheight-posY));

  fill(bg,70);
  rect(posX, posY, rectw,recth);

}
```

Chapter 07: Same Value Studies

You may have noticed that some of the studies you've seen so far work pretty effectively, while others don't seem to work at all.

You may be waiting for me to provide you with some sort of explanation for that.

You may be stopping and asking yourself, 'Is this really ART???'

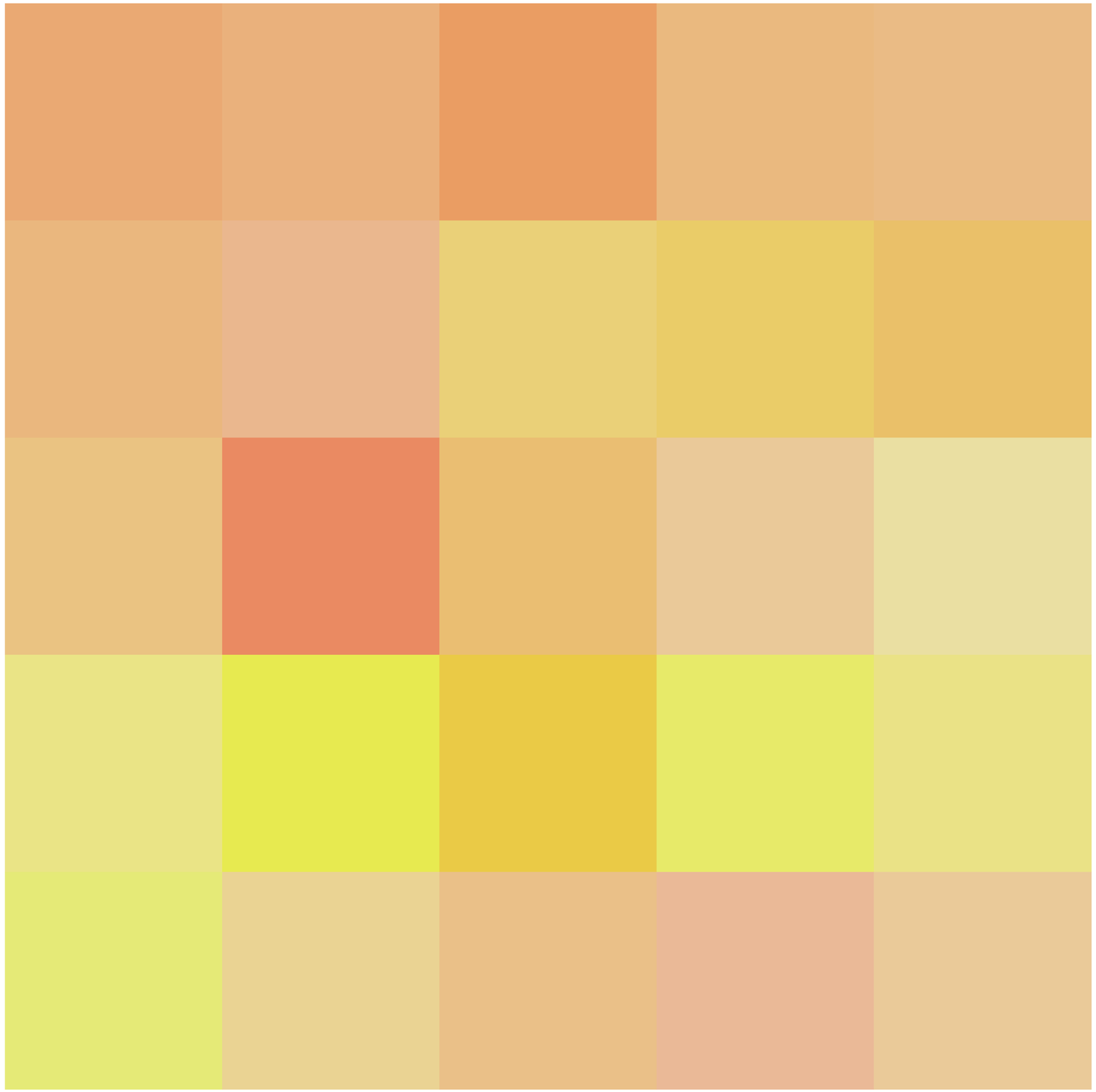
You may be debating whether you should just skip this useless expository text or read on.

You may be sitting on your living room couch right now--winding and rewinding your mind--trying to pull yourself back to the time when you were able to feel the warm sun on your face without having to worry about getting burned later, when Tuesdays were good and you didn't miss the tingle of someone else's breath on your skin.

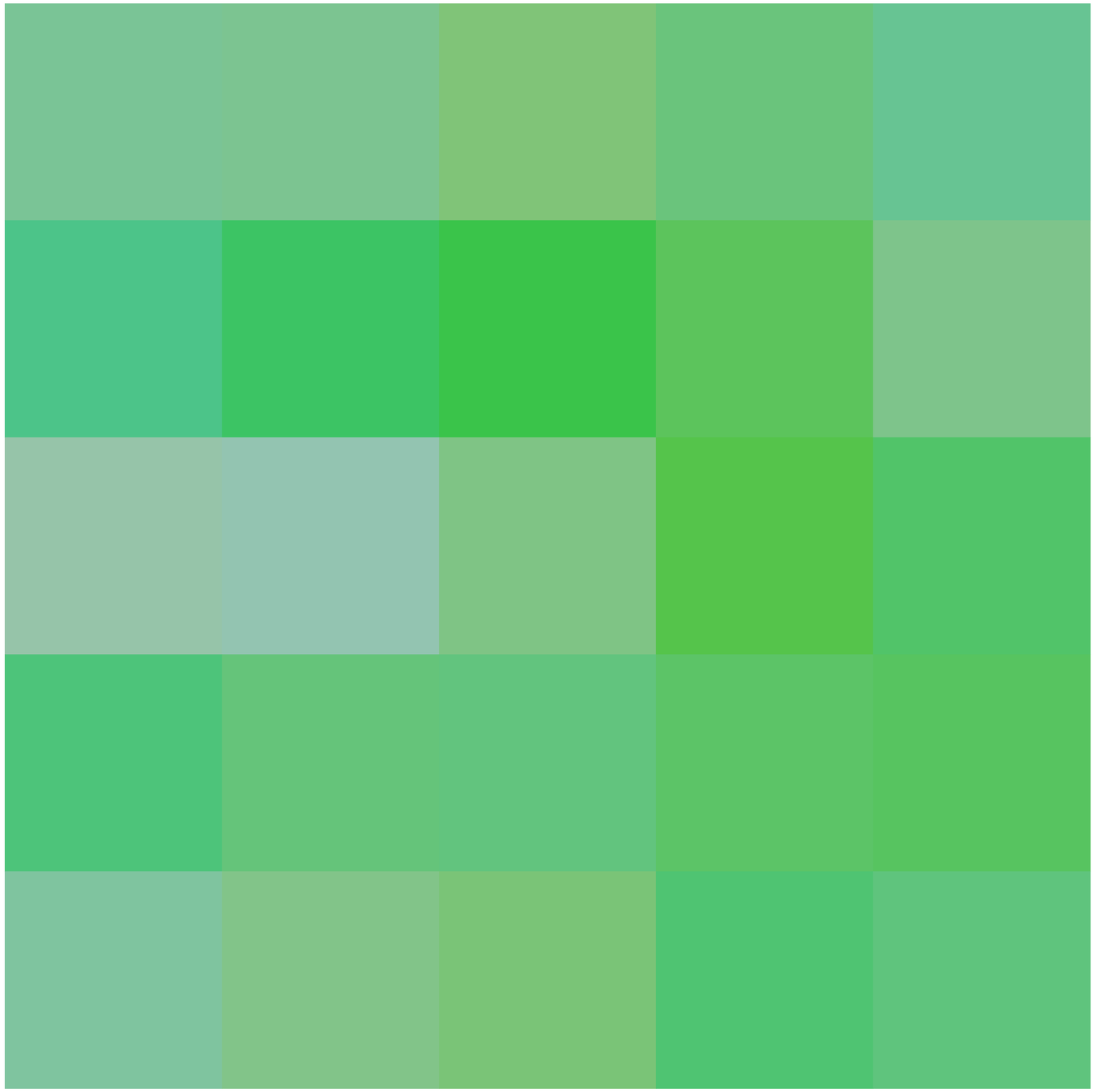
You may be wondering what that previous sentence has to do with anything.

You may not even care. But even if you don't, thanks for caring enough to read on.

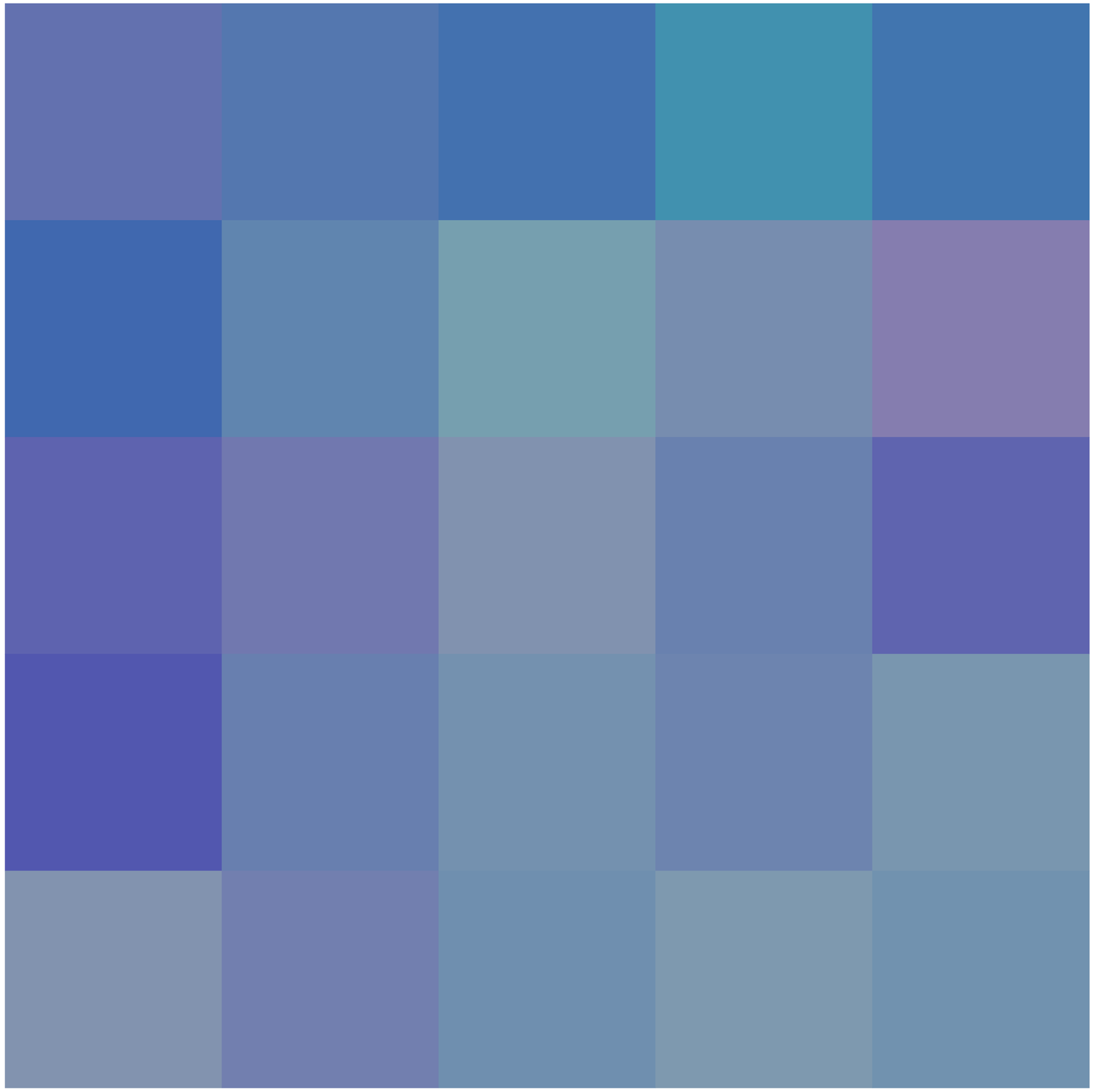
value: 92



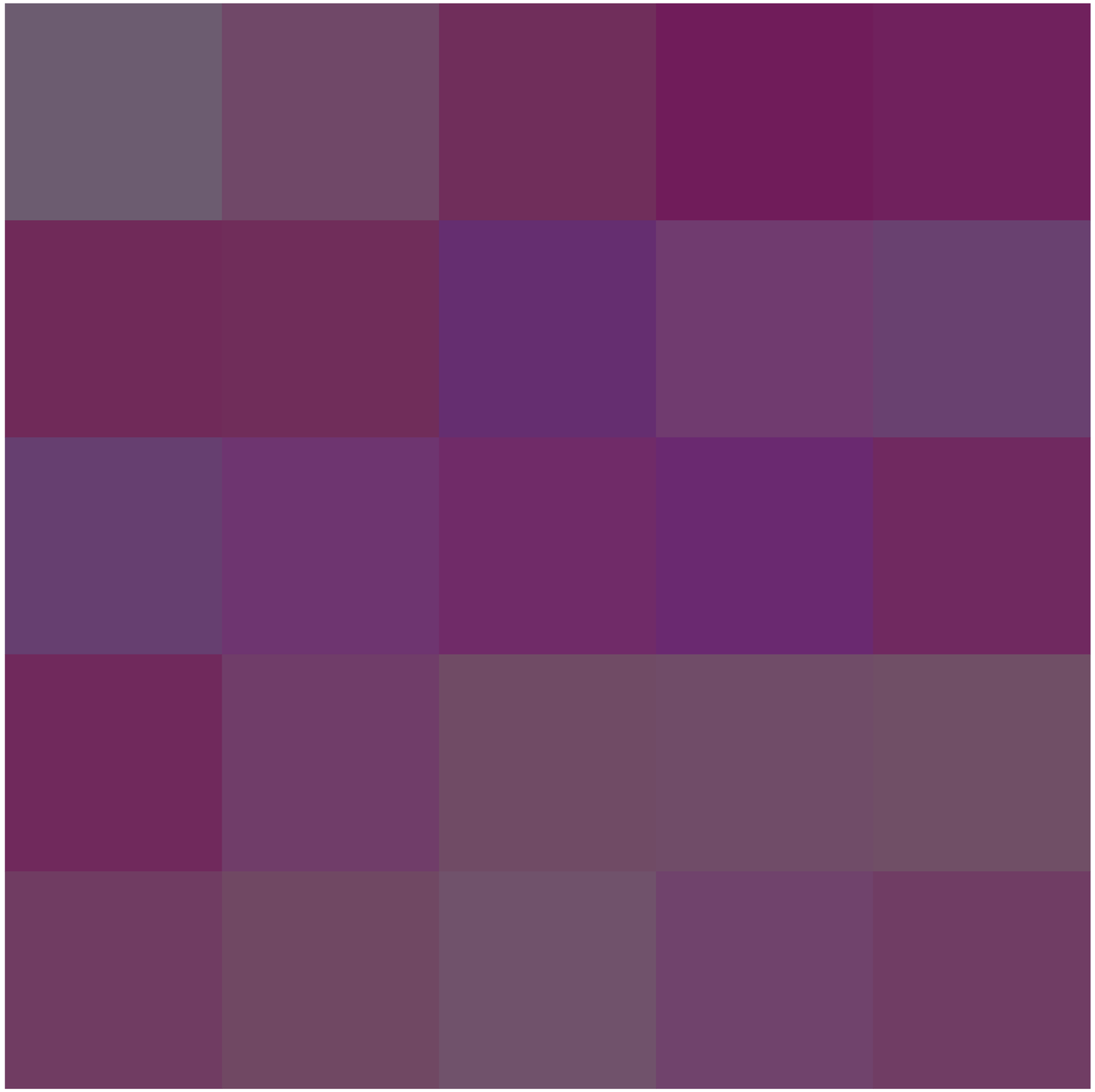
value: 77



value: 69



value: 44



```
// recipe for same value studies (a la Johannes Itten)

// we're working in HSB space now!
// RGB is for posers
colorMode(HSB,360,100,100);
SecureRandom random = new SecureRandom();

int min = 40;
int max = 95;
int value = random.nextInt(max-min+1)+min;

// boundaries for saturation
int satmin = 0;
int satmax = 100;

// noise variables
int noisemin = 0;
int noisemax = 100;
float hoff = random.nextInt(noisemax-noisemin+1)+noisemin;
float soff = random.nextInt(noisemax-noisemin+1)+noisemin;

rectMode(CORNER);
pushMatrix();
translate(margin + 500, margin);
noStroke();

// sprinkle squares in a matrix
for(int row = 0; row < 5; row++) {
  for(int col = 0; col < 5; col++) {
```

```
    // instead of using random, I'm mapping the colors to noise
    // noise is a bit more 'organic' than random
    // huemin and huemax are just parameters I defined at the beginning of the function
    float hue = map(noise(hoff), 0,1, huemin,huemax);
    float sat = map(noise(soff), 0,1, satmin,satmax);
    color squarecol = color(hue, sat, value);

    fill(squarecol);
    stroke(squarecol);
    rect(400*col, 400*row, 400, 400);

    // add a touch of magic numbers
    hoff += 0.55;
    soff += 0.3;
}
  }

  popMatrix();
  colorMode(RGB);

  textFont(mainFont, 60);
  fill(0);
  text("value: " +value, width*0.05, height*0.05);
```

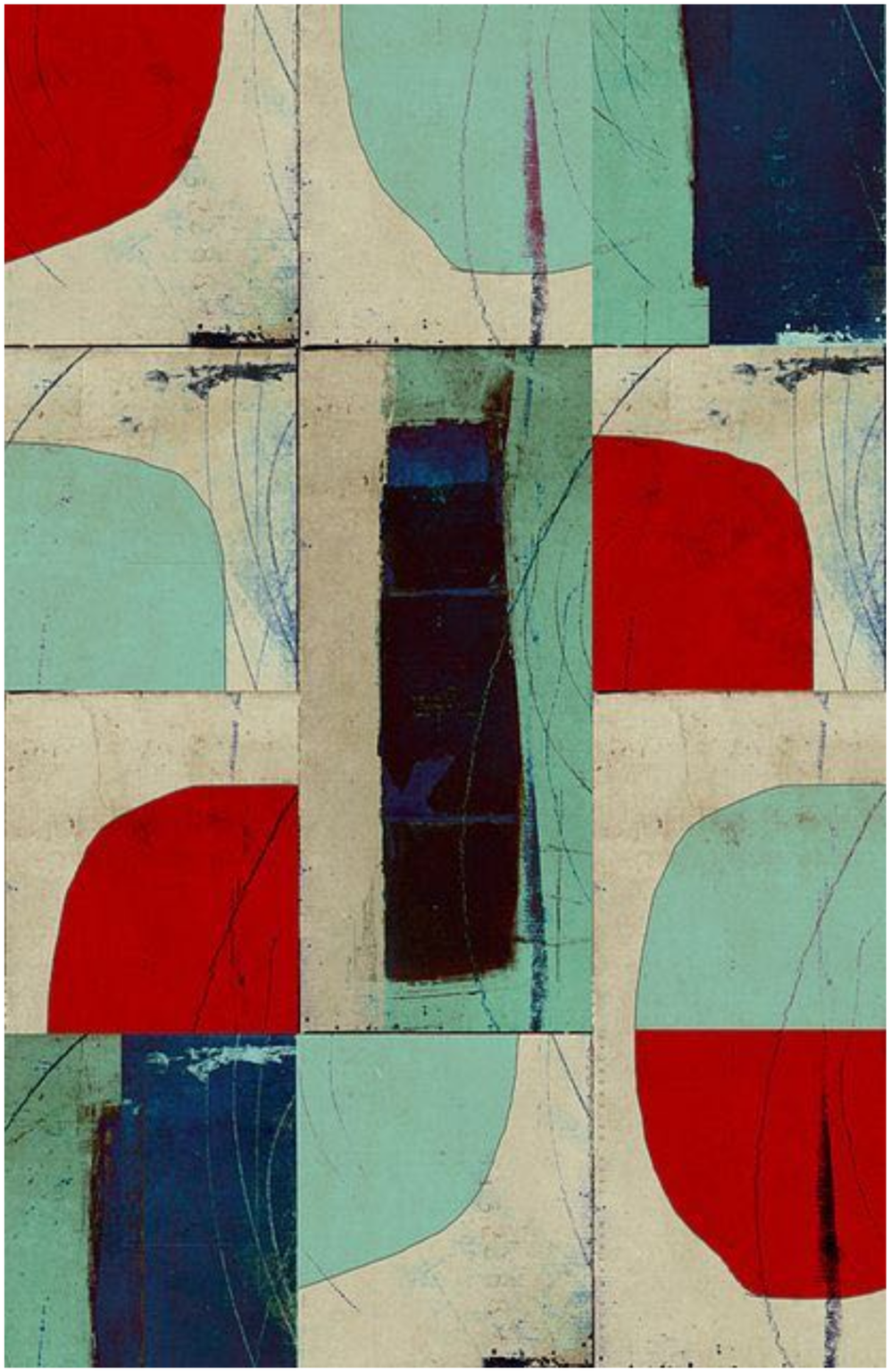
Appendix: Proportional Palettes

I'm not very good at goodbyes. I'm not referring to the ones you say to your friends after you've had dinner with them or the ones you might have said to your parents after they dropped you off to your first day of school--I'm talking about the real goodbyes that creep up on you and then seize you by the chest the minute you realize you may not even be able to say 'hello' to that person again. A lot of the goodbyes I've made came in the form of poorly-worded emails, uncomfortable hugs, or long silences on the phone. Goodbyes, from my experience, tend not to be very 'good'--they're unsettling, they're awkward, they're emotionally taxing, but most of all they're just downright sad. So I've given up on trying to make my goodbyes 'good'. Instead, I decided that if I could manage to make them less sad, that's about as good as I can make it.

I hope the goodbye here is not the sad kind I have just described. But in case it is, here's my attempt at making it less sad: I don't know who you are, whether you're a collegue or a friend or a family member or someone I've never even talked to before. Though that does not even matter, because this book is not about who you are. And it's not really even about color theory; this book is ultimately about who I am and how I understand some tiny subset of the universe. Regardless of whether you've read this book from the very beginning or decided to skip to the very end, I just wanted to say thanks. Thanks for thinking this book was worth more than just a few seconds of your time. Thanks for caring about what this existentially trapped twenty-something-year-old had to say. Thanks for reading.

Thanks for staying.

artist:
Linda Vachon



0xFFA50202
0xFF85120E
0xFFBDB89B
0xFF1D1521
0xFF84A990
0xFF89B49C
0xFF627E68
0xFFCBC6AA
0xFF465249
0xFF170609
0xFF1A263E
0xFF6DA487
0xFF141E37

artist:
Gustav Klimt

0xFFB77946
0xFF875336
0xFFDF9D51
0xFFA02F20
0xFFDAB383
0xFFF6D583
0xFFAA8A7A
0xFFCFB7AB
0xFFC3642B
0xFF683B27
0xFFC2A89D
0xFF281C17
0xFF876859
0xFFF3B75A
0xFF634E3C
0xFFD88235
0xFF4F3C3D
0xFF1F1B28
0xFF130D0B

artist:
Alphonse Mucha

0xFFF8F6F3
0xFF3B2631
0xFF411E25
0xFF160C10
0xFF6A372A
0xFF9D5727
0xFFC87228
0xFFE28E35
0xFFD67222
0xFFBE5B1B
0xFF663747
0xFF8B3914
0xFFB66A6A
0xFF995A60
0xFF501E0E
0xFF586C3F
0xFF55213B
0xFFA54E4F

artist:
Paul Klee

0xFF6D553B
0xFF52432A
0xFF36301E
0xFF73411F
0xFFA53119
0xFF50341A
0xFF827A6F
0xFFB67306
0xFFC77C05
0xFFBE0528
0xFFA40427
0xFF3D5426
0xFF5B5D4F

artist:
Piotr Jablonski

0xFF8E6805
0xFFB19001
0xFFBD9E02
0xFFF2EA05
0xFFFAF707
0xFF452C17
0xFF1D1109
0xFF745C42
0xFF0A0503
0xFF46362C
0xFF947D60
0xFFA6906A
0xFF1B3227
0xFF19464F

artist:
James Jean

0xFF3E2D43
0xFF68555D
0xFF476D77
0xFF227F8A
0xFF68939B
0xFFFAF8BF
0xFFA59B9C
0xFF8A777B
0xFF9A4C4F
0xFF288583
0xFF140C27
0xFF1F1D3D
0xFFA63637
0xFFC63839
0xFFE83333
0xFFCB2828
0xFF0E0823
0xFFC8C0C1
0xFF373E57
0xFFD6D1D3

artist:
James Jean

0xFFDBCECB
0xFFBCA5A1
0xFF997A70
0xFF7A5848
0xFF4F3029
0xFF251414
0xFFE6DDDB
0xFEA98A37
0xFF674449
0xFFAB8F91

artist:
Tatsuyuki Tanaka

0xFFA99383
0xFFBDA38E
0xFF867162
0xFF5D4A3C
0xFF402F24
0xFF201810
0xFF0D0C07

artist:
Alphonse Mucha

0xFFECD2B8
0xFFDDAC8B
0xFFF6E3D0
0xFE5D230C
0xFE2C0B06
0xFF611303
0xFF8F320E
0xFFD8986A
0xFFAF6234
0xFFB54D1F
0xFFCF7F4D
0xFFEAAB89
0xFE784517
0xFFCD6531
0xFFE49362
0xFFA82305

```
// recipe for palette grabber, which creates a proportional
// color palette from a given source image

ArrayList<Pixel> colors = new ArrayList<Pixel>();
// tolerance for the similarities between two colors
float tolerance = 50.0;
// keeps track of all the colors found
int totalCount = 0;
// tolerance for the proportion of a color in an image
float proptolerance = 0.005;
PImage[] images = new PImage[folderimages];
String[] names = new String[folderimages];

public class Pixel {
  public ArrayList pixelgroup;
  public int count;


float colorDist(color c1, color c2) {
  float r = red(c1) - red(c2);
  float g = green(c1) - green(c2);
  float b = blue(c1) - blue(c2);

  return sqrt(sq(r) + sq(g) + sq(b));


void addPixel(color currpix) {
  Pixel p = new Pixel();
  // head of  the color group
```

```
  p.pixelgroup.add(currpix);
  p.count = 1;
  colors.add(p);
  totalCount++;


boolean notBlackorWhite(color col)  {
  return ((col != 0.0) && (col != 255.0));


void palettePage(int imgindex)  {
  colorMode(RGB,255,255,255);

  totalCount = 0;

// rinse palette before use
for(int i = colors.size()-1; i >= 0 ; i--)  {
   colors.remove(i);
}
PImage sourceimg = images[imgindex];

sourceimg.resize(0,  (int)pgheight);
int iwidth = sourceimg.width;
int iheight = sourceimg.height;
color temp = sourceimg.get(0,0);
float palettepos = sourceimg.width + 50;

// extract colors from image
// and add them to the palette
```

```
for(int j = 0; j < iwidth; j++) {
  color currpix = sourceimg.get(i,j);

  if(notBlackorWhite(currpix)) {
    if(colors.size() == 0) {
      addPixel(currpix);
    }

    else {
      int idx;
      boolean foundMatch = false;
      int prevIdx = 0;
      float minDist = 0.0;

      // look through existing colors in the palette
      for(idx = 0; idx < colors.size(); idx++) {
        float currDist = colorDist(currpix, (Integer) colors.get(idx).pixelgroup.get(0));
        // if the color is in the palette
        if(currDist < tolerance) {
              // increase the count for the palette color closest to
              // the current pixel
              if(foundMatch && (currDist < minDist)) {
                colors.get(idx).count++;
                colors.get(prevIdx).count--;
                colors.get(idx).pixelgroup.add(currpix);

                prevIdx = idx;
                minDist = currDist;
              }
```

```
            colors.get(idx).count++;
            colors.get(idx).pixelgroup.add(currpix);
            totalCount++;
            foundMatch = true;
            prevIdx = idx;
            minDist = currDist;
          }
        }
}

if(!foundMatch) {
  addPixel(currpix);

}
      }
    }
  }
}

int netCount = totalCount;
for(int idx = 0; idx < colors.size(); idx++) {
  float prop = colors.get(idx).count / ((float)totalCount);

  if(prop < proptolerance) {
    netCount -= colors.get(idx).count;
  }
}
rectMode(CORNER);
```

```
pushMatrix();
// place image a little to the right of your margins
translate(margin + 700, margin);
image(sourceimg, 0,0);
// place palette beside image
for(int i = 0; i < colors.size(); i++) {
  float prop = colors.get(i).count / ((float)netCount);

  // if there is enough of the color in the image, display it
  if(prop >= proptolerance) {
    float rheight = pgheight*prop;
    float totred = 0;
    float totgreen = 0;
    float totblue = 0;

    // calculate an average value of the colors in each group
    int groupsize = colors.get(i).pixelgroup.size();
    for(int j = 0; j < groupsize; j++) {
      Pixel thepixel = colors.get(i);
      color currcolor = (Integer) thepixel.pixelgroup.get(j);
      totred += red(currcolor);
      totgreen += green(currcolor);
      totblue += blue(currcolor);
    }
    color paletteColor = color(totred/groupsize, totgreen/groupsize,totblue/groupsize);

    stroke(paletteColor);
    fill(paletteColor);
```

```
    // top it off with the hex value for the colors
    textFont(mainFont, 40);
    fill(paletteColor);
    text("0x"+hex(paletteColor), palettepos + 350, ypos+30);

    ypos += rheight;

  }
}

popMatrix();
```

A special thanks to
Clayton Merrell
and Bob Bingham
and Rafael Abreu-Canedo
for your guidance and support.