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

Table of Contents

00. Introduction.....5
01. 3 Colors Into 4 Part I.....7
02. 3 Colors Into 4 Part II.....15
03. 4 Colors Into 3.....23
04. 5 Colors Into 3.....31
05. Color Modulation.....39
06. Bridging Colors.....49
07. Same Value Studies.....57

Appendix.....65

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.

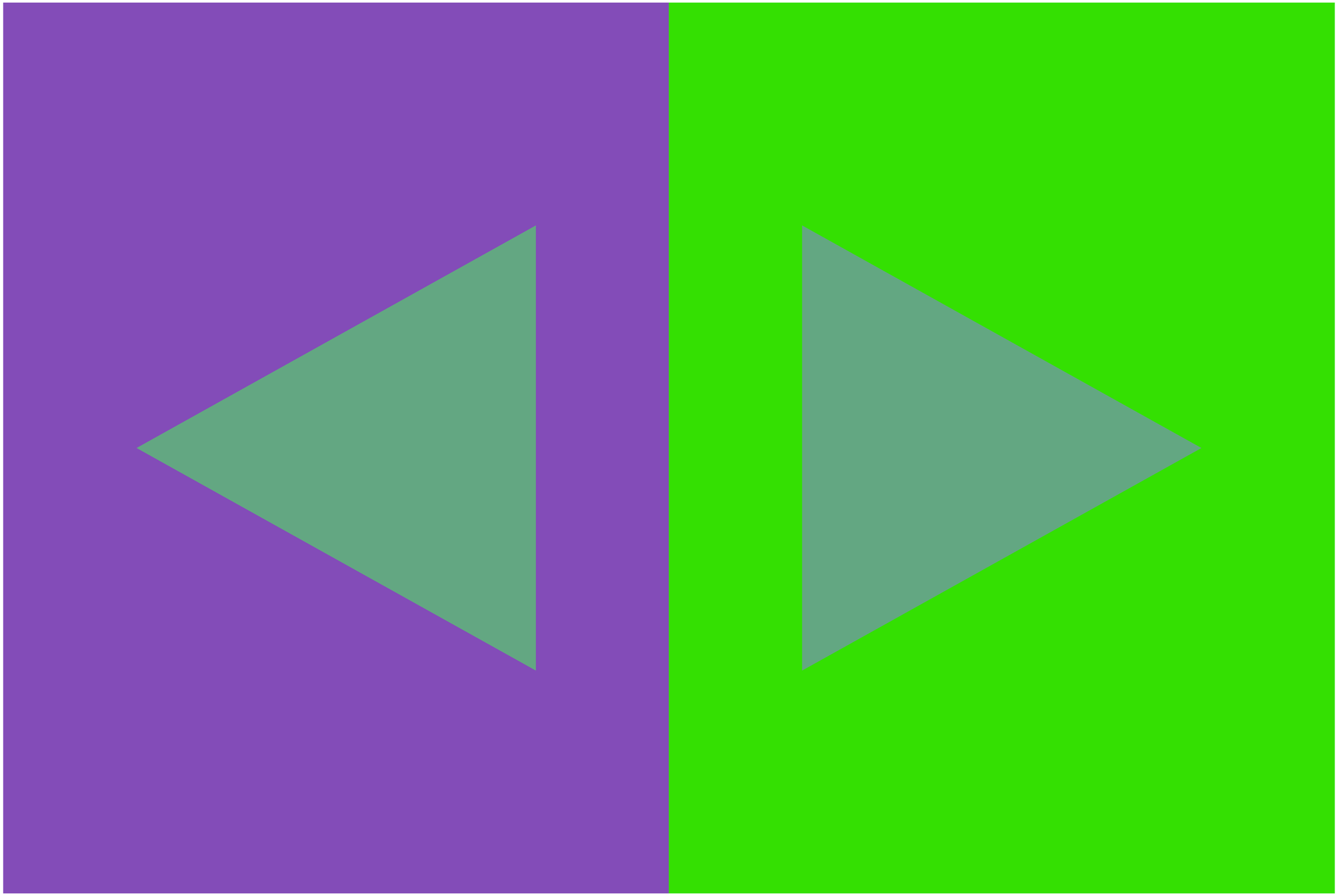
Chapter 01: 3 Colors Into 4 Part 1

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.*

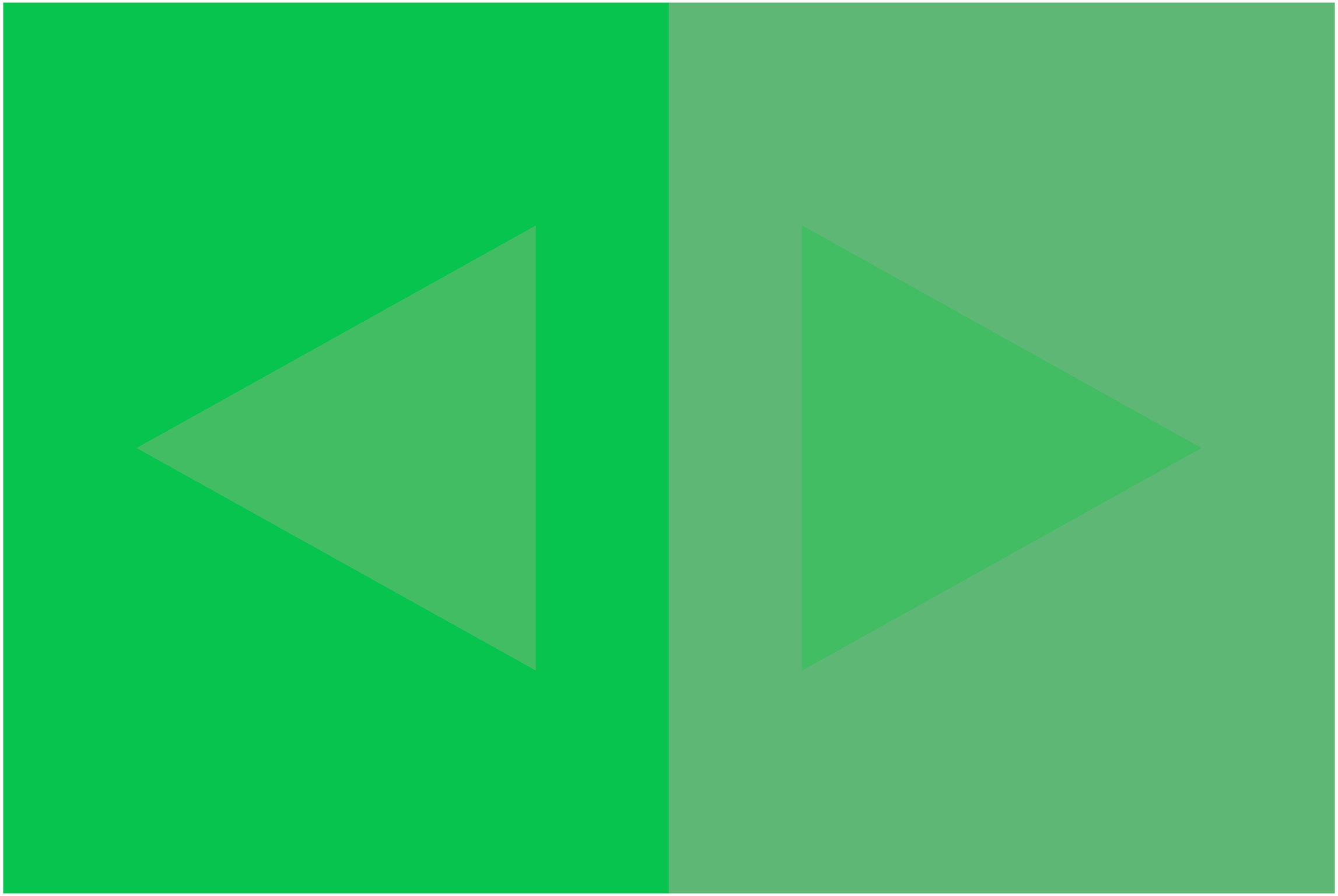
According to color theory, having a color in the background removes that color from the foreground--this is why the dress shown against yellow lighting would make it look less yellow and more blue. I won't get into the specifics of the science behind this because I'm running out of space on this page 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

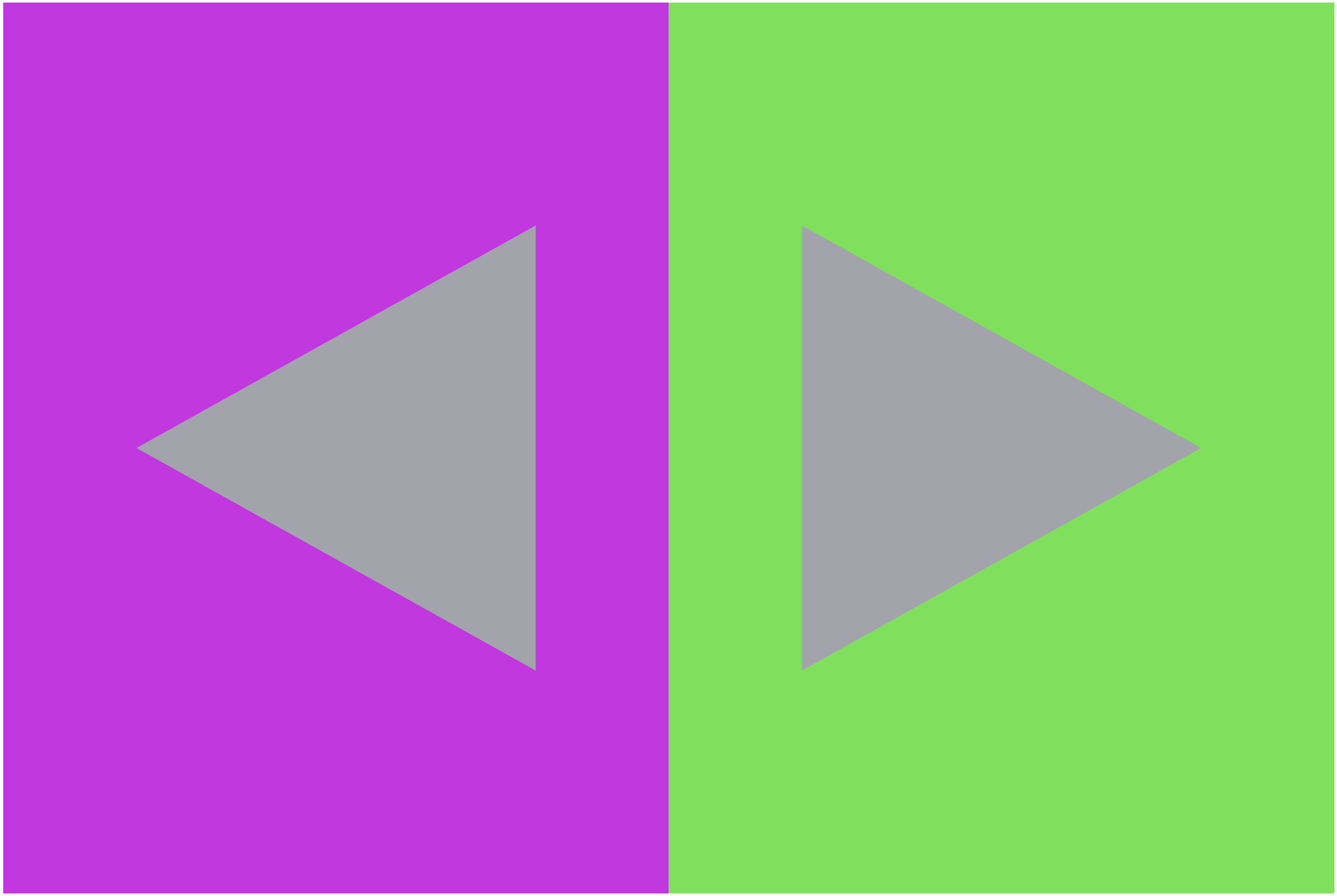
0xFF834CB8 0xFF34E002 0xFF63A782



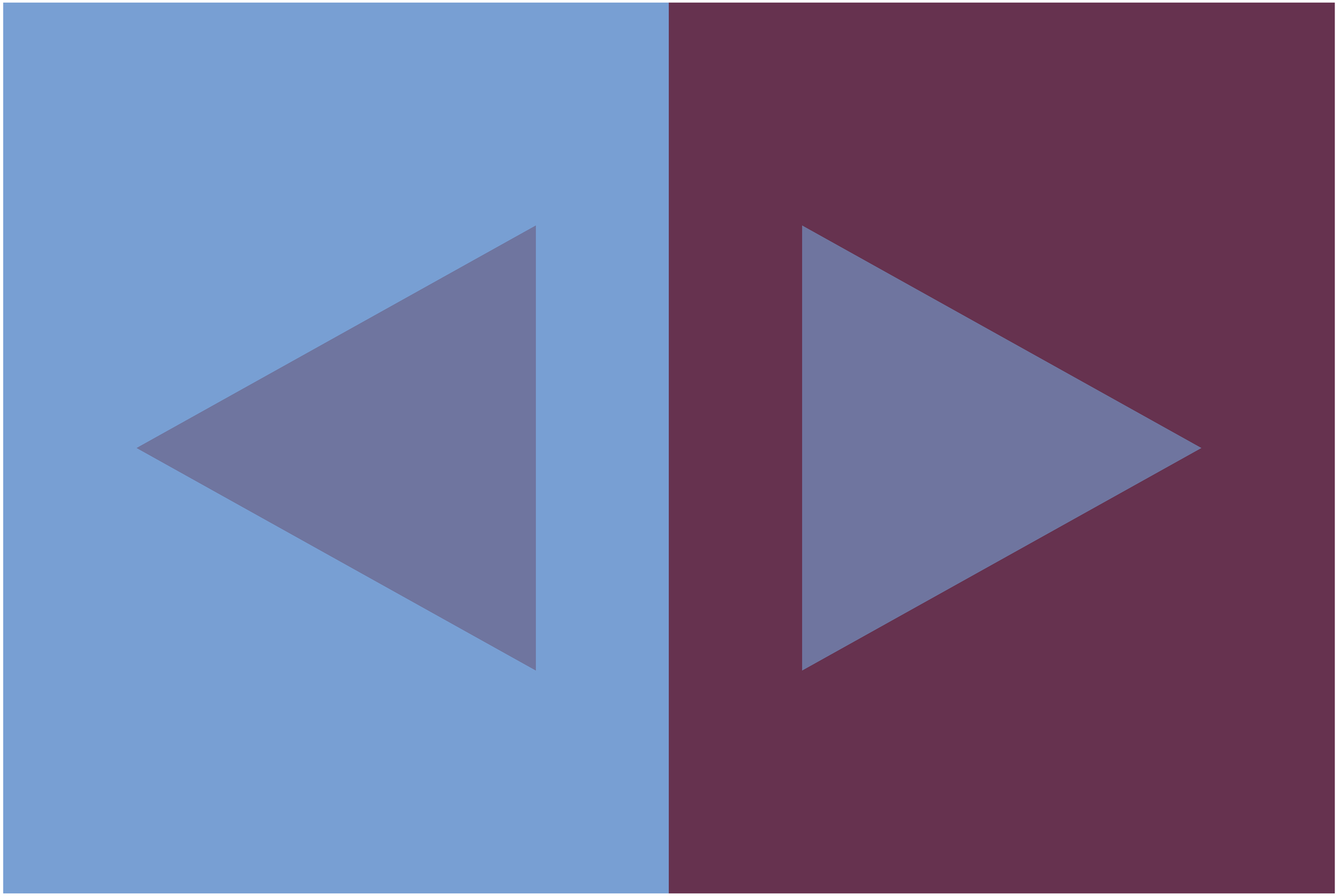
0xFF06C44E 0xFF5EB775 0xFF42BD63



0xFFC138DF 0xFF80E05E 0xFFA3A3AB



0xFF789FD3 0xFF66324F 0xFF6F759F



```
// 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

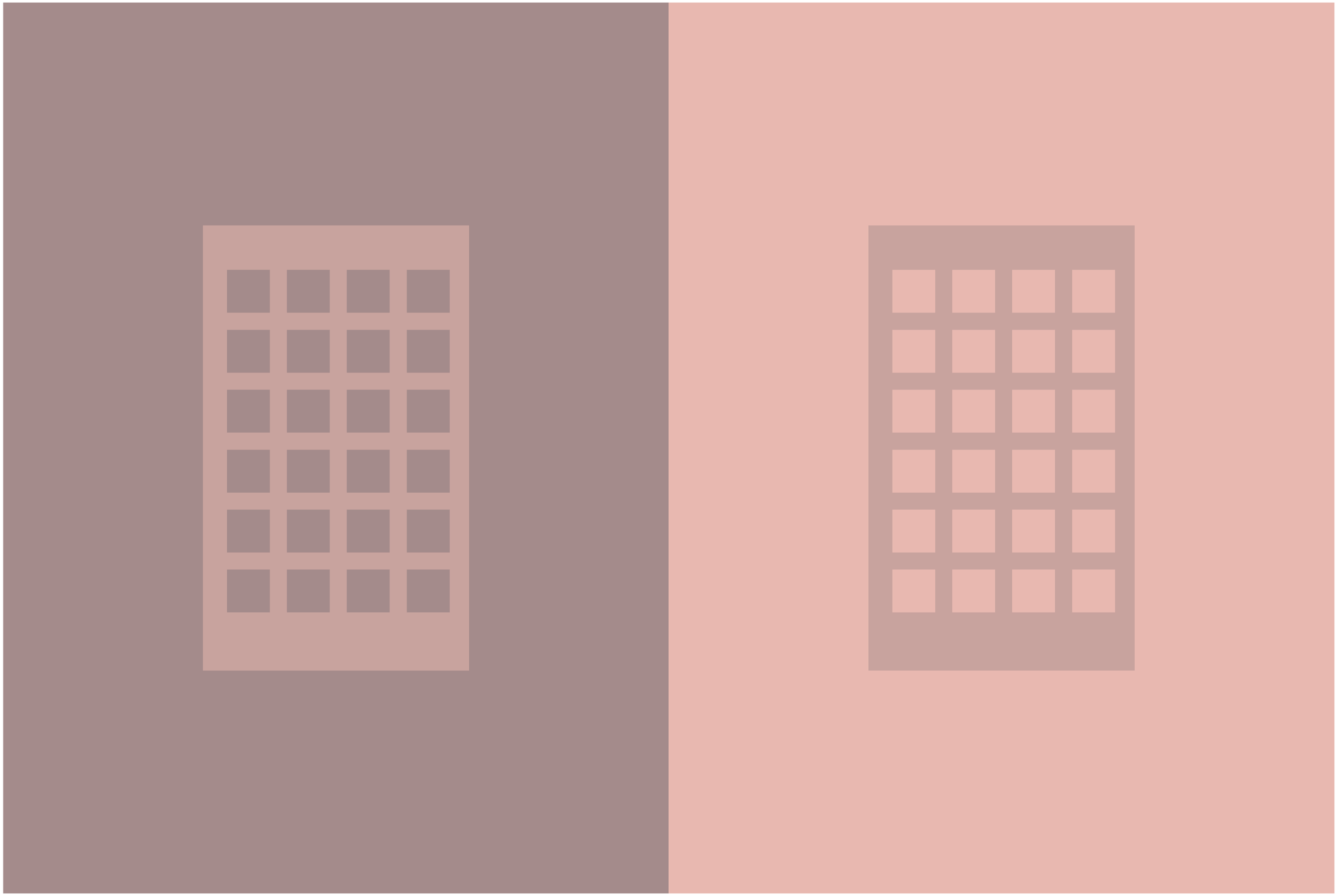
A guy named Pablo Picasso once said, "Why do two colors, put one next to the other, sing? Can one really explain this? No. Just as one can never learn how to paint." I'm sorry Pab, but I'm going to have to disagree with you on this one because:

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

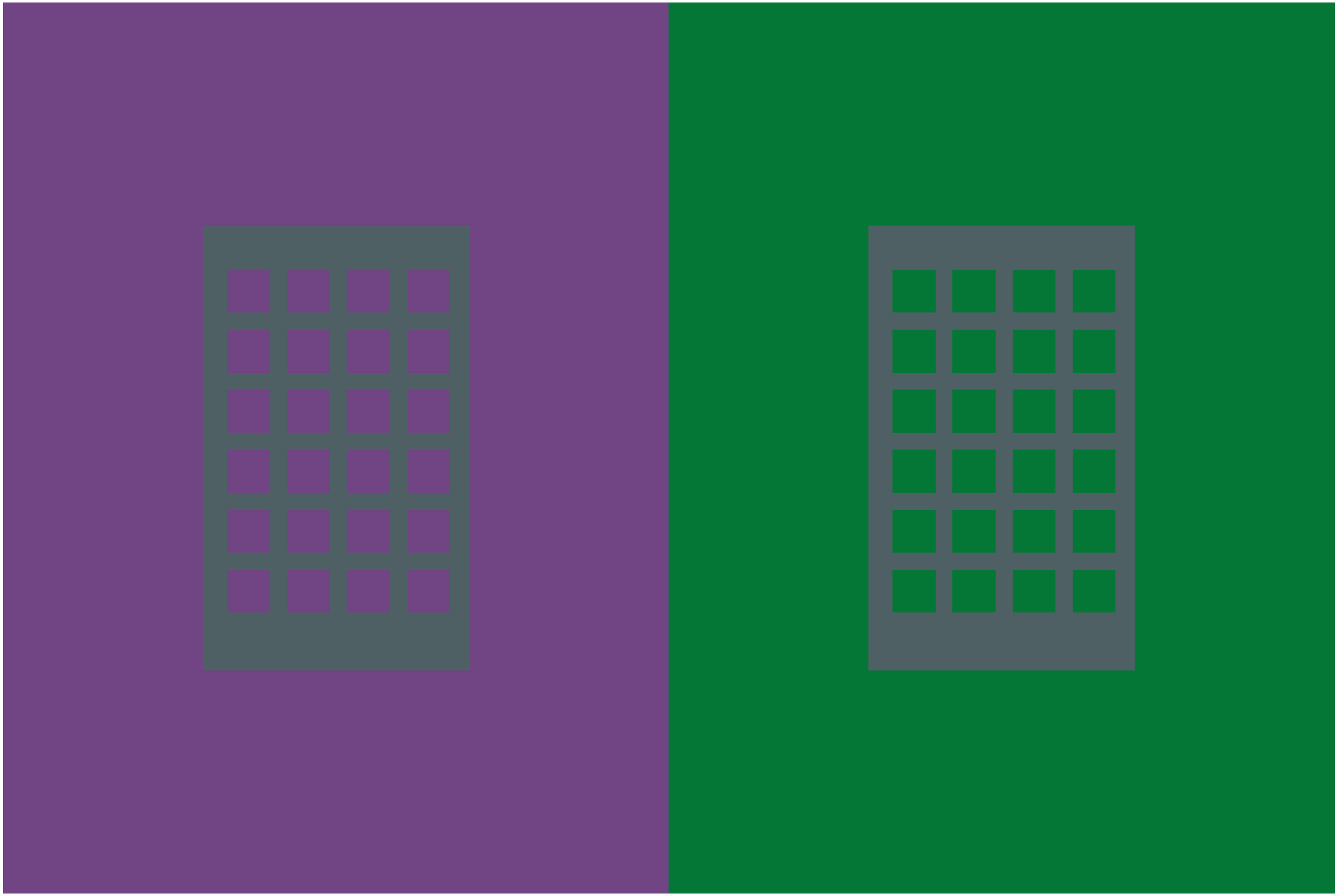
2) Although it's true that color harmony is complex and often tricky to articulate, it is not entirely inexplicable. Color theorists are better equipped to answer this than I am, but one reason why two colors may 'sing' when placed next to each other is because aesthetic responses can be evoked by complementary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

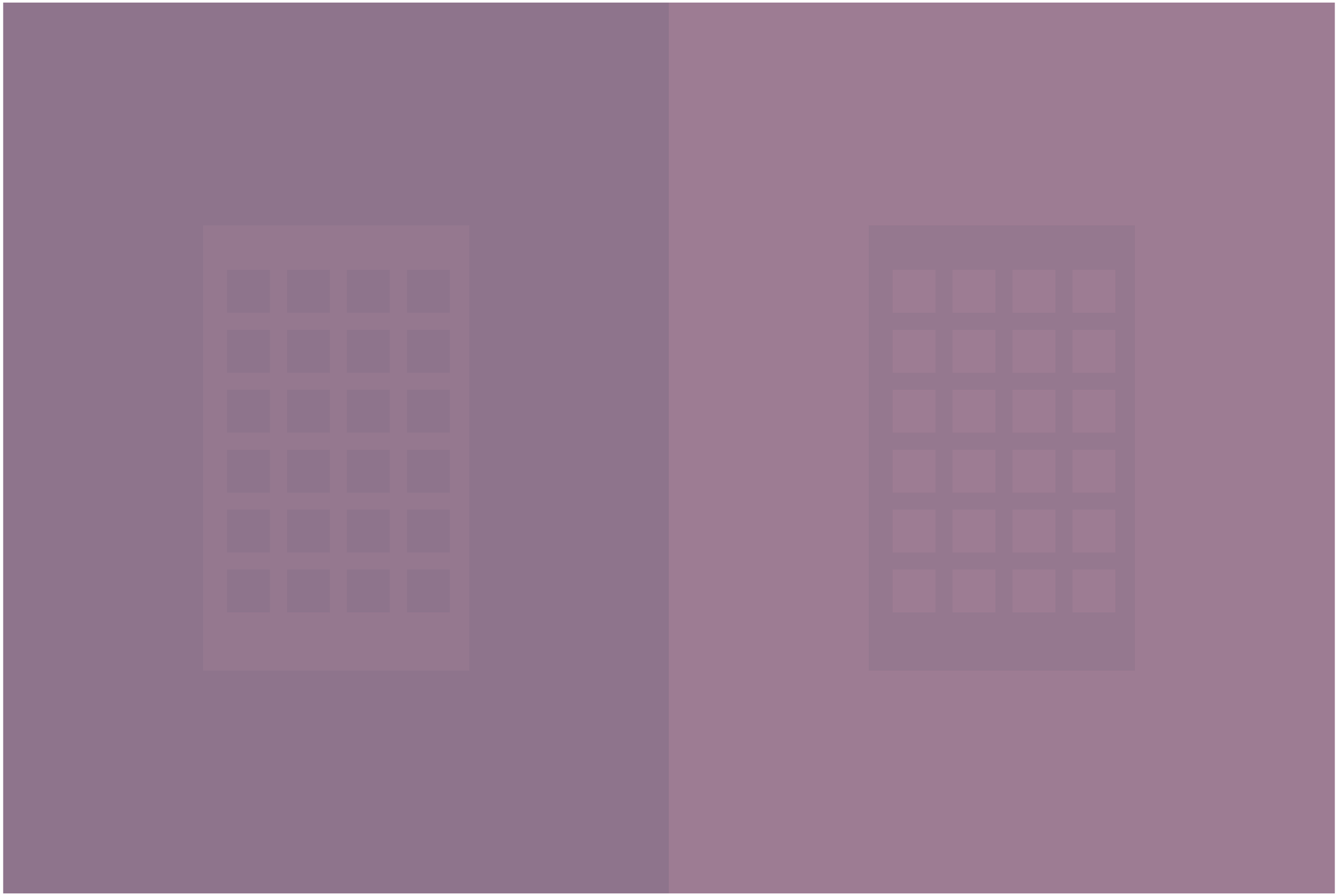
0xFFA48B8B 0xFFE8B8B0 0xFFC8A39E



0xFF714484 0xFF047737 0xFF4F6065



0xFF8E748C 0xFF9D7C93 0xFF95788F



```
// 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();
```

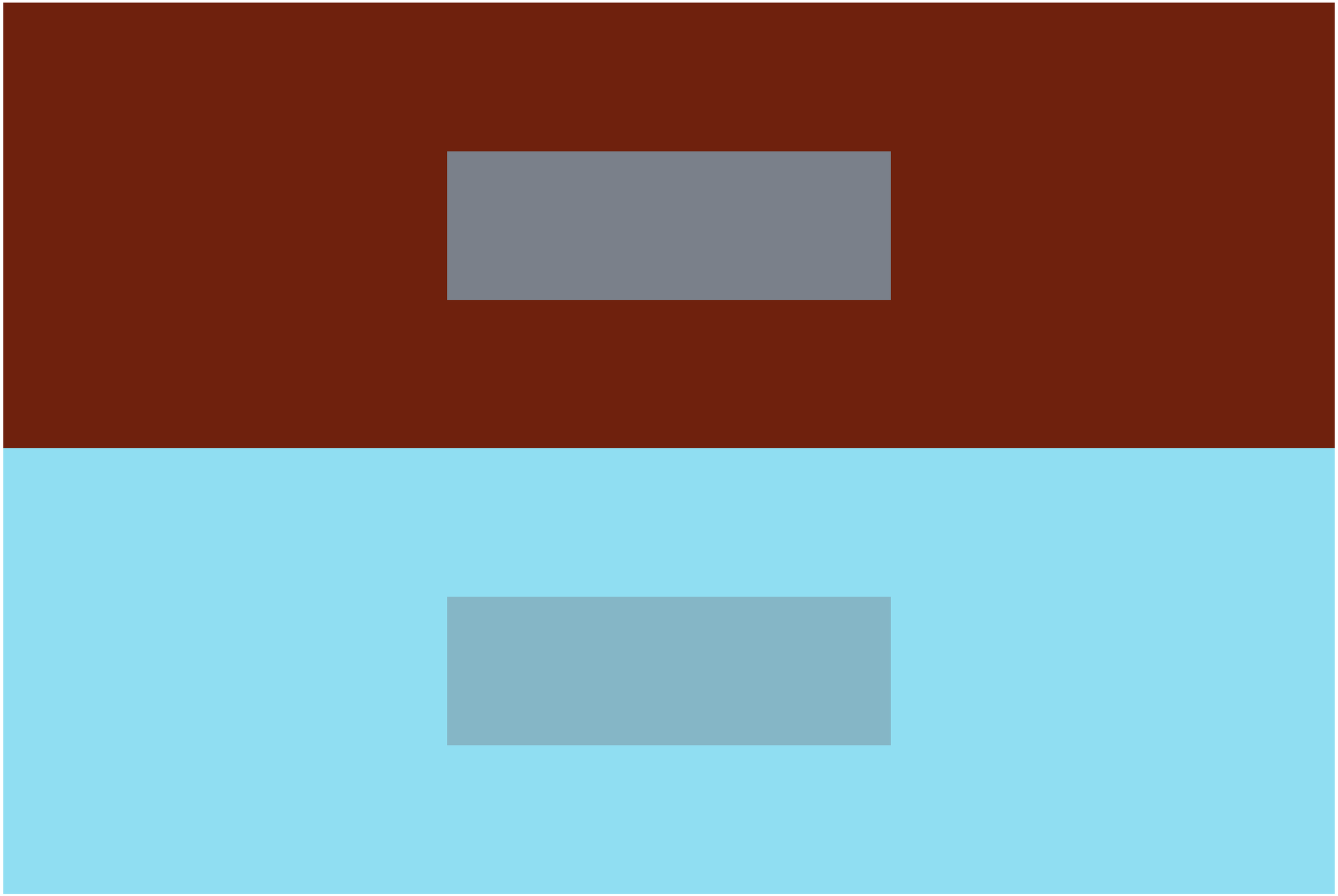
Chapter 03: 4 Colors Into 3

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 a 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?

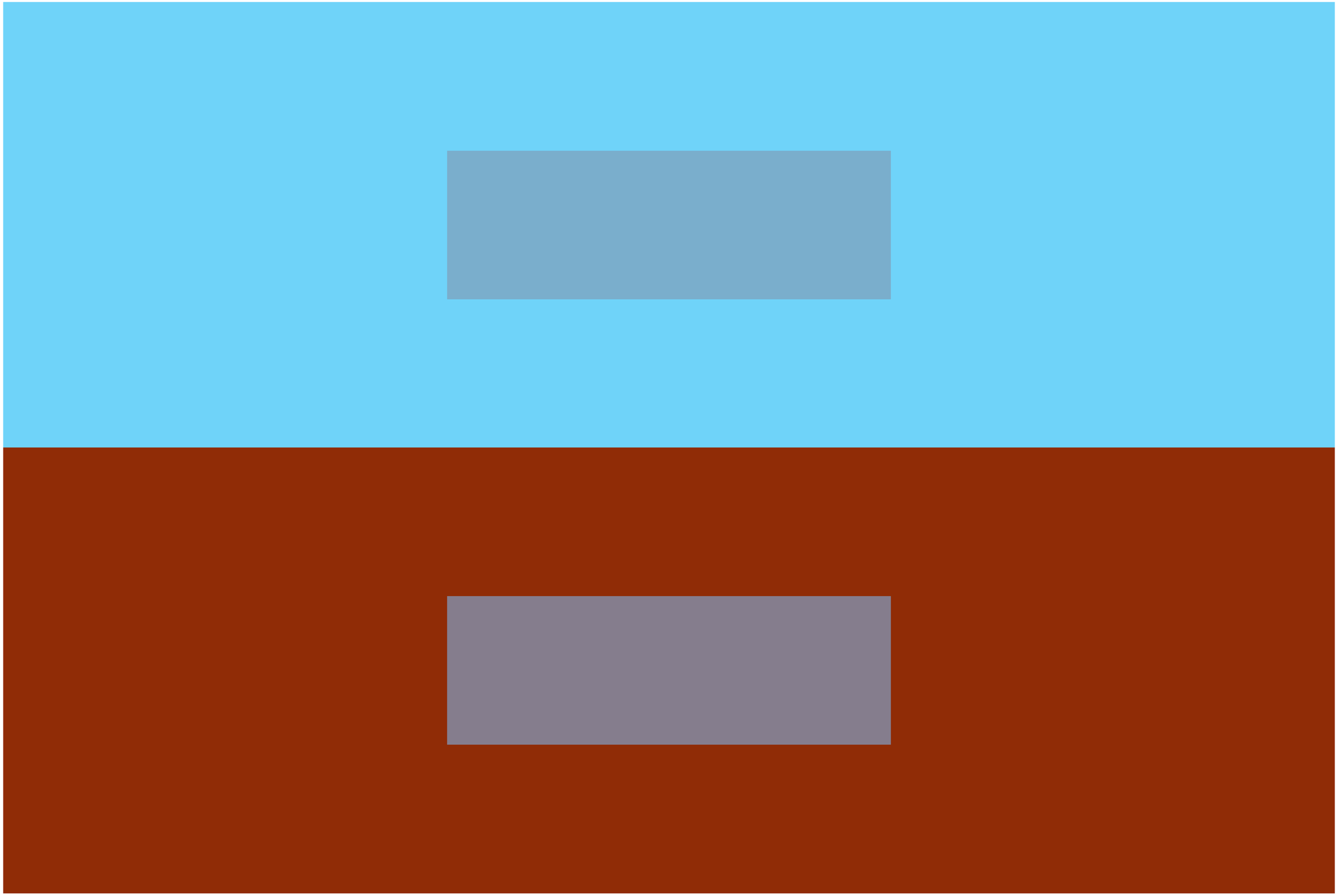
Well...no. It turns out that humans, on average, can only see about 10 million colors in a single viewing condition--so computers really encode 6 million more colors than necessary. So is the answer 10 million, then? Remember that the 10 million refers to the number of colors we can see in a SINGLE viewing 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.

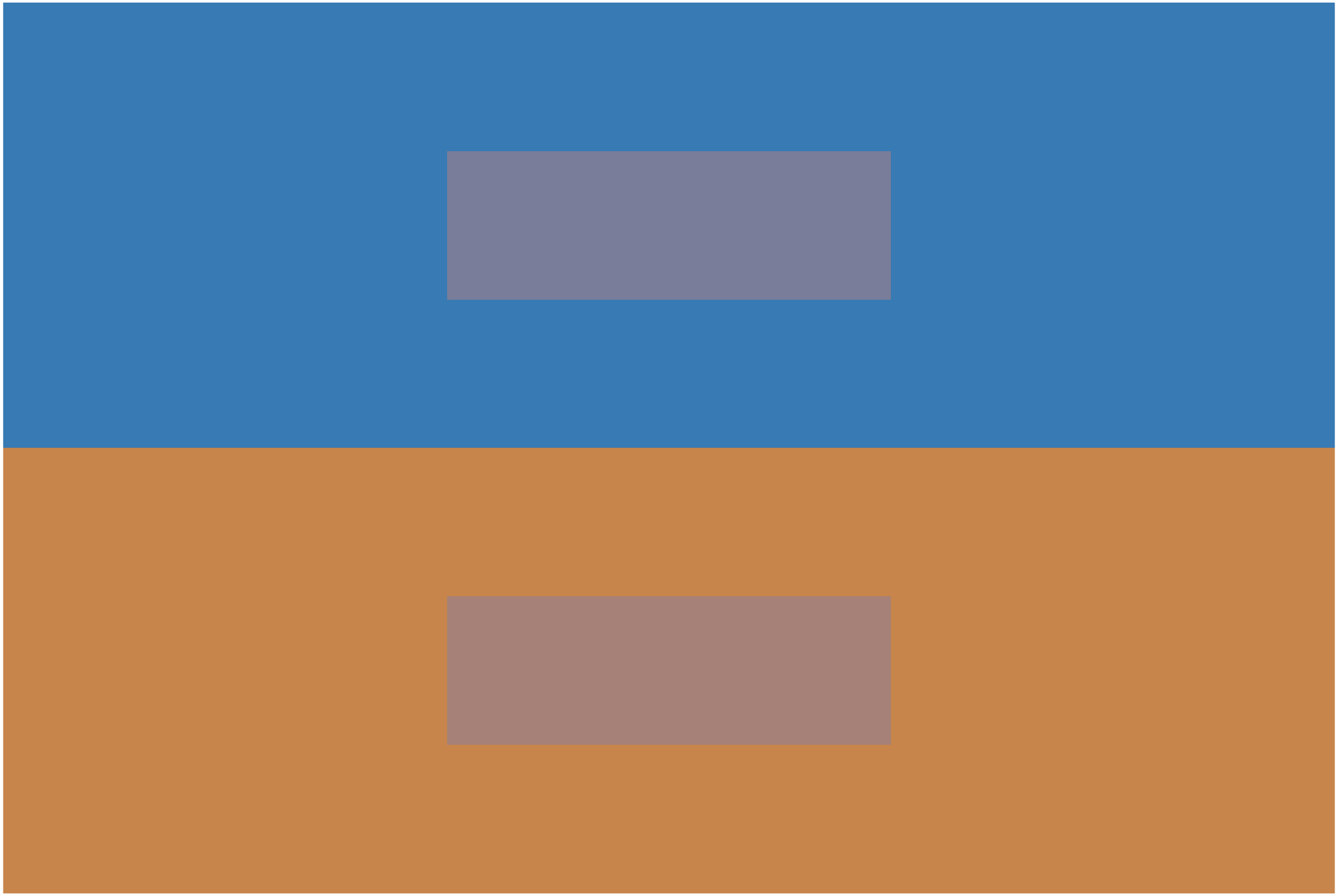
0xFF6F210D 0xFF90DEF2 0xFF7A808A 0xFF85B6C6



0xFF6FD3F9 0xFF902C06 0xFF7AAECC 0xFF857D8D



0xFF387AB3 0xFFC7854C 0xFF7A7D99 0xFFA68177



```
// 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

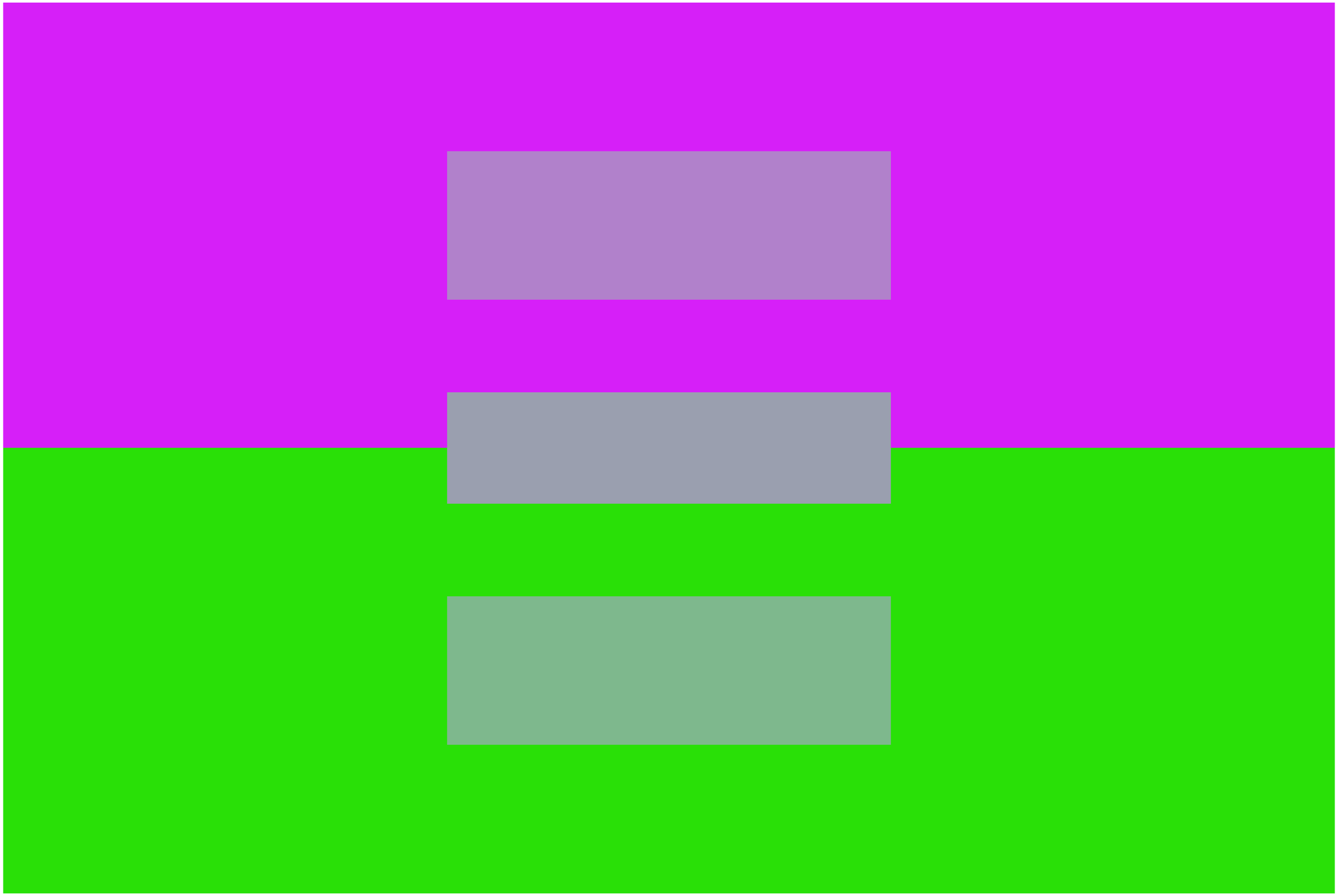
So I probably should be saying something about the next chapter, but I want to tell you a story instead. Last Tuesday, I had to go to my professor's office hours to prepare for an exam. My friend and I made our way to the back corner of the room and sat down, taking out our notes and note-taking instruments 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."

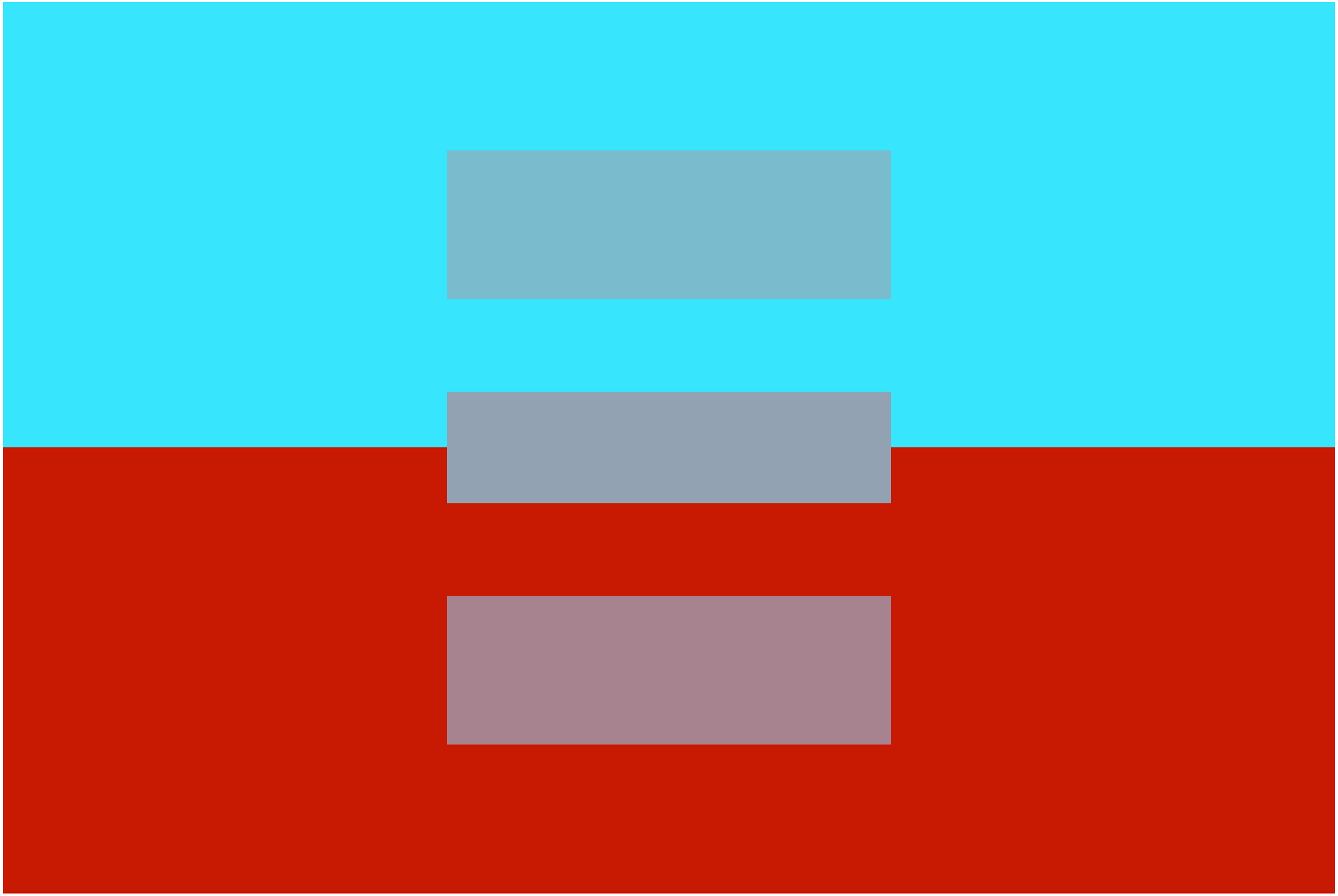
And that's when the room was thrown into a passive-aggressive debate about whether Macs were better than Windows machines (or rather, why Windows machines were better than Macs).

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.

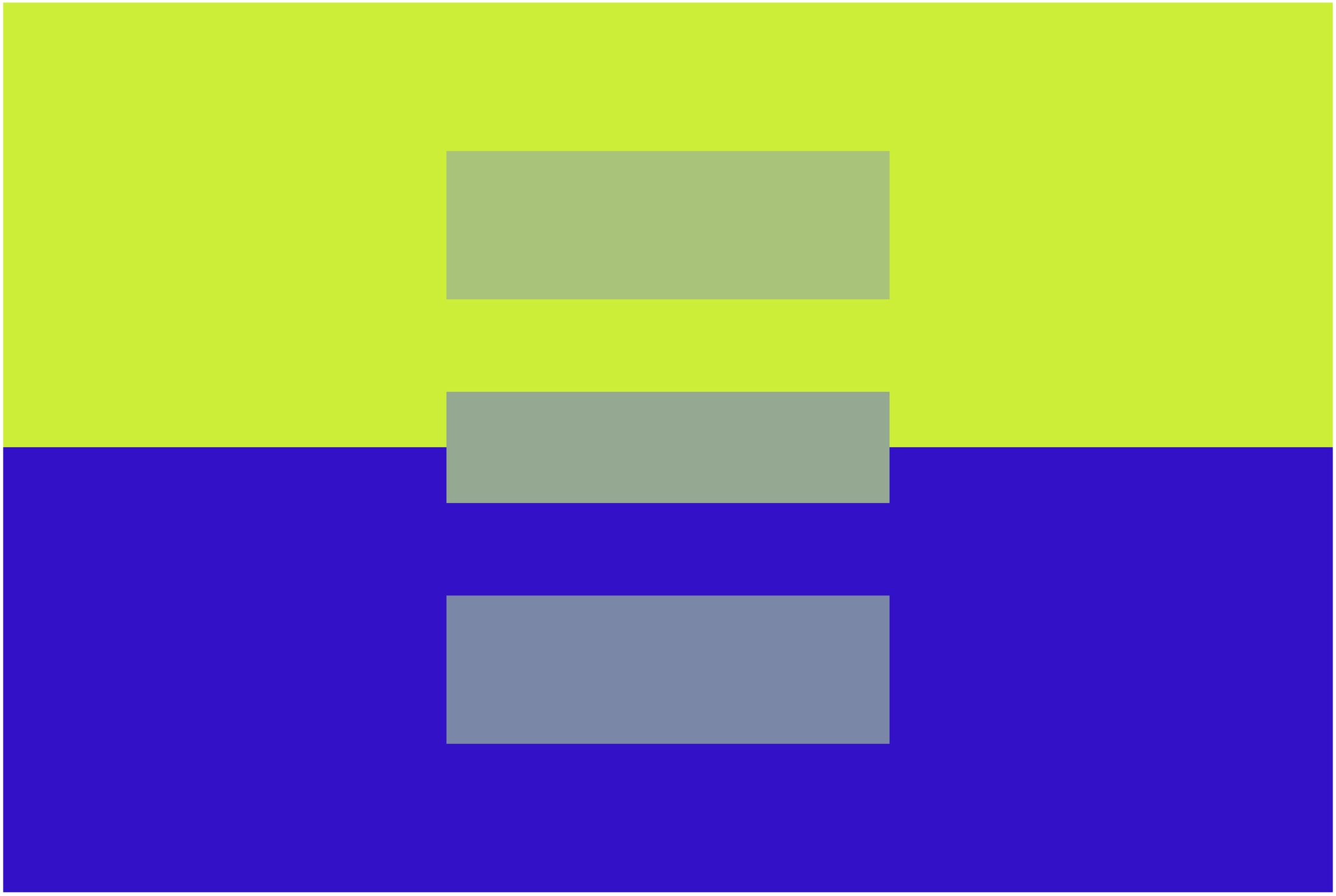
0xFFD61FF8 0xFF29E007 0xFFB181CB 0xFF9A9FAF 0xFF7EB88D



0xFF37E5FC 0xFFC81A03 0xFF7ABCCE 0xFF92A2B2 0xFFA6838F



0xFFCCEE38 0xFF3311C7 0xFFA9C37A 0xFF94A892 0xFF7B87A6



```
// 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?

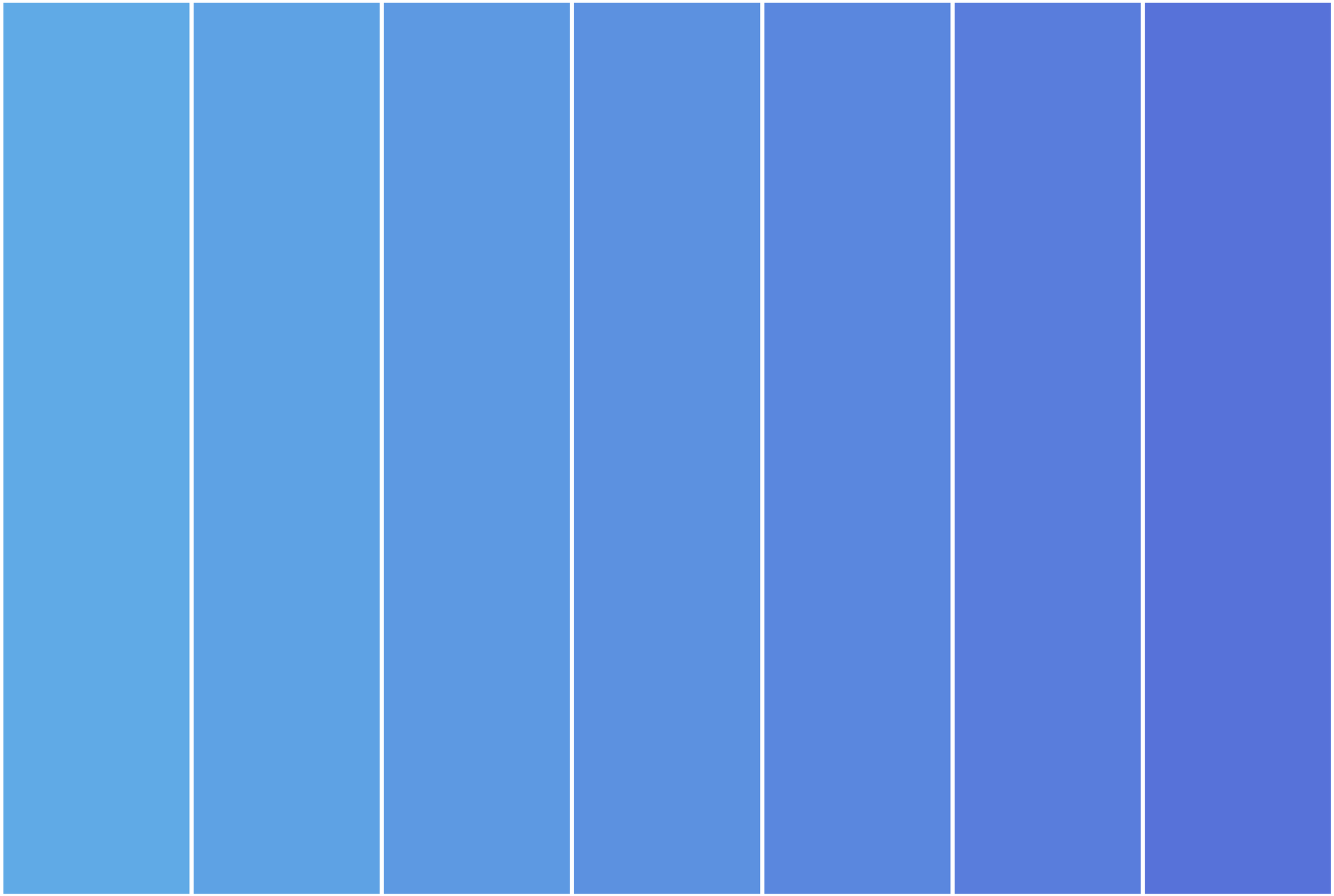
When were colors ever 'a valid reason'?

0xFF5E5644	0xFF7A4E3E	0xFF914639	0xFFA53D32	0xFFB7322B	0xFFC72422	0xFFD70A16

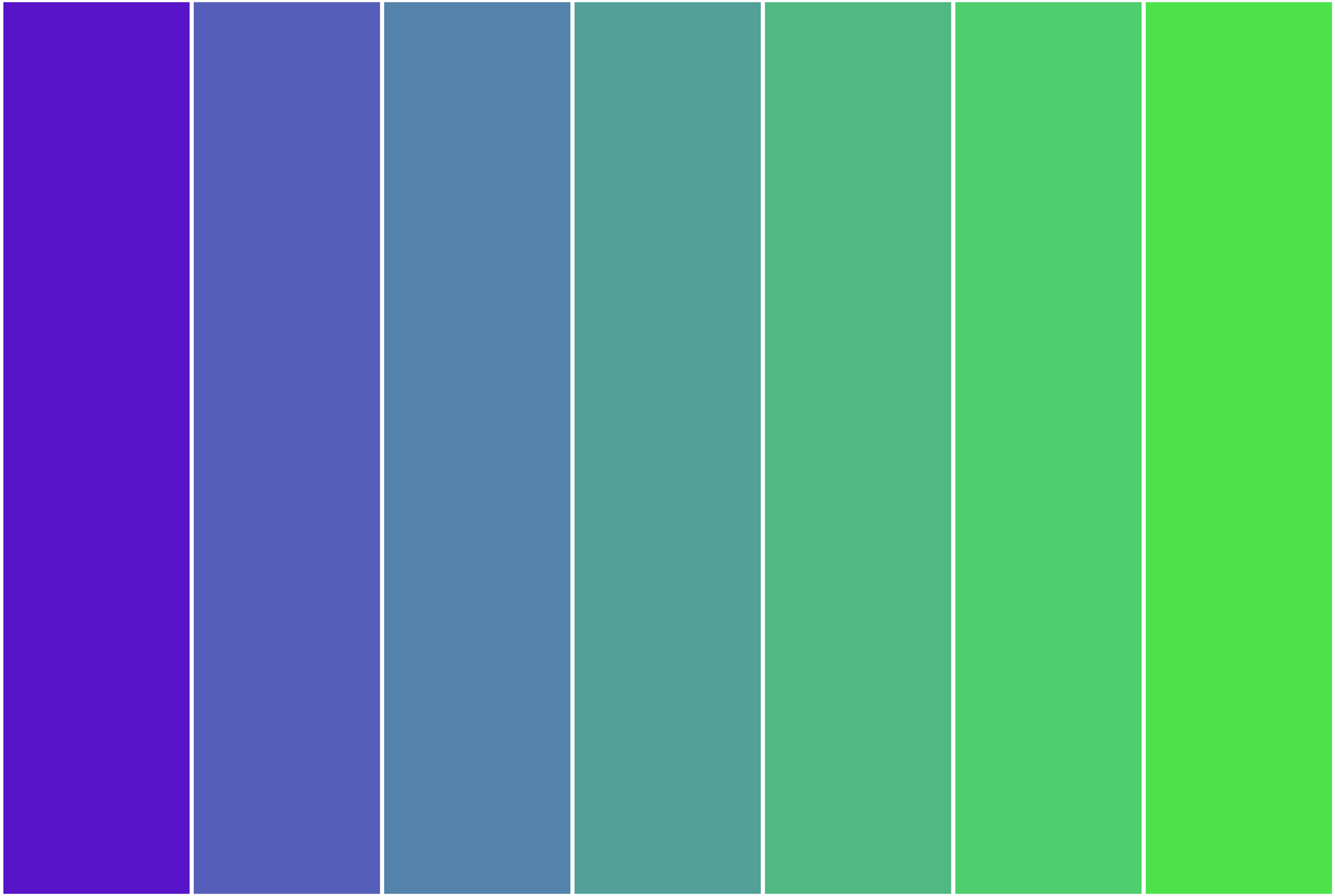
0xFF3527B4	0xFF313FA7	0xFF2E509A	0xFF2A5E8C	0xFF266A7C	0xFF21756A	0xFF1B8054

0xFFB479D2	0xFFAA71D2	0xFF9F6AD2	0xFF9461D3	0xFF8858D3	0xFF7B4ED3	0xFF6C42D3

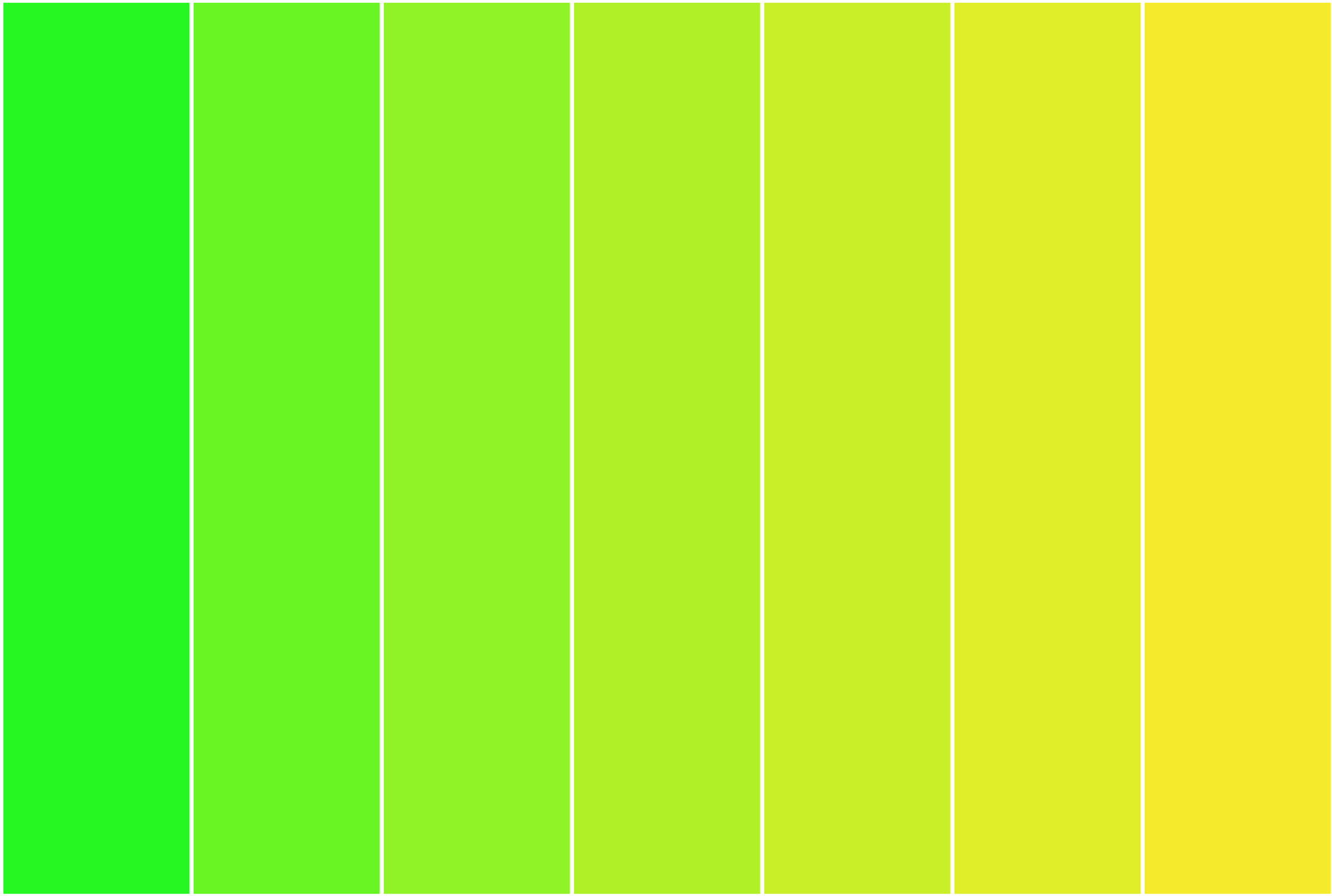
0xFF60AAE6 0xFF5EA2E4 0xFF5D99E2 0xFF5C91E0 0xFF5A87DE 0xFF597DDC 0xFF5772D9



0xFF5714C9 0xFF555EBA 0xFF5383AA 0xFF52A098 0xFF50B883 0xFF4ECE6C 0xFF4DE24C



0xFF26F722 0xFF69F523 0xFF90F325 0xFFAFF126 0xFFC9EF28 0xFFE0ED29 0xFFF5EA2B



```
// 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();
```

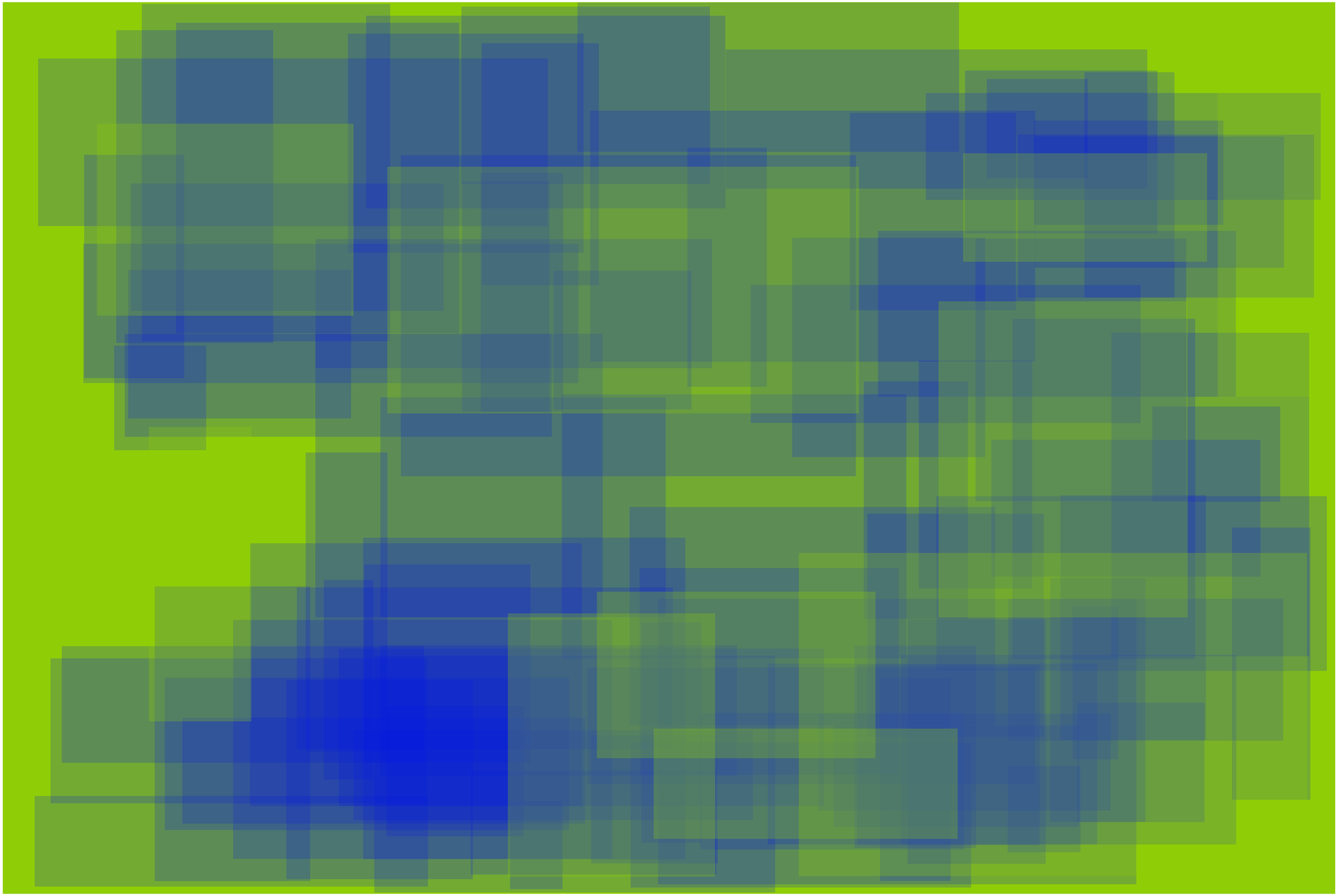
Chapter 06: Bridging Colors

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.

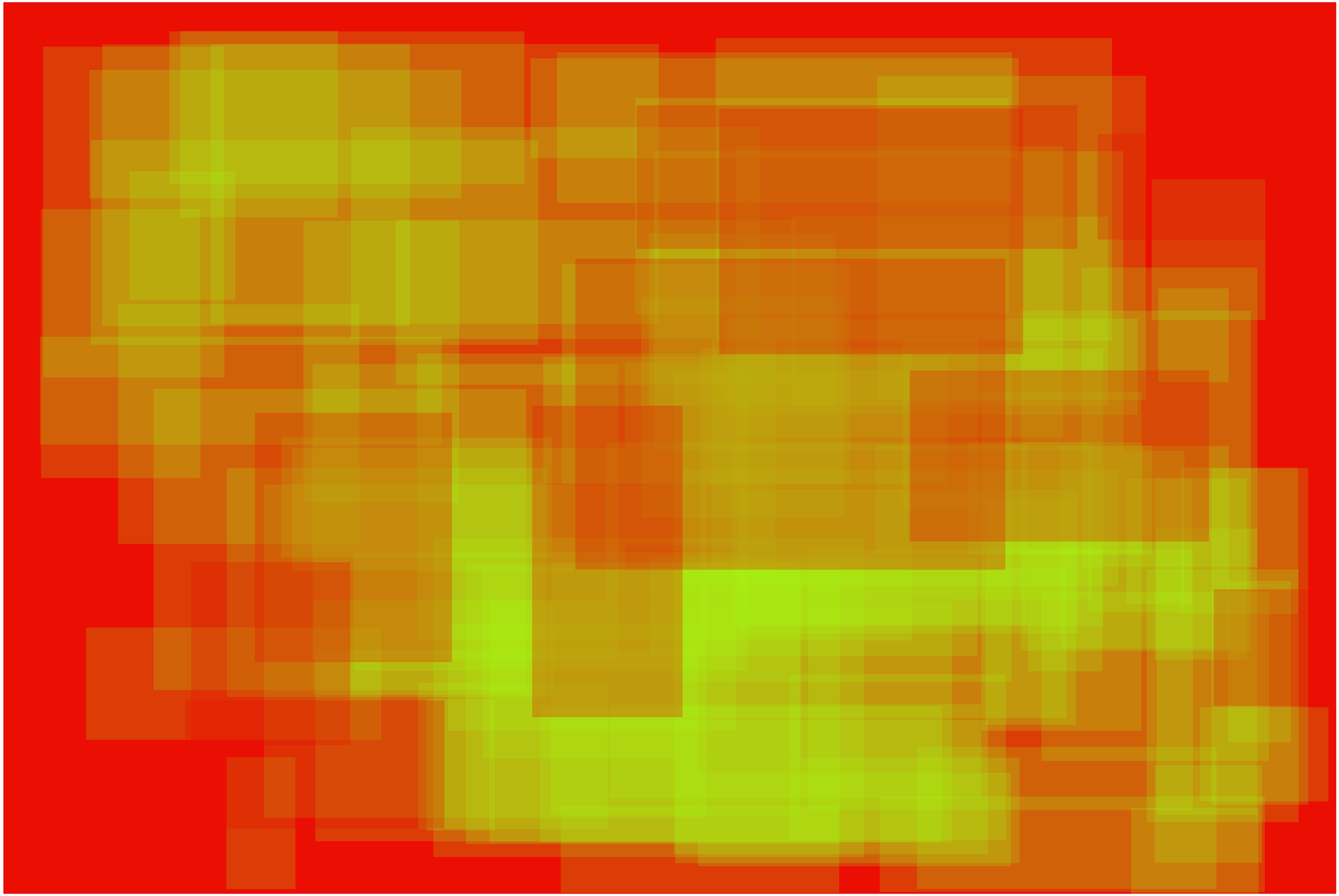
"So yeah, I think in general my alcohol tolerance is pretty good," I began. A few moments later: "I'm getting kinda sleepy."

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

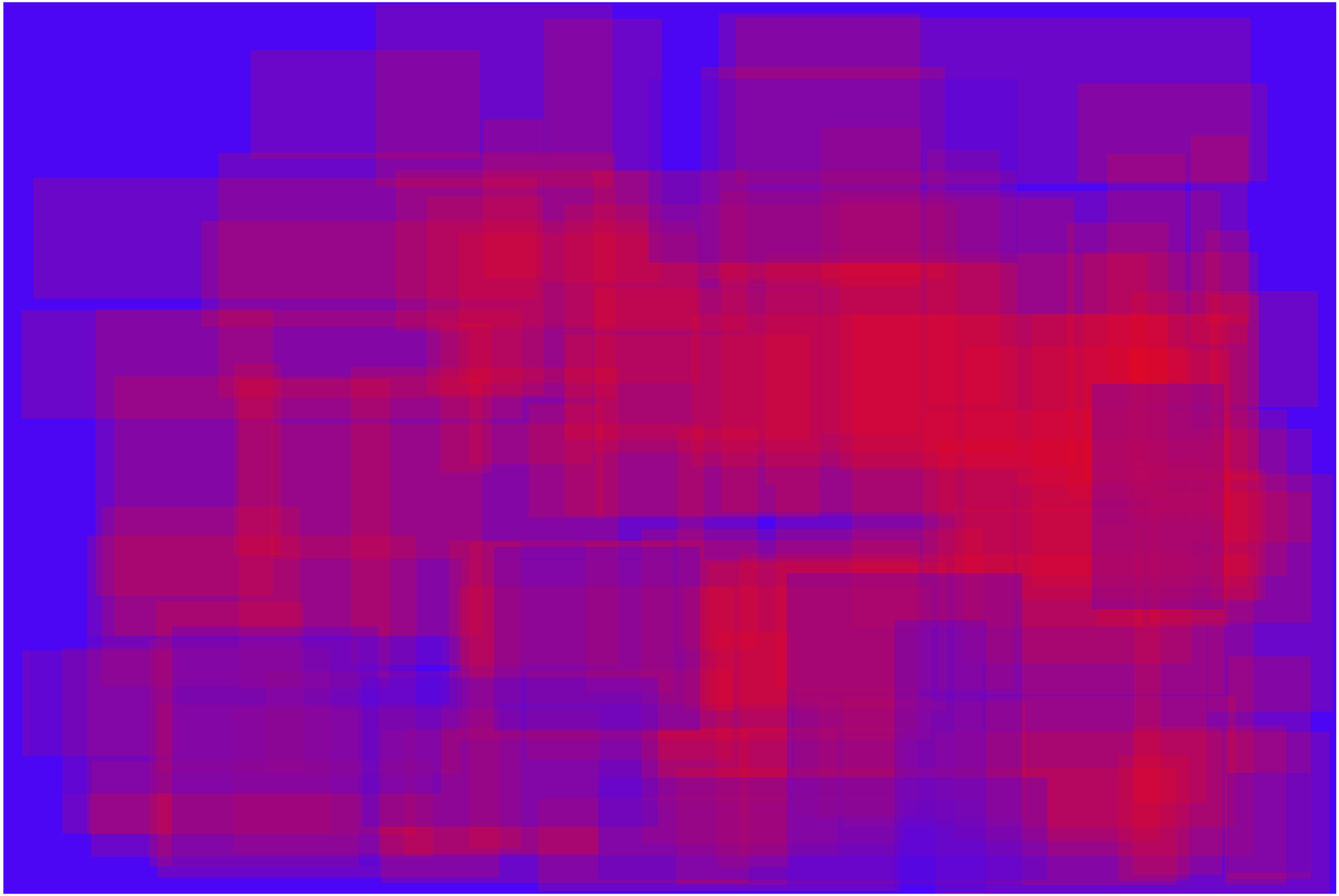
0xFF0518E5 0xFF8FCE07



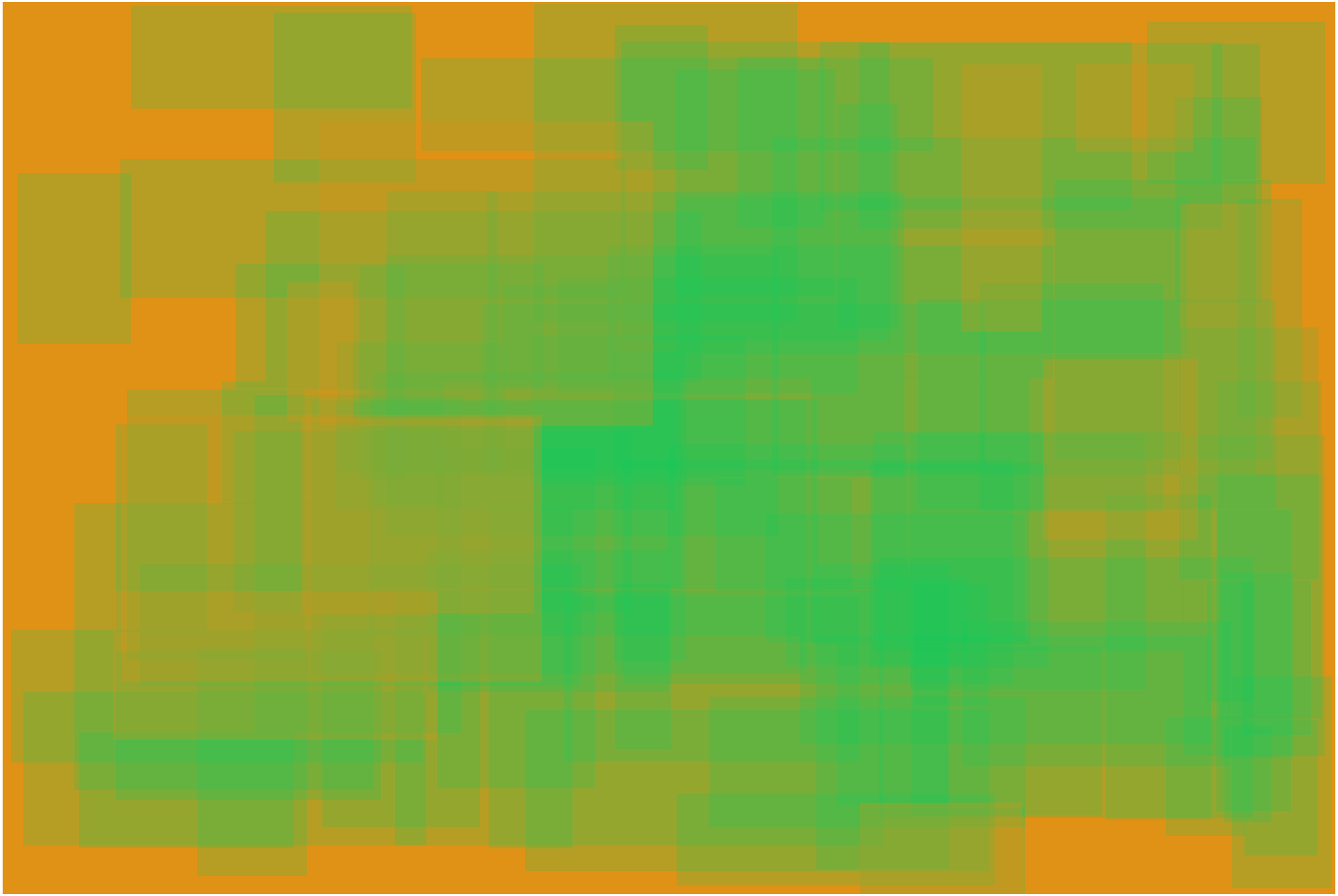
0xFFA7FA17 0xFFEA0E03



0xFFED0C18 0xFF4C06F3



0xFF0ECD61 0xFFE09216



```
// 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???'

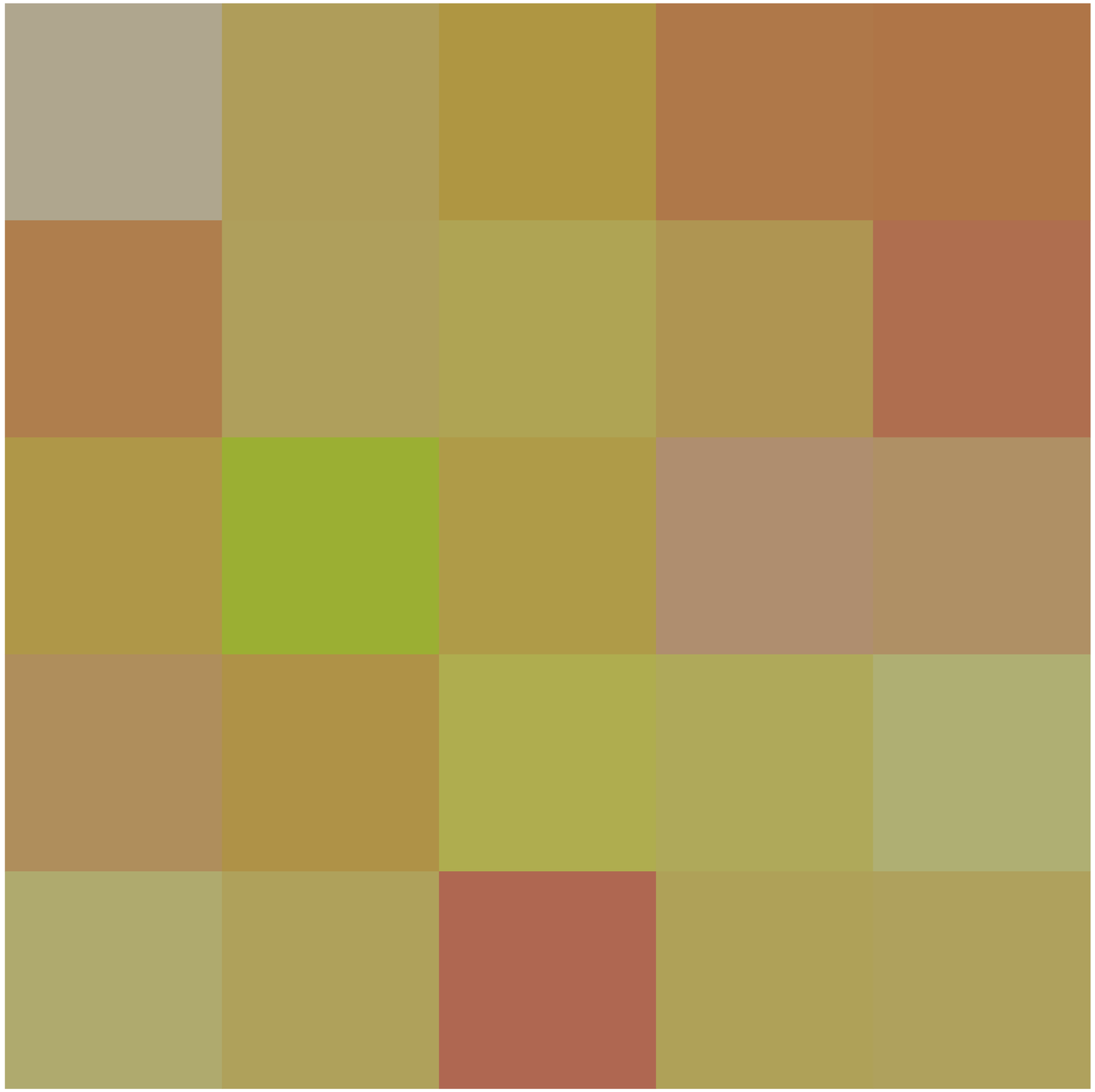
You may be debating whether you should just skip this useless expository text or read on.

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.

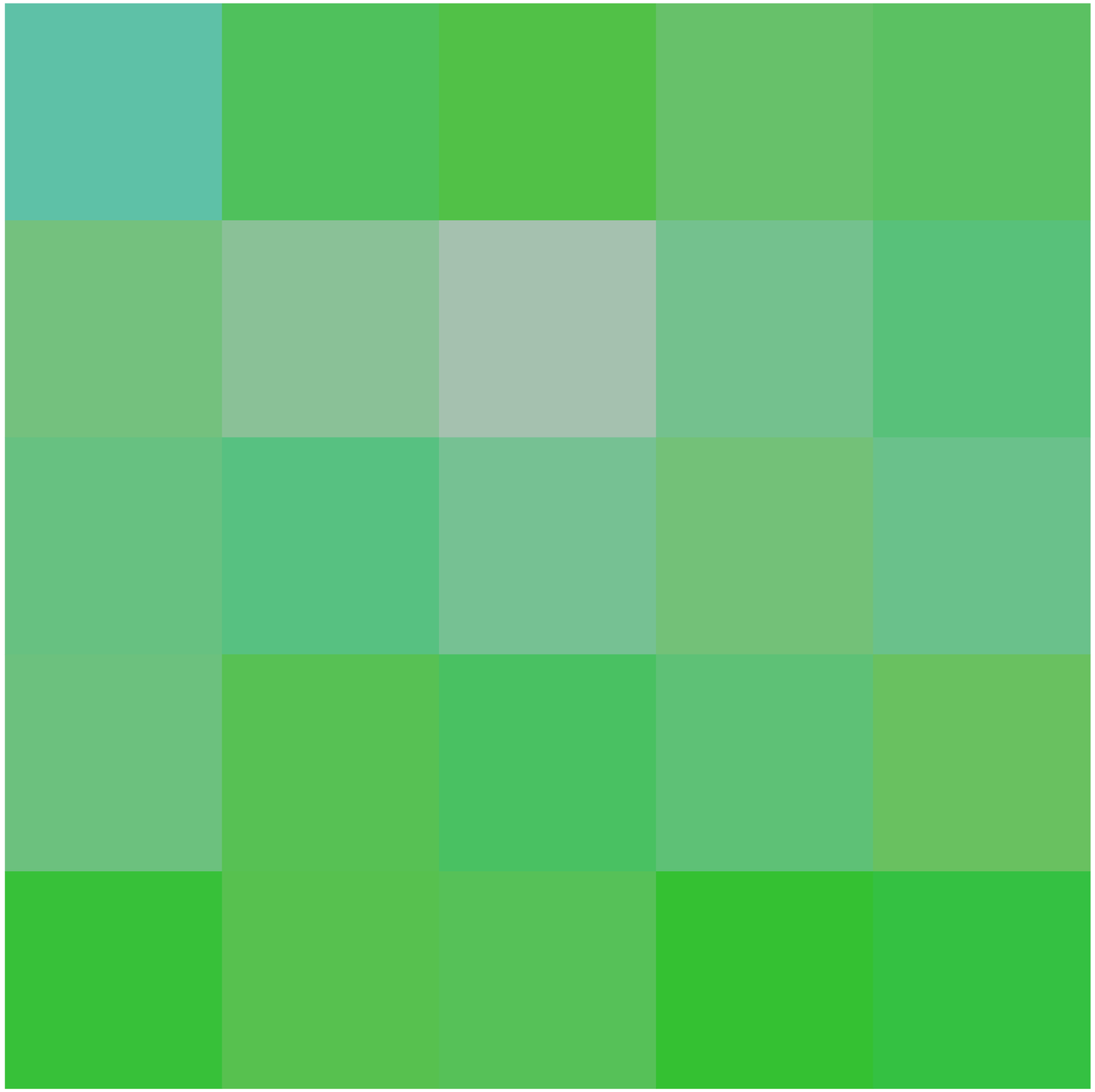
You may be wondering what that previous sentence has to do with anything.

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

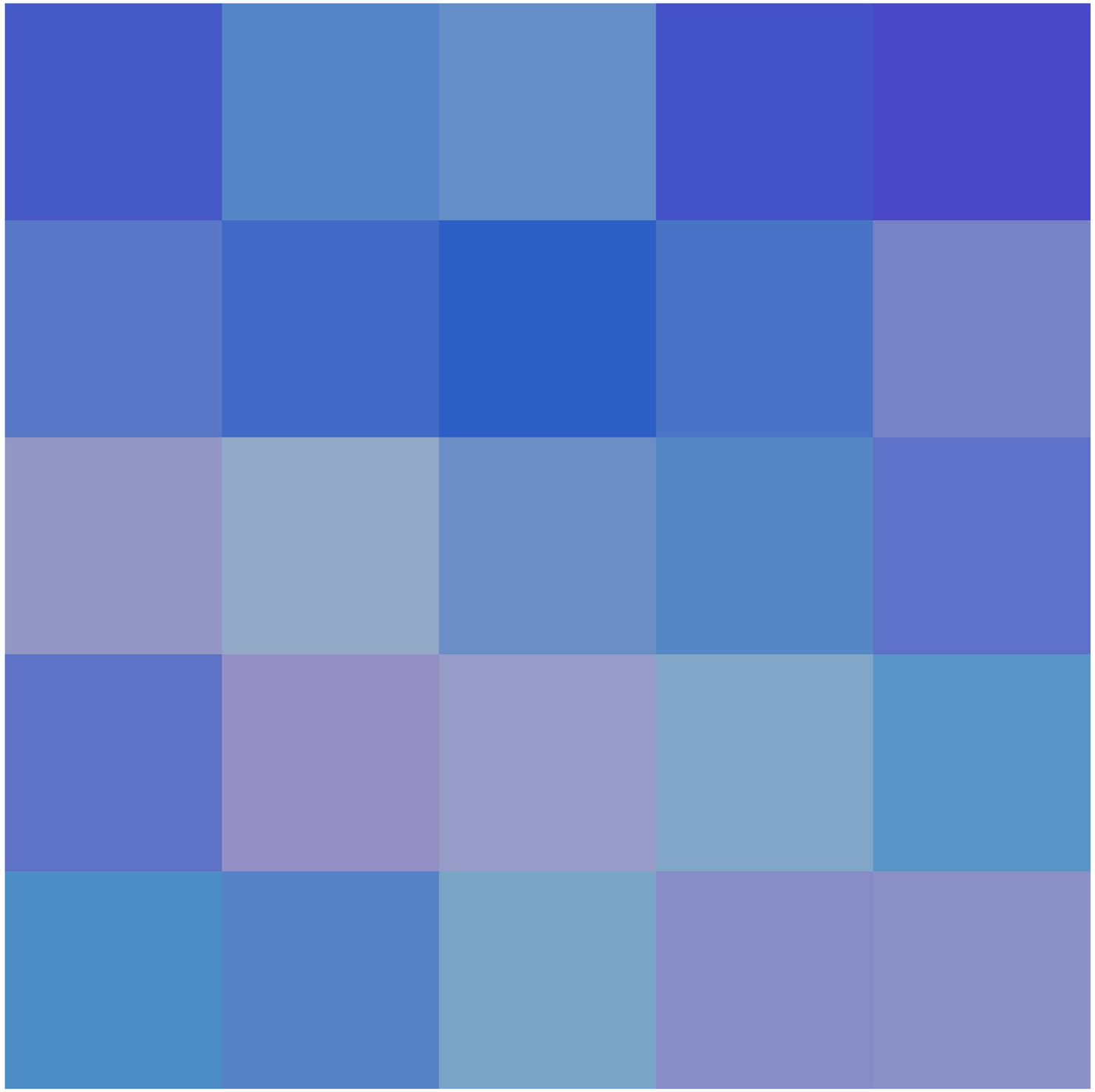
value: 69



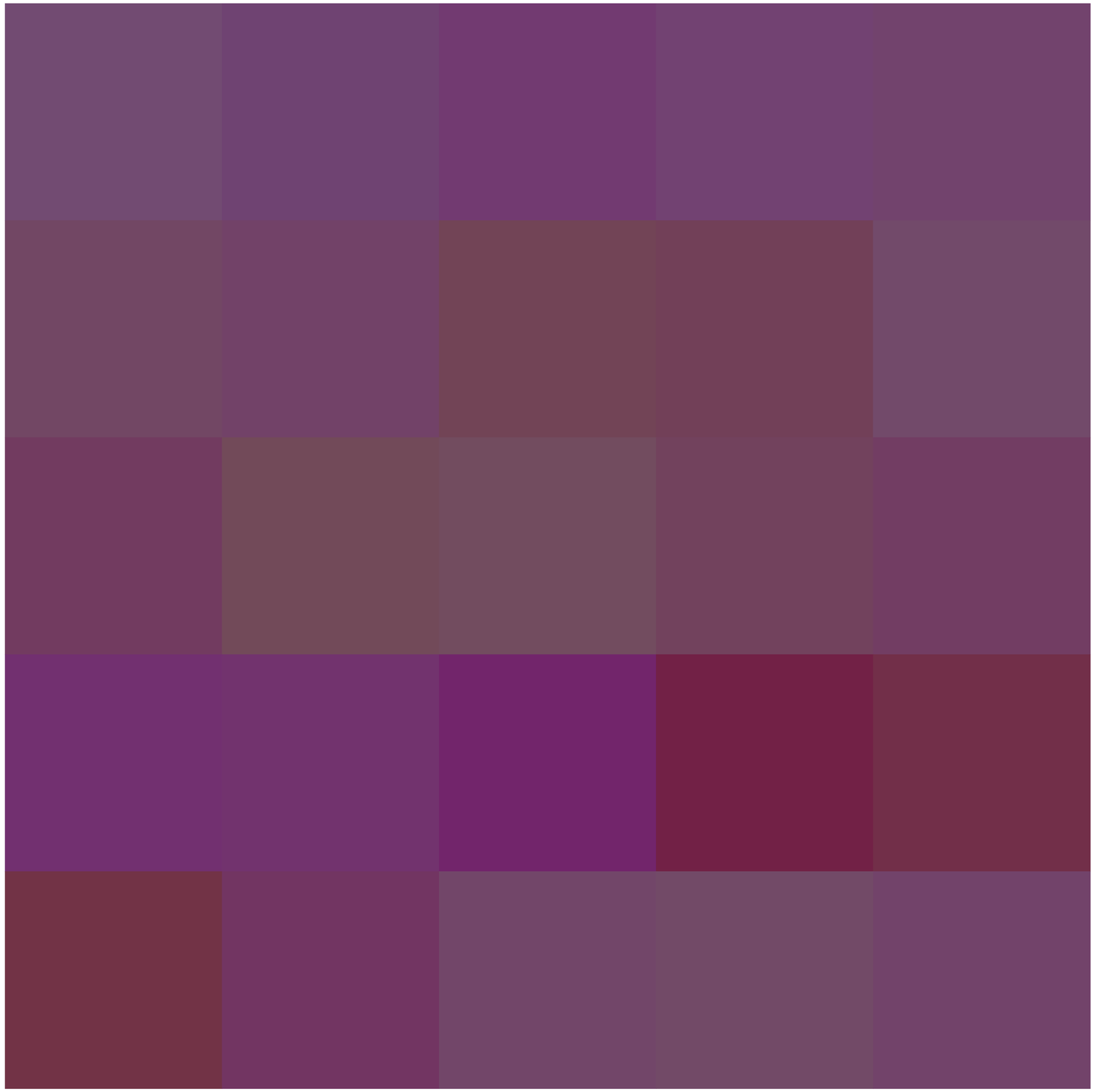
value: 76



value: 78



value: 45



```
// 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:
Tatsuyuki Tanaka

0xFFAF9888

0xFFD4C0B7

0xFF8B7667

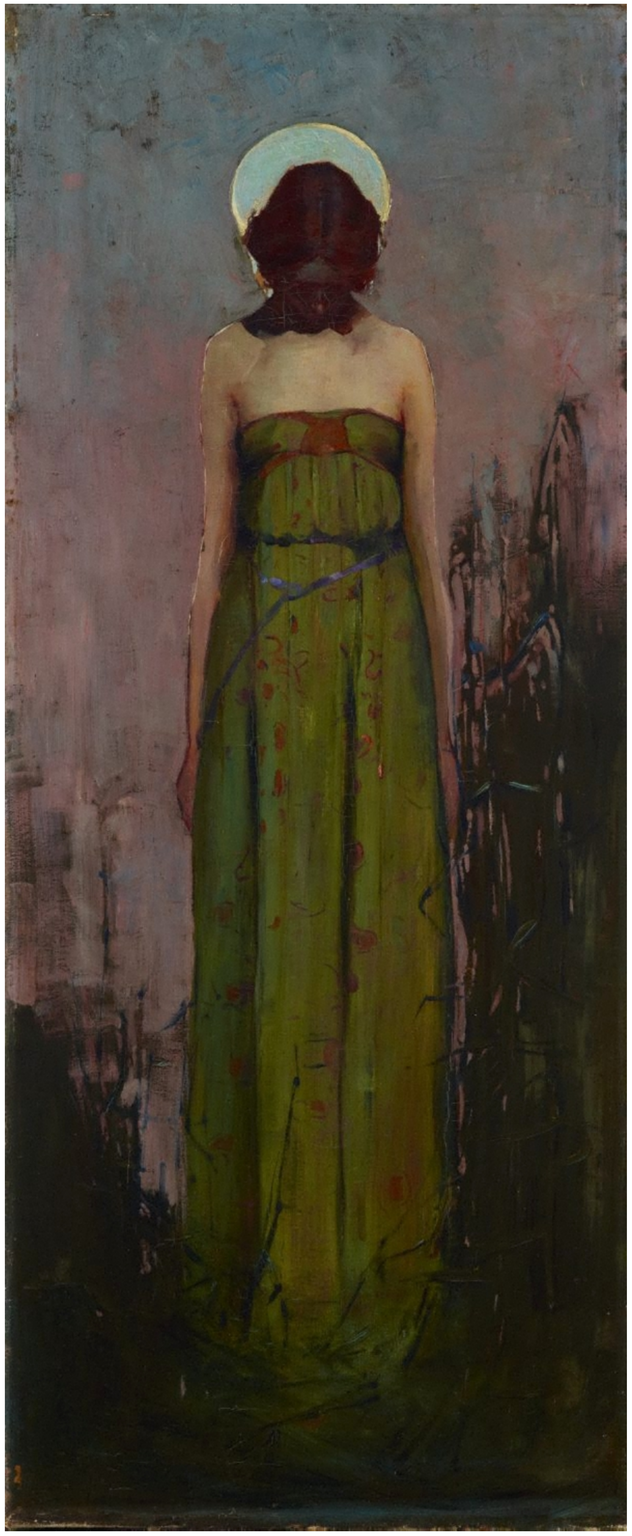
0xFF624E41

0xFF423126

0xFF211811

0xFF0F0E08

artist:
Sydney Long



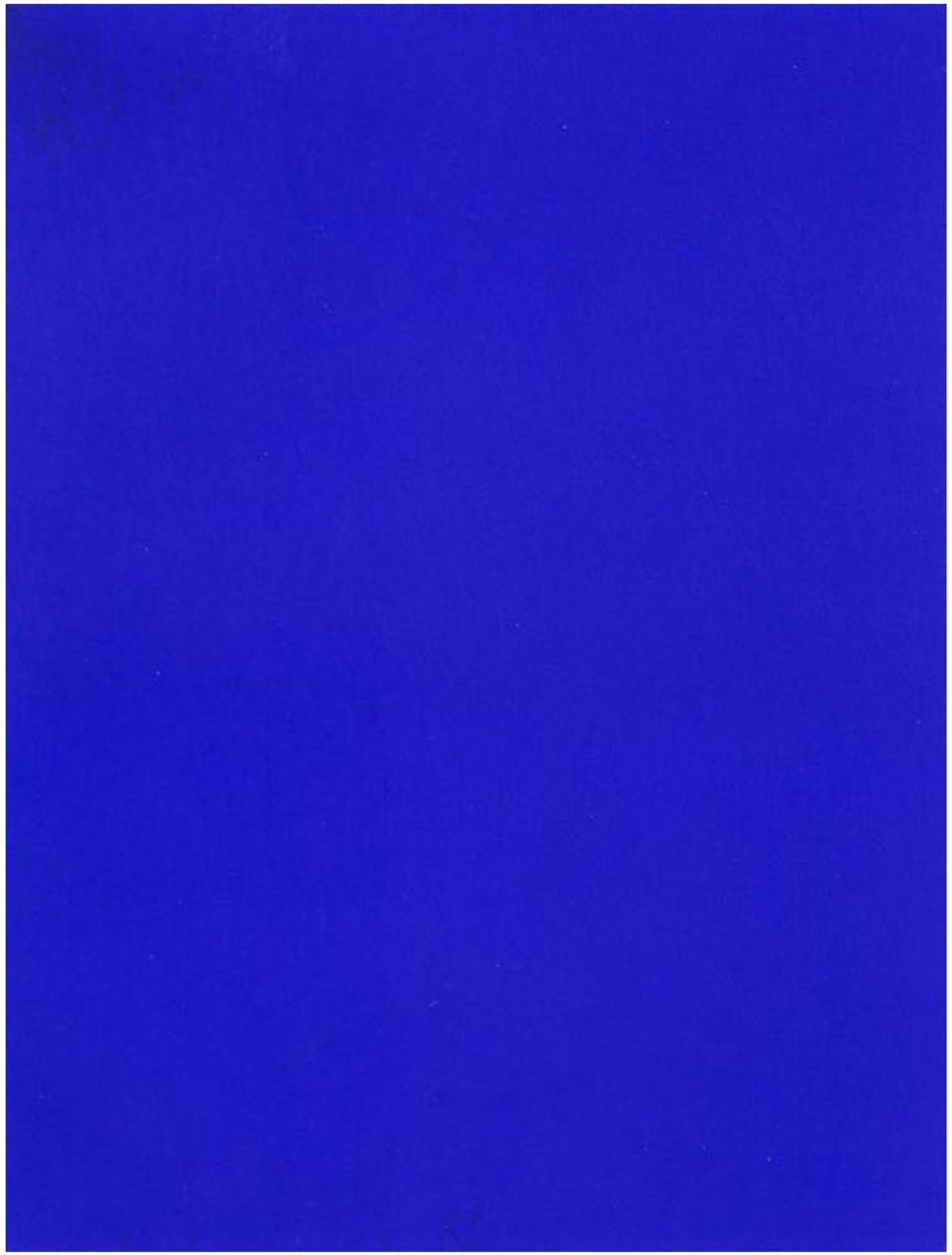
0xFF69605B

0xFF514E44

0xFF96876D

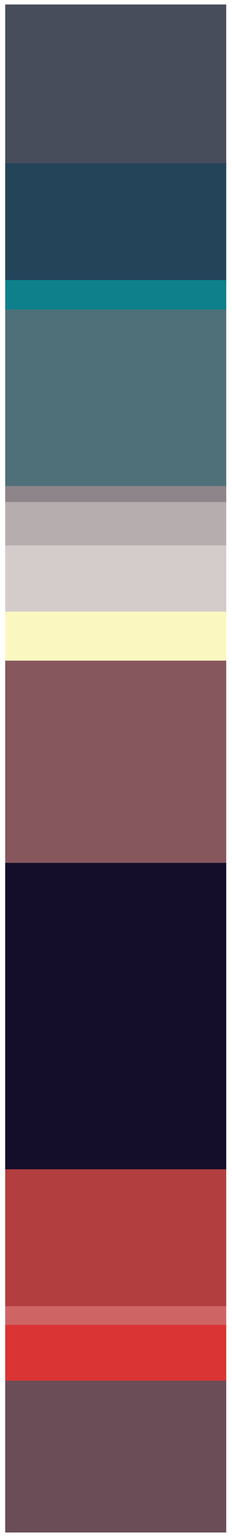
0xFF2F211A

artist:
Yves Klein

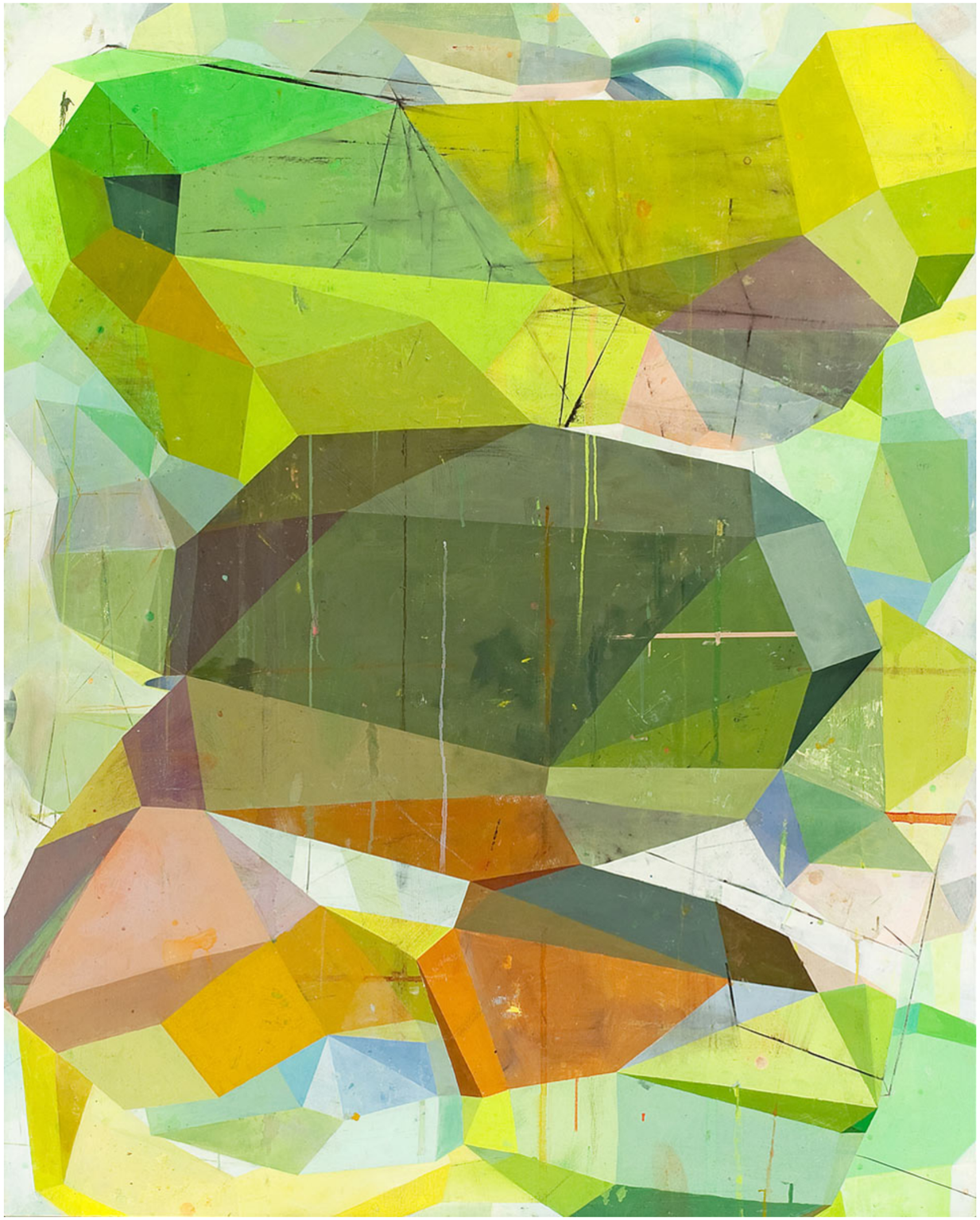


0xFF201BC4

artist:
James Jean



artist:
Deborah Zlotsky

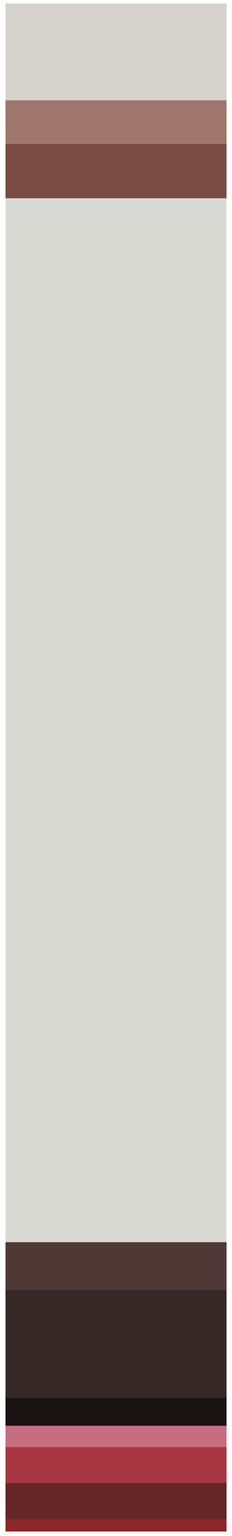


artist:
Gustav Klimt

artist:
Piotr Jablonski

0xFF866520
0xFFB08F0C
0xFFBE9F03
0xFFF5EF07
0xFF4D3523
0xFF332113
0xFF0E0A06
0xFFA38D68
0xFF255C52
0xFFFDF00E

artist:
Cy Twombly



0xFFD5D2CC

0xFF9E766B

0xFF784B45

0xFFD9DAD4

0xFF4F3936

0xFF352826

0xFF1A1313

0xFFC86C7F

0xFFA83640

0xFF662527

0xFF8F2627

artist:
Paul Klee

```
ArrayList<Pixel> colors = new ArrayList<Pixel>();
float tolerance = 50.0;
int totalCount = 0;
float proptolerance = 0.005;
PImage[] images = new PImage[folderimages];
String[] names = new String[folderimages];

// recipe for palette grabber
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 = new ArrayList();
  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) {
  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 i = 0; i < iheight; i++) {
    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;
          }
          else if(!foundMatch) {
            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);
float ypos = 0f;

pushMatrix();
// place image a little to the right of your margins
translate(margin + 700, margin);
image(sourceimg, 0,0);
// place palette beside image
```

```
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);
  rect(palettepos, ypos, 300, rheight);

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

```
    }
  }

  popMatrix();

}
```

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