THE COLORIST'S COOKBOOK

"WOMEN THINK OF ALL COLORS EXCEPT THE ABSENCE OF COLOR. I HAVE SAID THAT BLACK HAS IT ALL. WHITE TOO. THEIR BEAUTY IS ABSOLUTE. IT IS THE PERFECT HARMONY."

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.

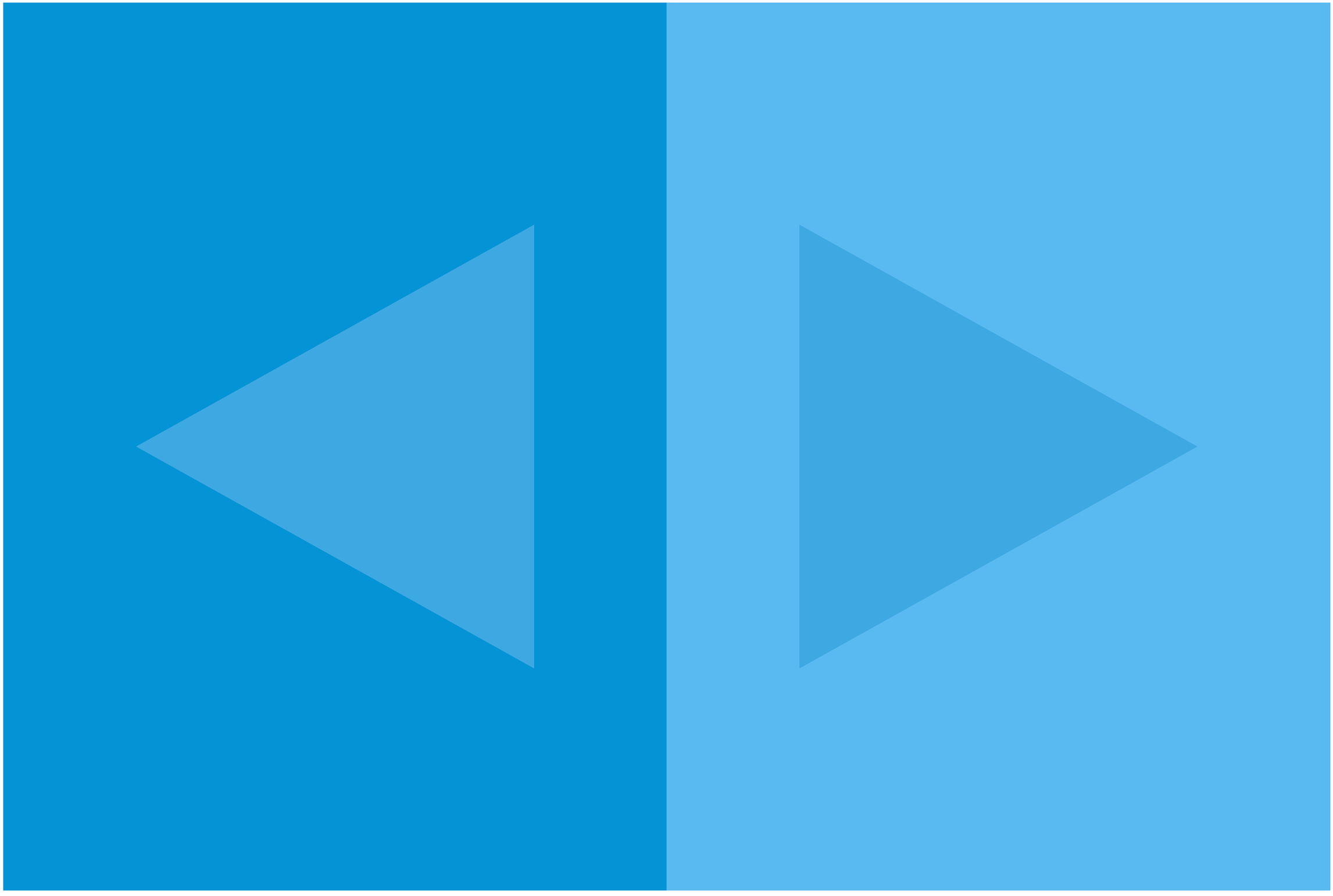
CHAPTER 01: 3 COLORS INTO 4 PART I

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.*

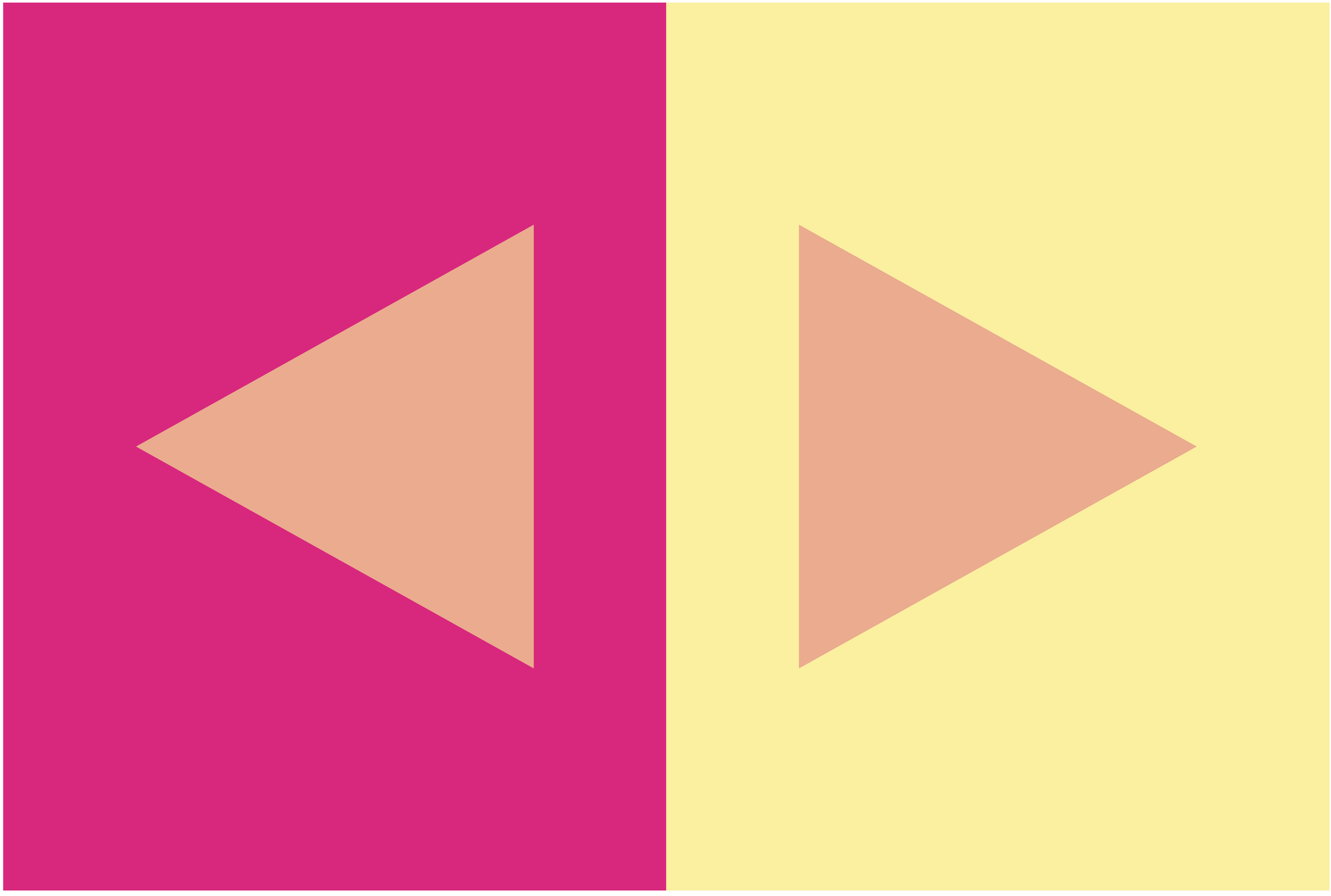
According to color theory, having a color in the background removes that color from the foreground--this is why the dress shown against yellow lighting would make it look less yellow and more blue. I won't get into the specifics of the science behind this because I'm running out of space on this page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

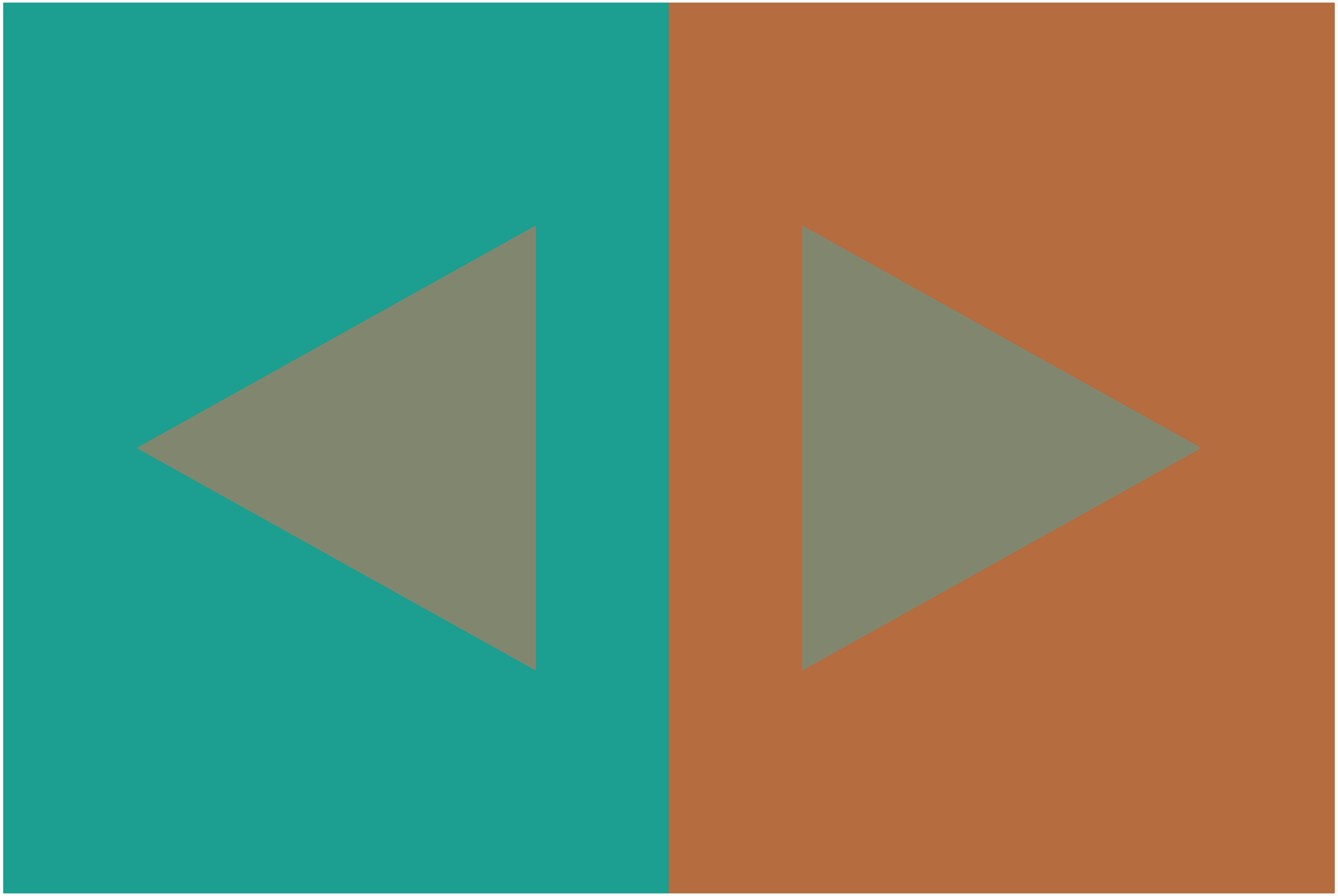
0xFF0494D5 0xFF59BAF1 0xFF3EA8E3



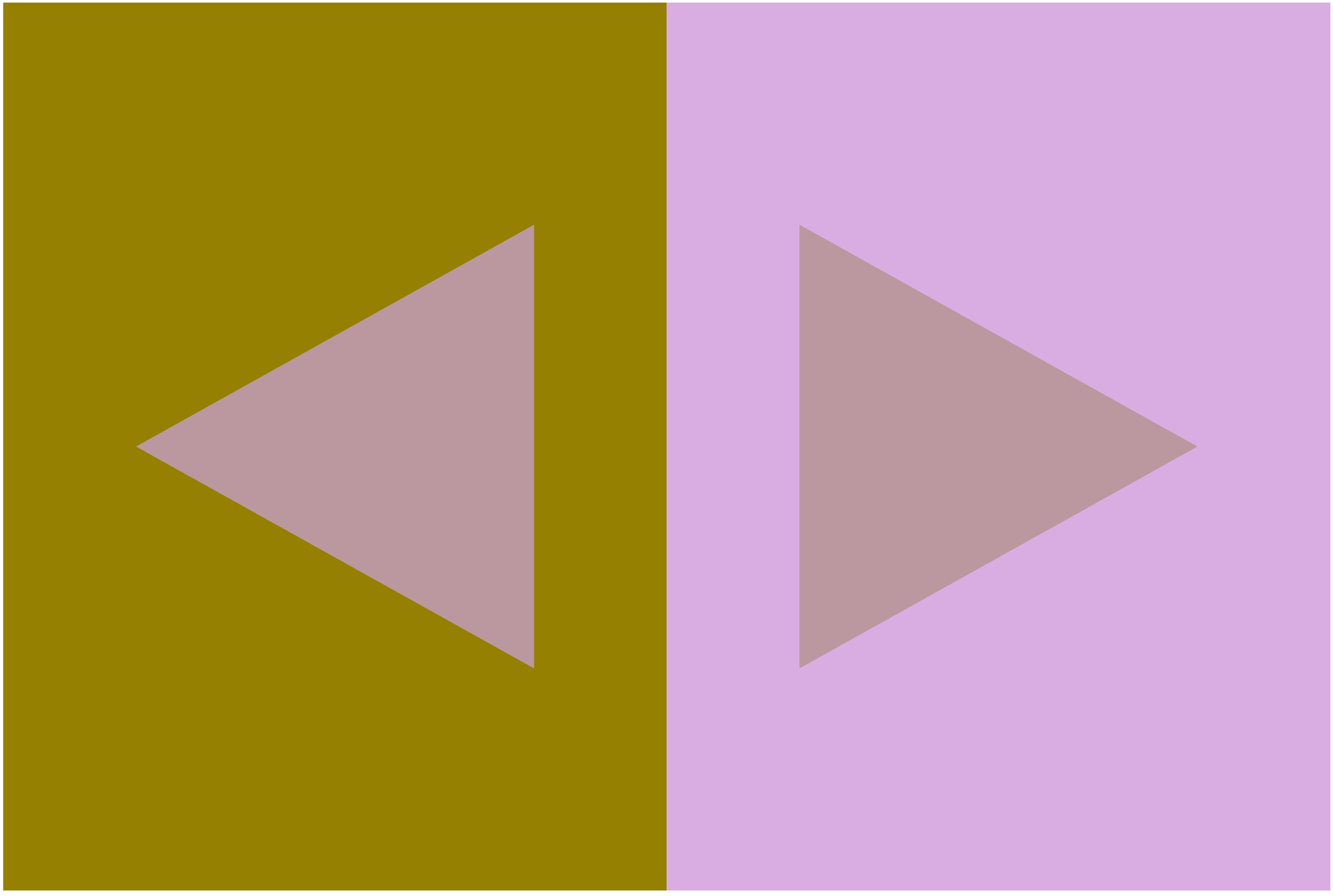
0xFFD8287D 0xFFFBEFA0 0xFFEAAB8F



0xFF1C9E90 0xFFB56C3F 0xFF81876F



0xFF968002 0xFFDAADE2 0xFFBB989F



```
// 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

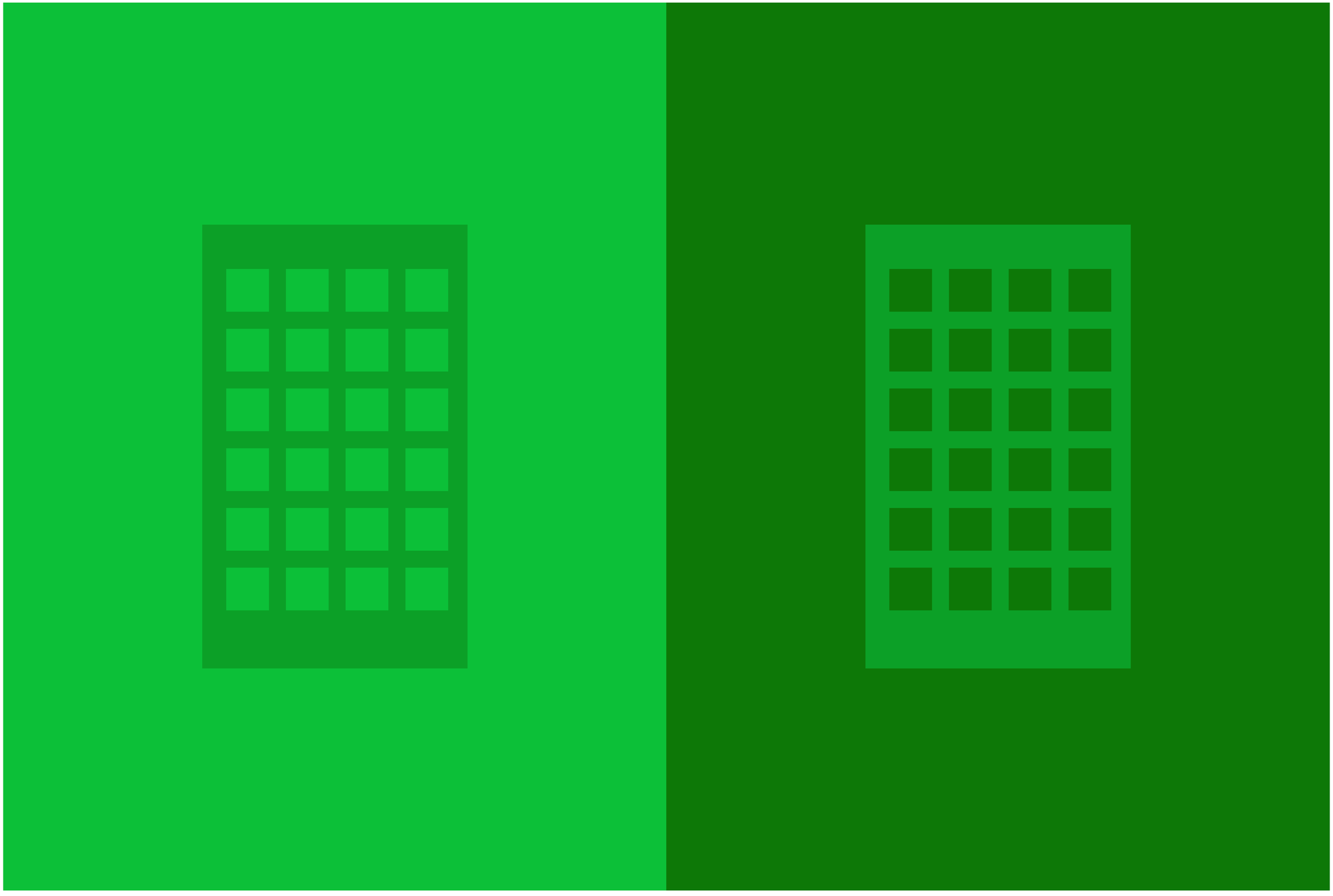
Last Tuesday, I had to go to my professor's office hours to prepare for an exam. My friend and I made our way to the back corner of the room and sat down, taking out our notes and note-taking instruments and whatever else people possess when they need to study. As more students shuffled into the small 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."

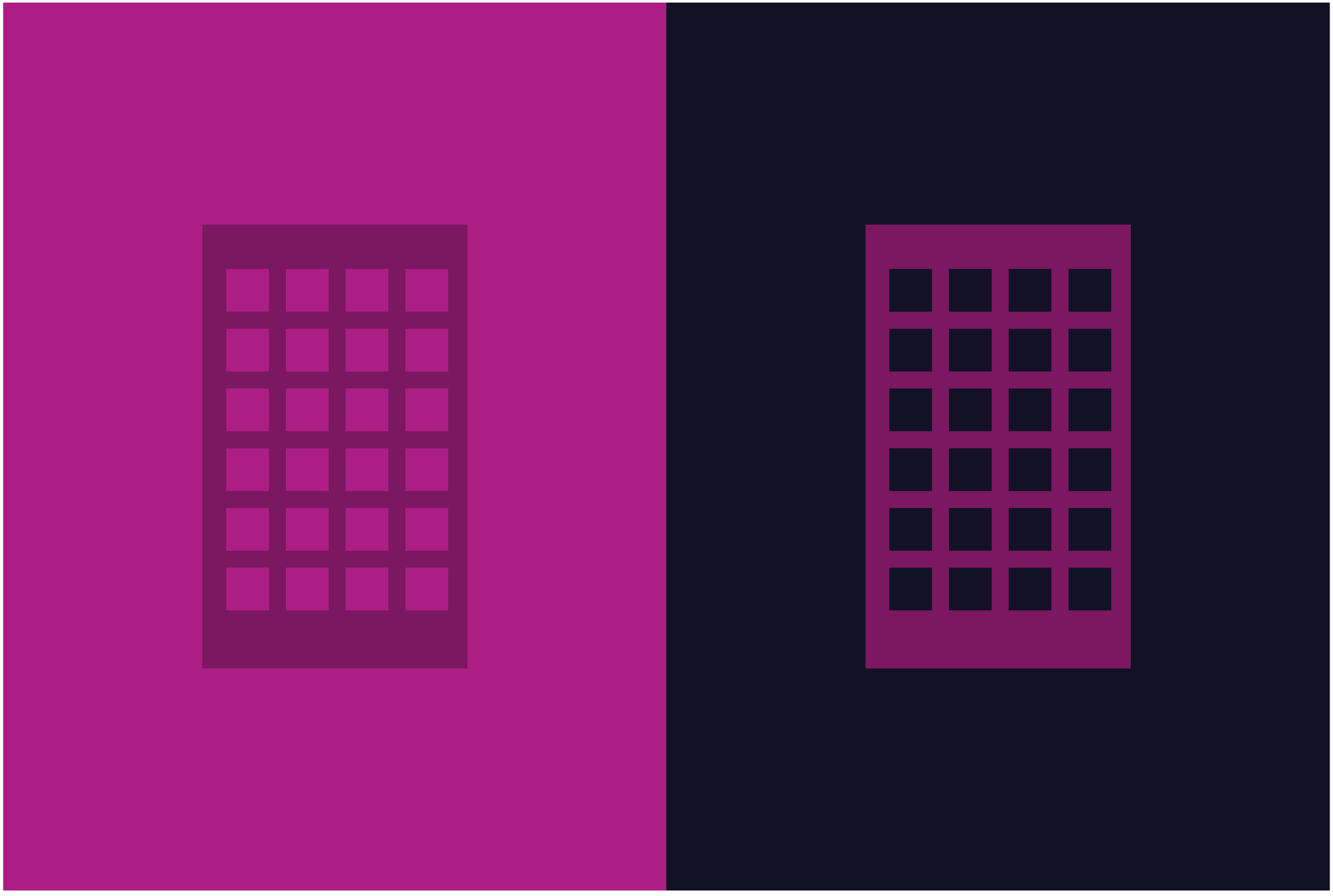
And that's when the room was thrown into a passive-aggressive debate about whether Macs were better than Windows machines (or rather, why Windows machines were better than Macs).

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 the metaphorical daggers thrown at my back. But for whatever reason, on that unremarkable Tuesday, during that unremarkable conversation, it was exactly what I did.

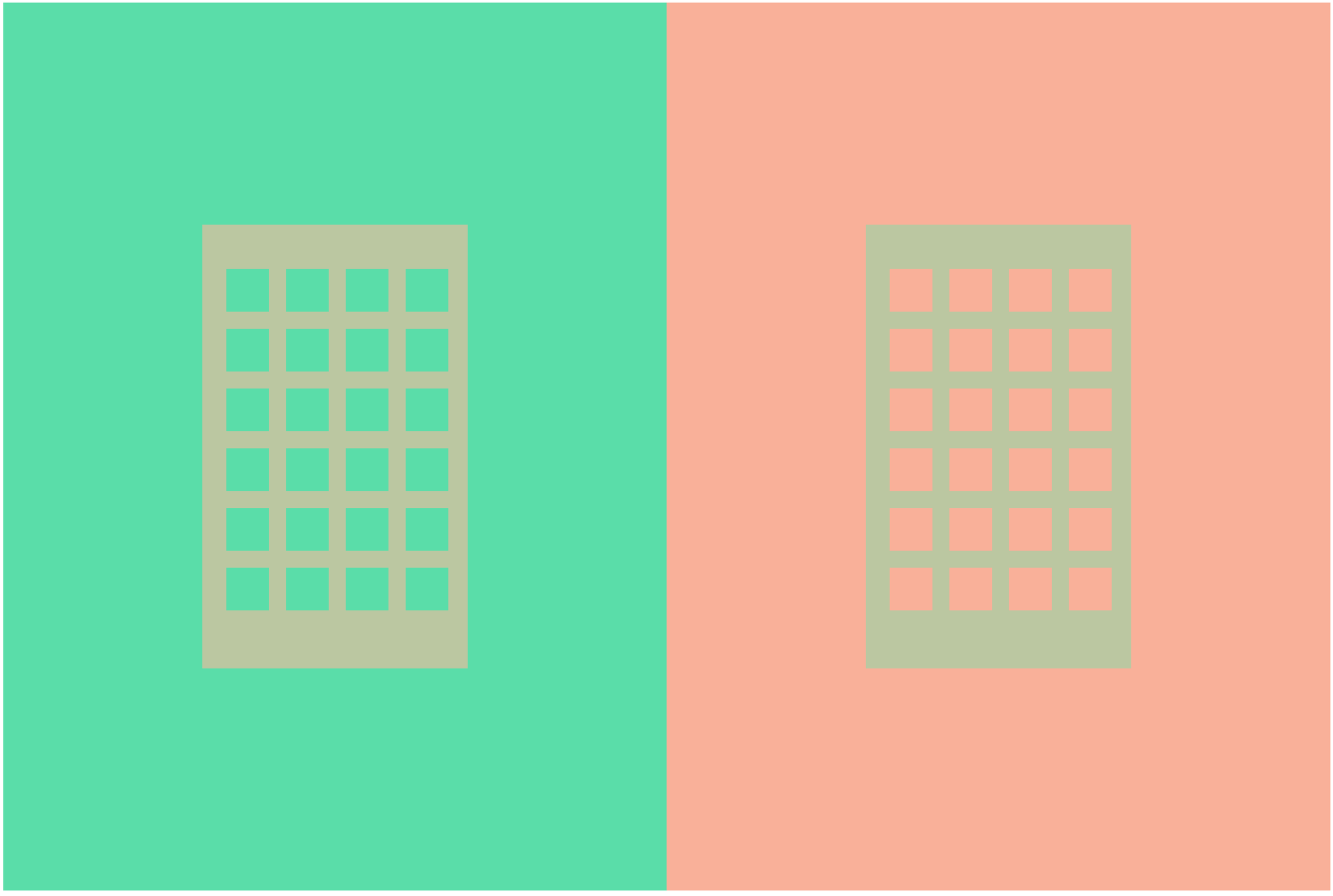
0xFF0CC038 0xFF0D7807 0xFF0CA027



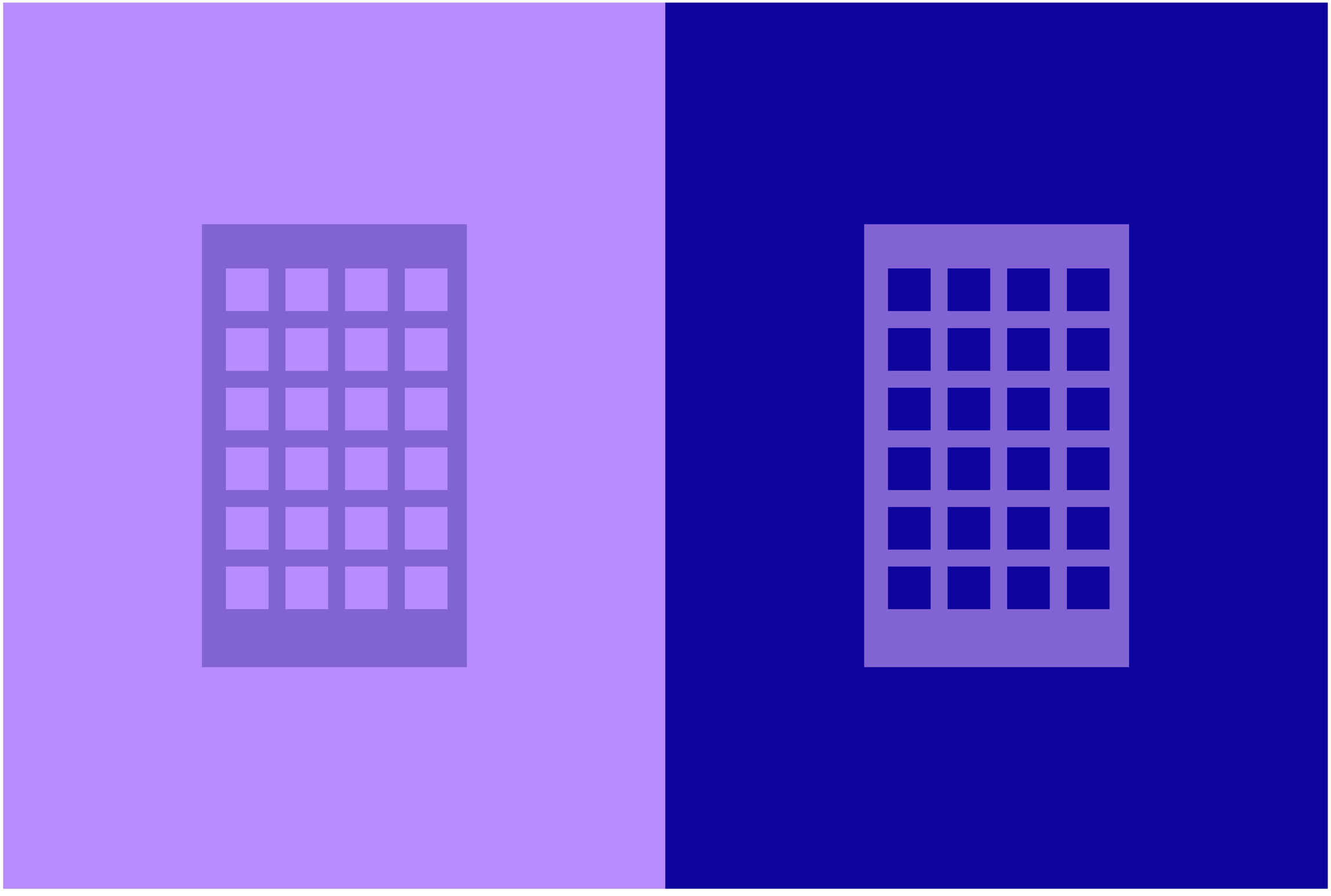
0xFFAD1E85 0xFF131126 0xFF7B1861



0xFF5ADDA9 0xFFF9B099 0xFFBBC7A1



0xFFB68CFE 0xFF0F039E 0xFF8163D3



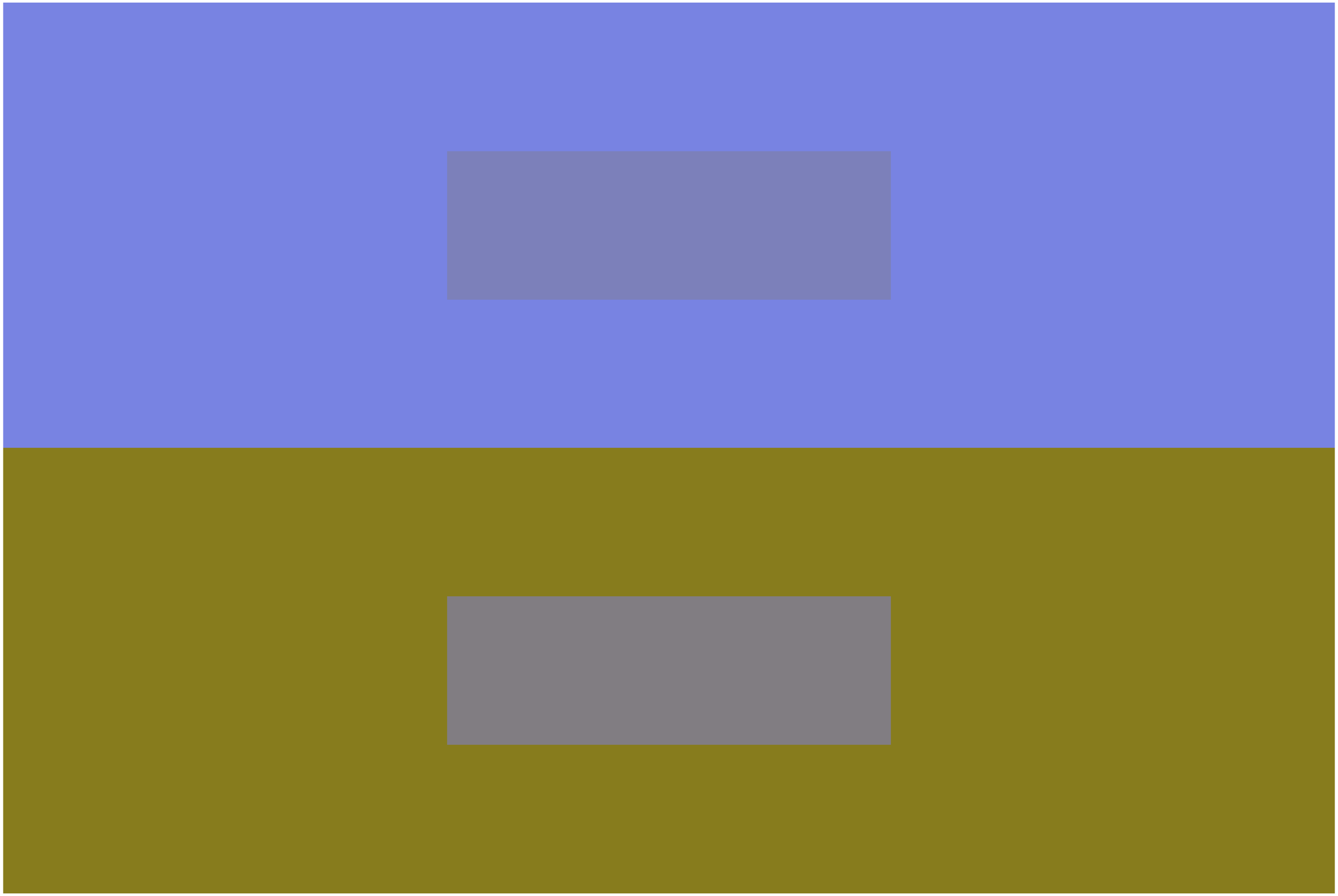
```
// recipe for making 3 colors look like 4...with holes!

// 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);
```

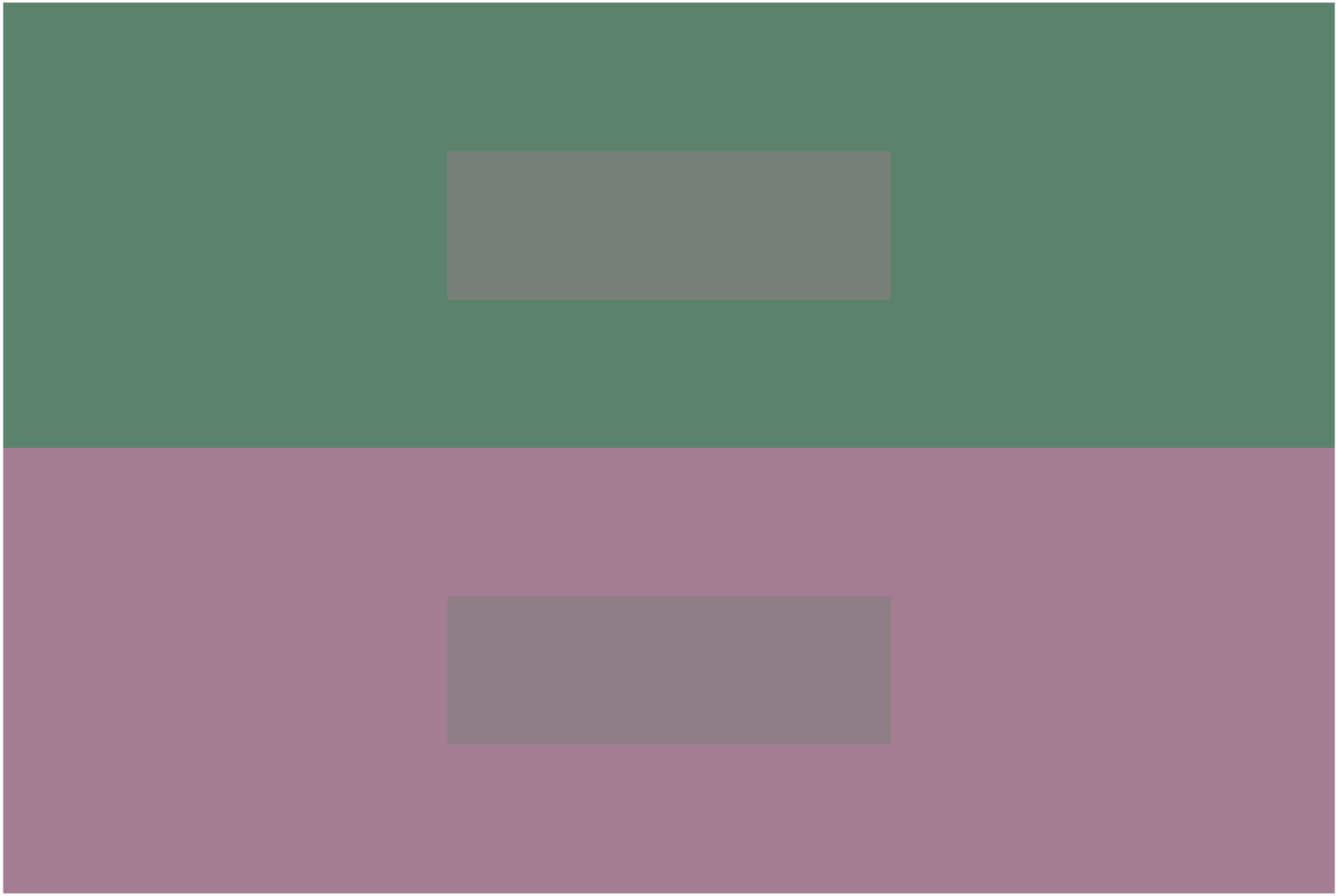
```
pushMatrix();

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

0xFF7883E2 0xFF877C1D 0xFF7C80BA 0xFF817D82



0xFF5B826C 0xFFA47D93 0xFF778079 0xFF907E86



0xFFBFC360 0xFF403C9F 0xFFA0A377 0xFF78798D

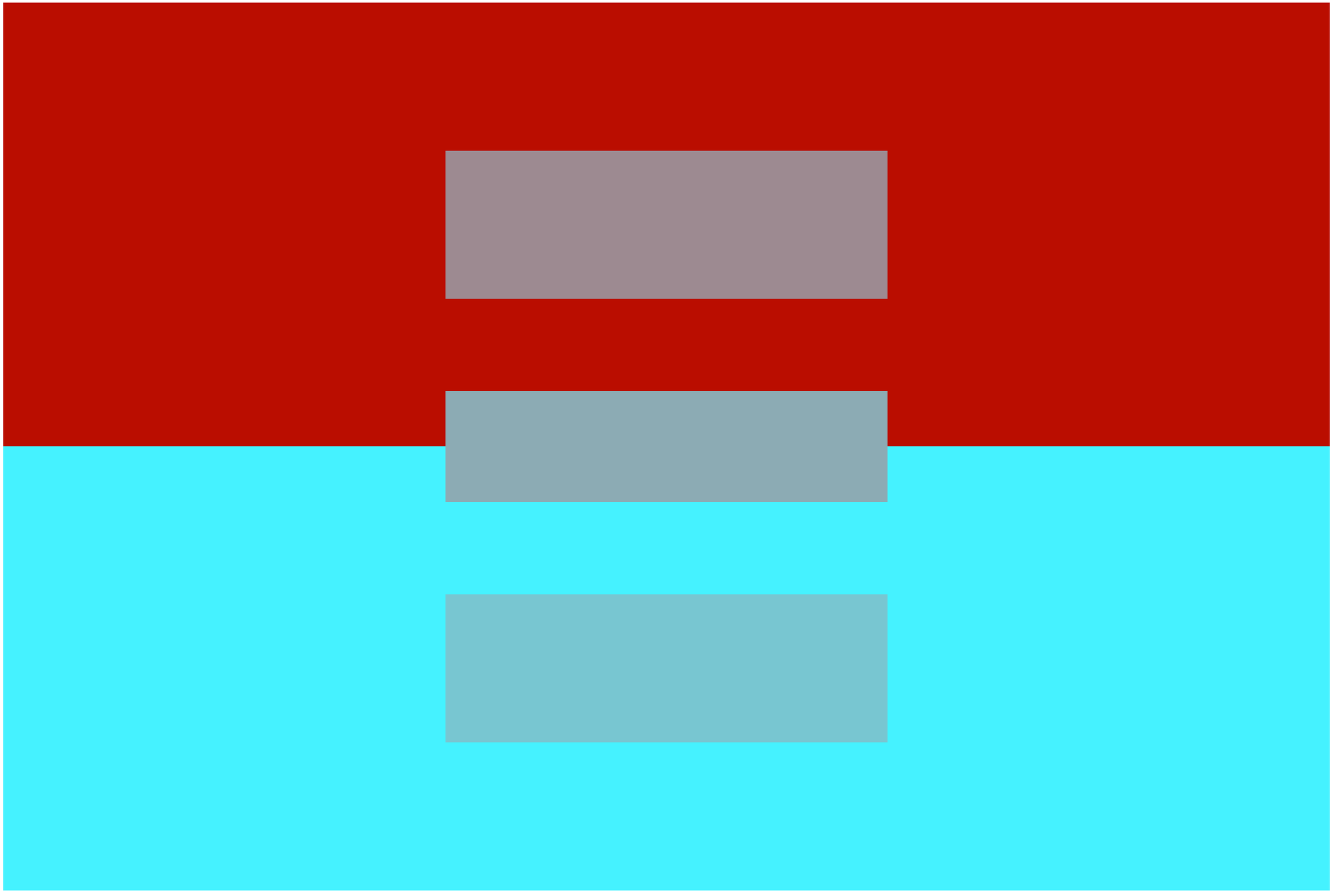
```
// recipe for making 4 colors look like 3
// prepare the first and second colors
SecureRandom random = new SecureRandom();

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();
```

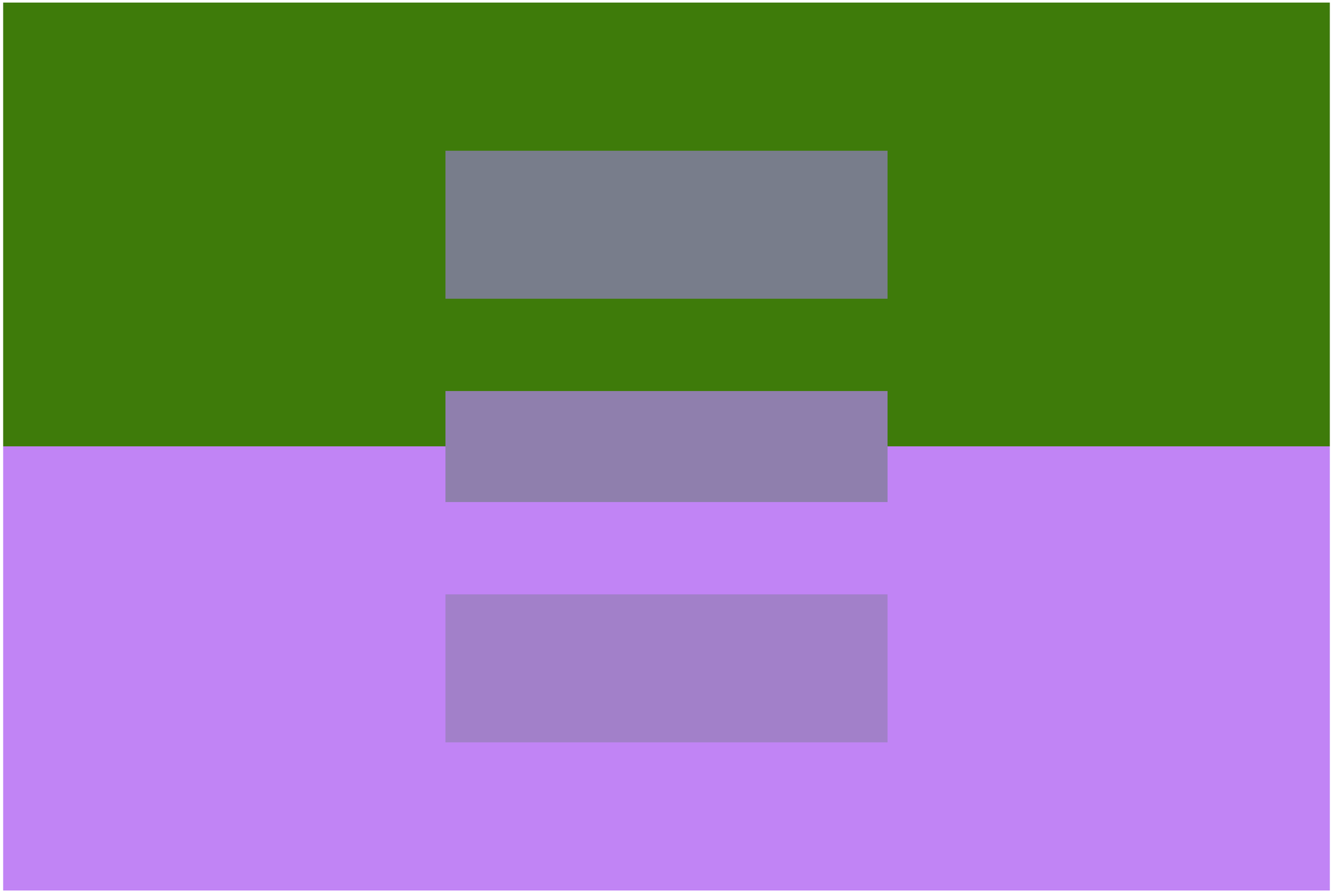
```
fill(mixed2);
rect((pgwidth)/3,  (pgheight*2)/3, pgwidth/3, pgheight/6);


popMatrix();
```

CHAPTER 04: 5 COLORS INTO 3

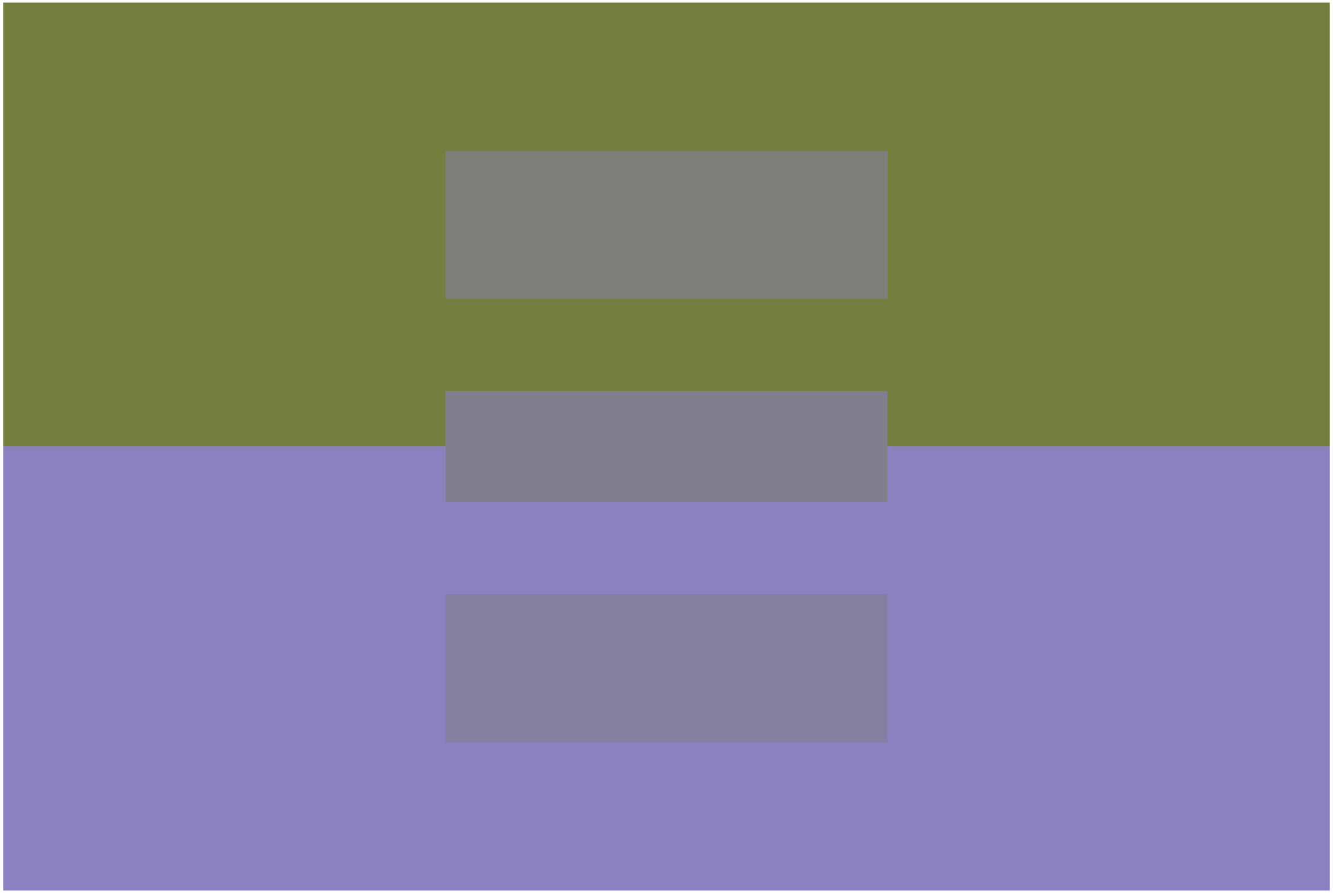
0xFFBA0D00 0xFF45F2FF 0xFF9D8A91 0xFF8CABB4 0xFF78C6D1



0xFF3E7B0A 0xFFC184F5 0xFF787D8B 0xFF8F7FAD 0xFFA280C9



0xFF737F3F 0xFF8C80C0 0xFF7B7F78 0xFF807F8E 0xFF847FA1



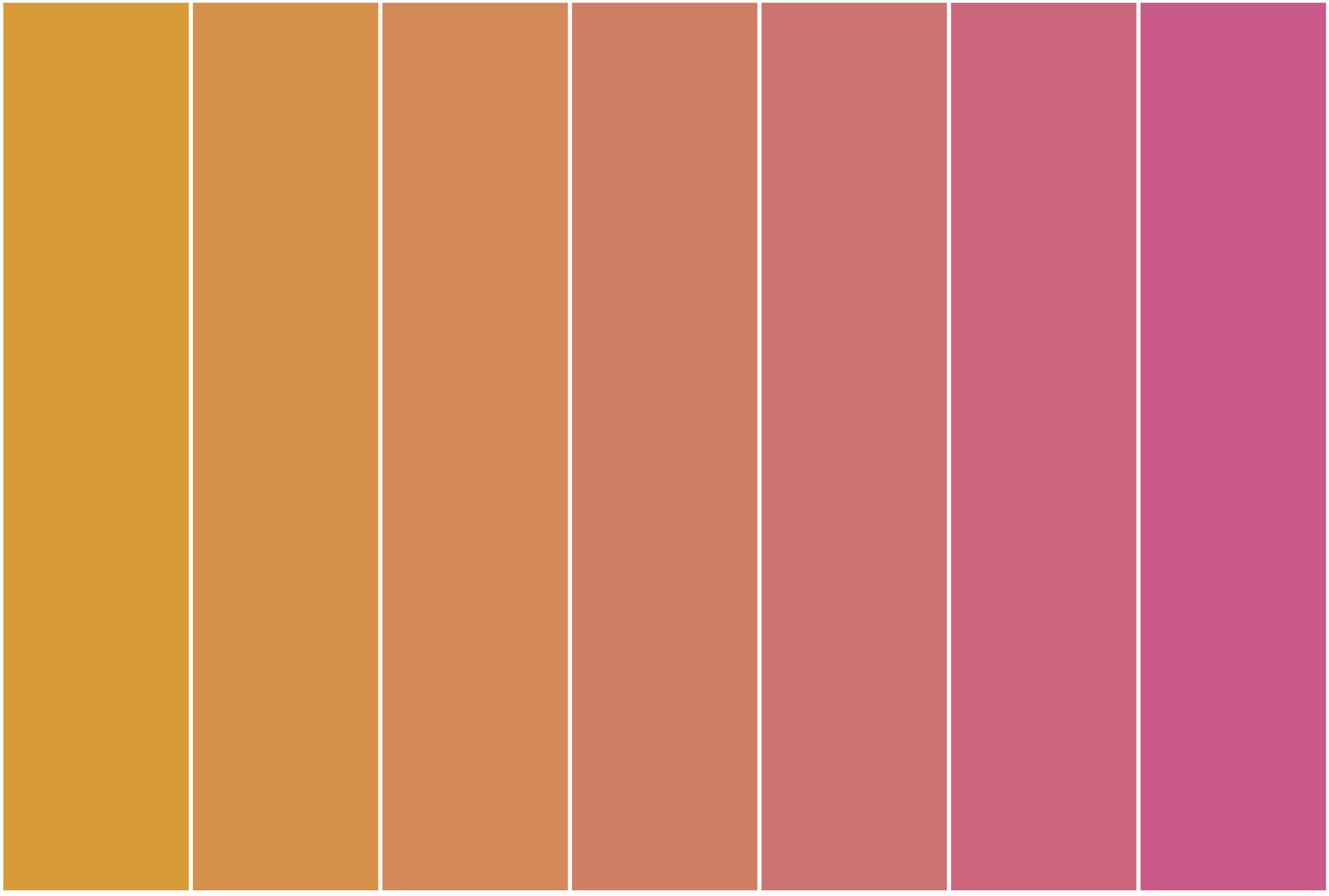
```
// recipe for making 5 colors look like 3
// prepare the first and second colors
SecureRandom random = new SecureRandom();

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();
```

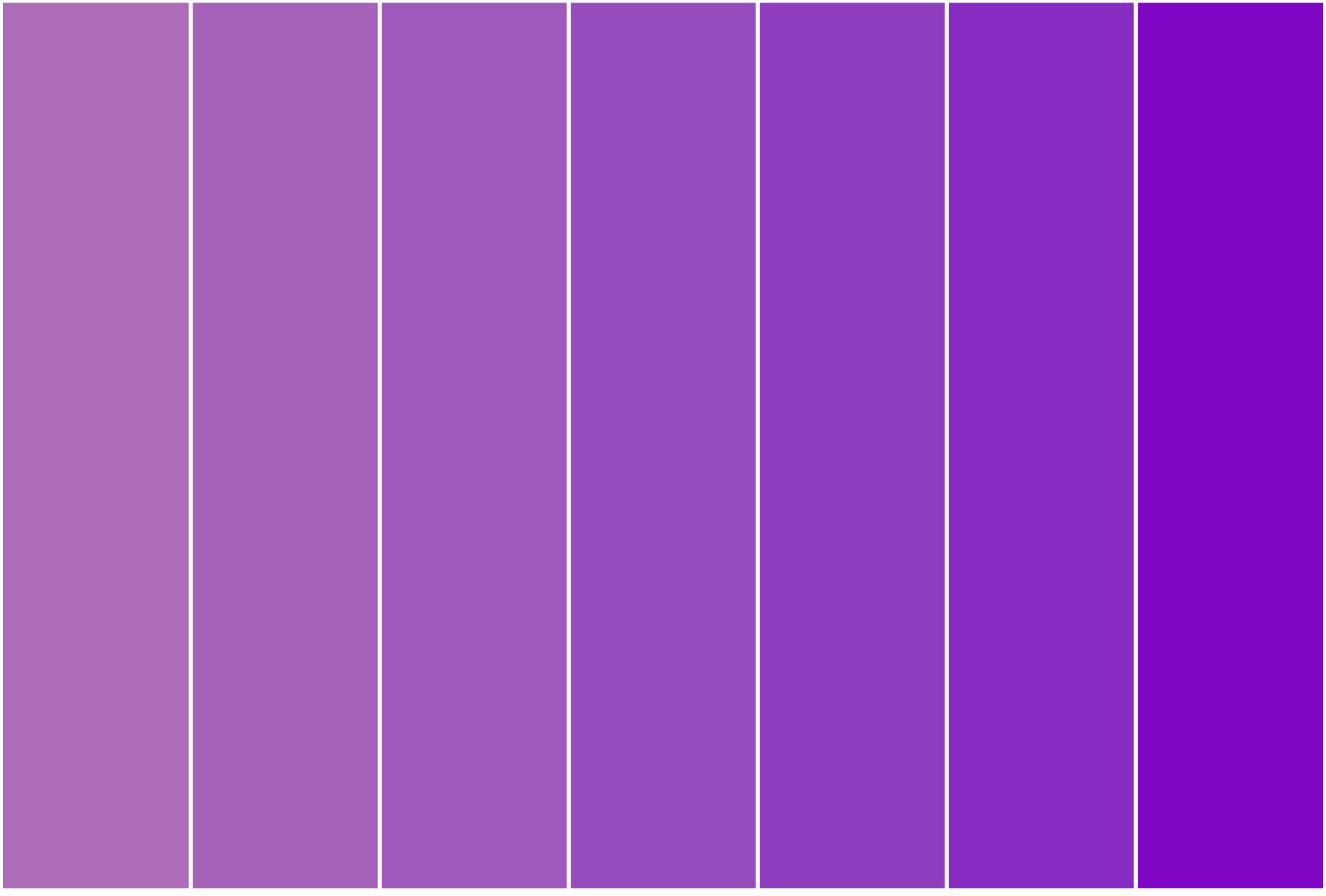
```
fill(mixed2);
rect((pgwidth)/3,  (pgheight*2)/3, pgwidth/3, pgheight/6);

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

popMatrix();
```

CHAPTER 05: COLOR MODULATION

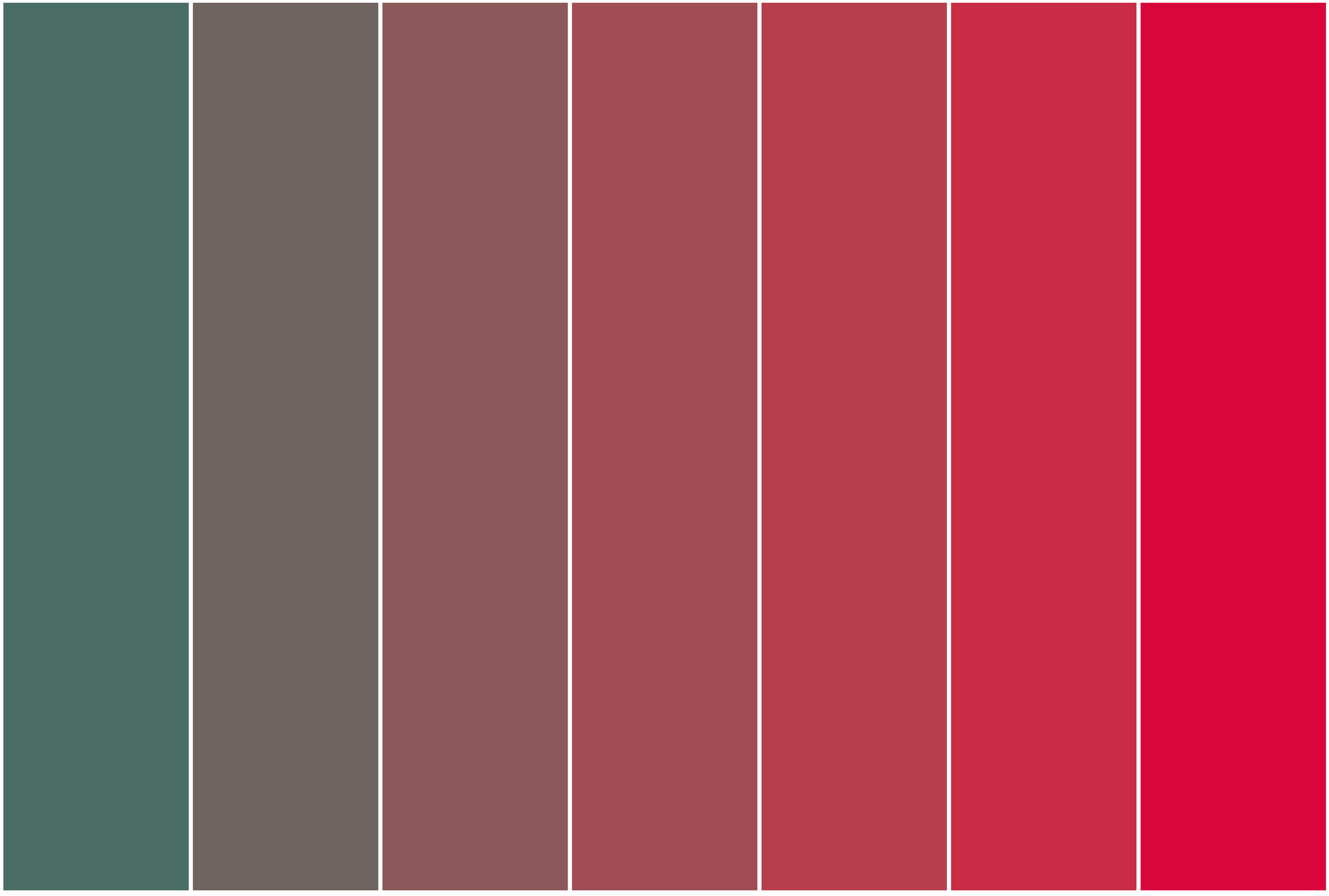
0xFFD89A36 0xFFD5914A 0xFFD3885A 0xFFD17E67 0xFFCE7373 0xFFCC677E 0xFFC95988



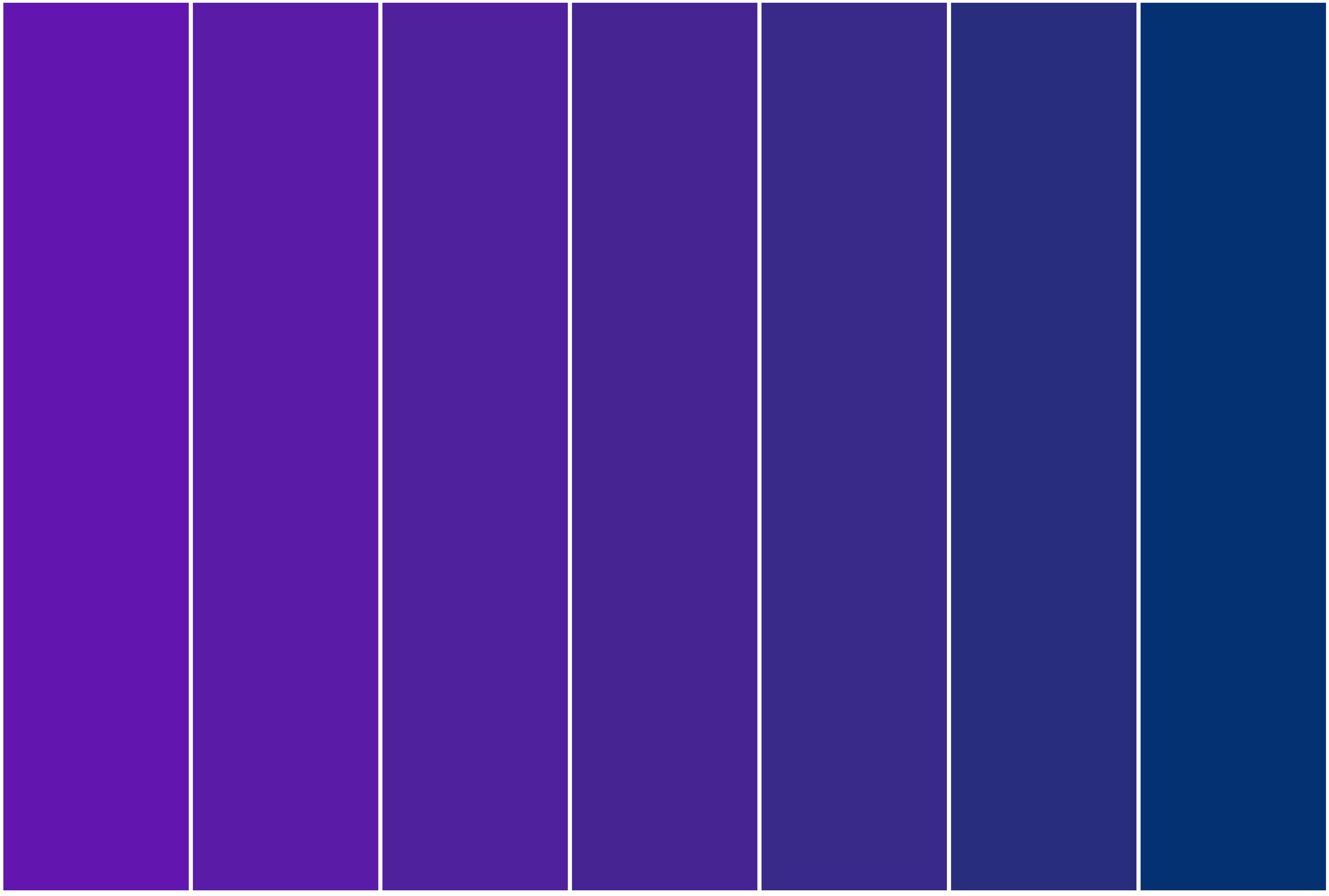
0xFFAD6CB7 0xFFA662B9 0xFF9F58BB 0xFF974CBE 0xFF8F3EC0 0xFF872CC2 0xFF7E06C4



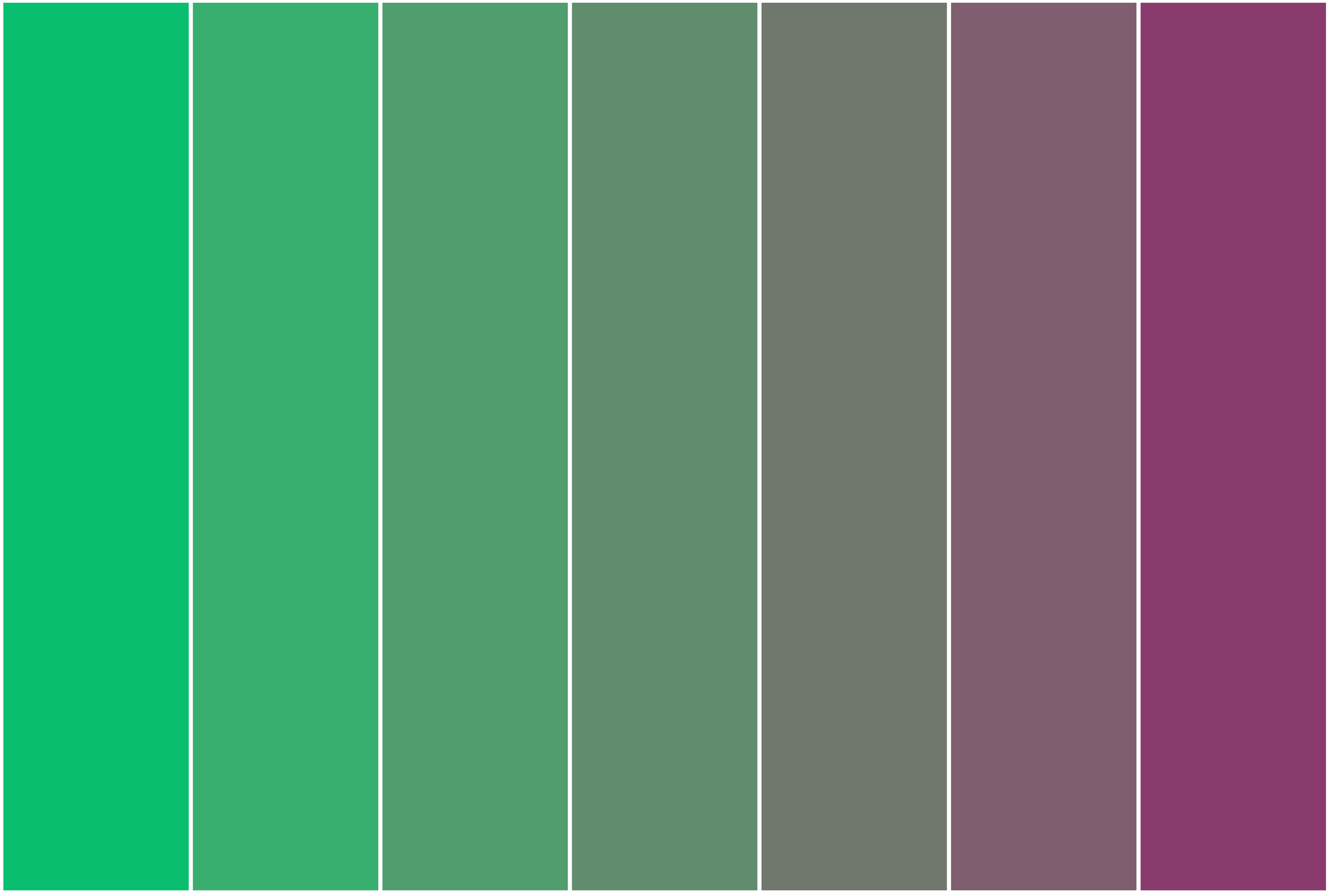
0xFF4A6E66 0xFF6F6460 0xFF8B595A 0xFFA24D54 0xFFB63F4D 0xFFC82D45 0xFFD9063C



0xFF6316AF 0xFF5A1CA6 0xFF50219D 0xFF462593 0xFF392989 0xFF282D7E 0xFF033171



0xFF0ABE6F 0xFF39AF6F 0xFF509E6E 0xFF618C6E 0xFF70786E 0xFF7E5E6F 0xFF8A3B6E

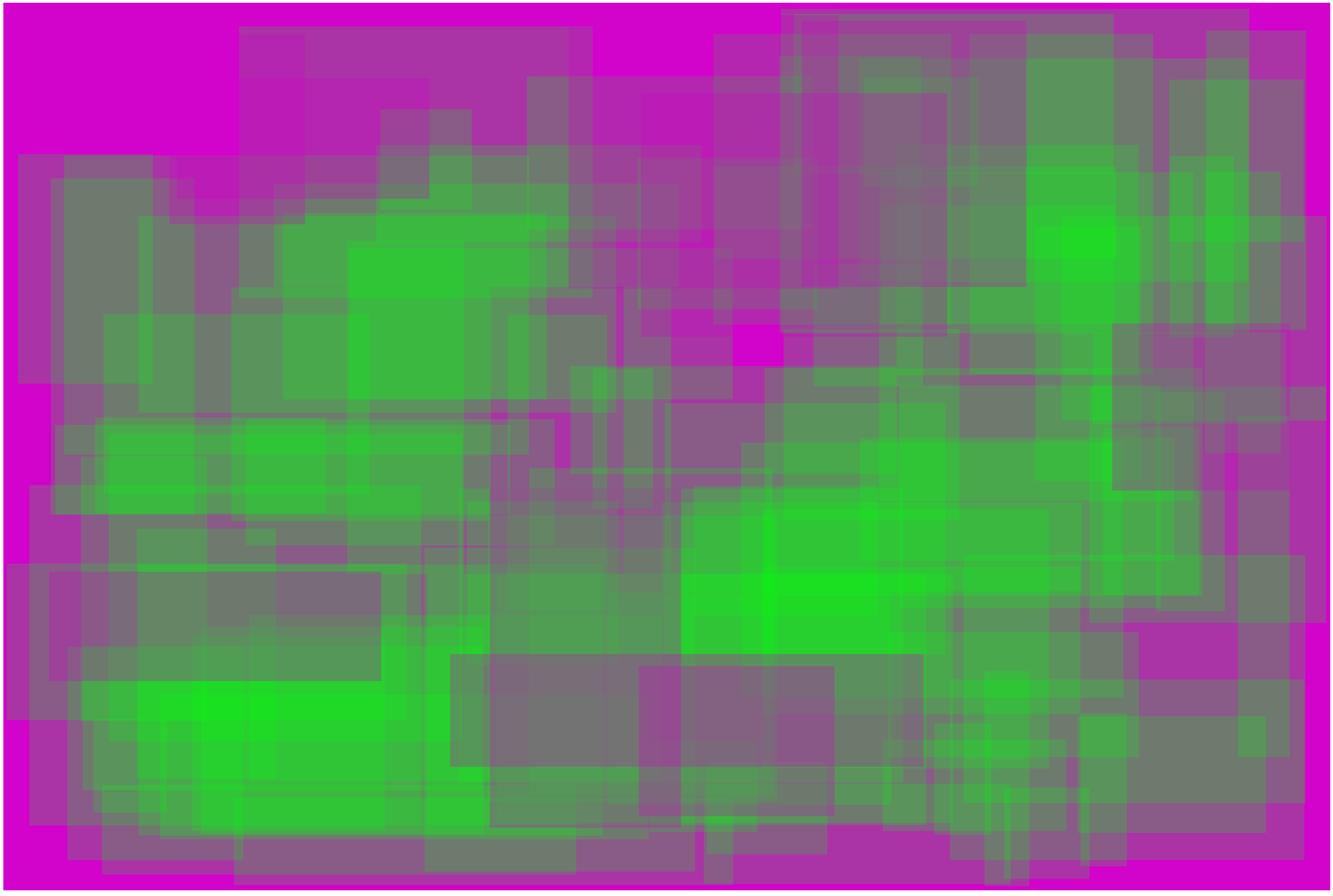


```
// recipe for color modulation

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();
```

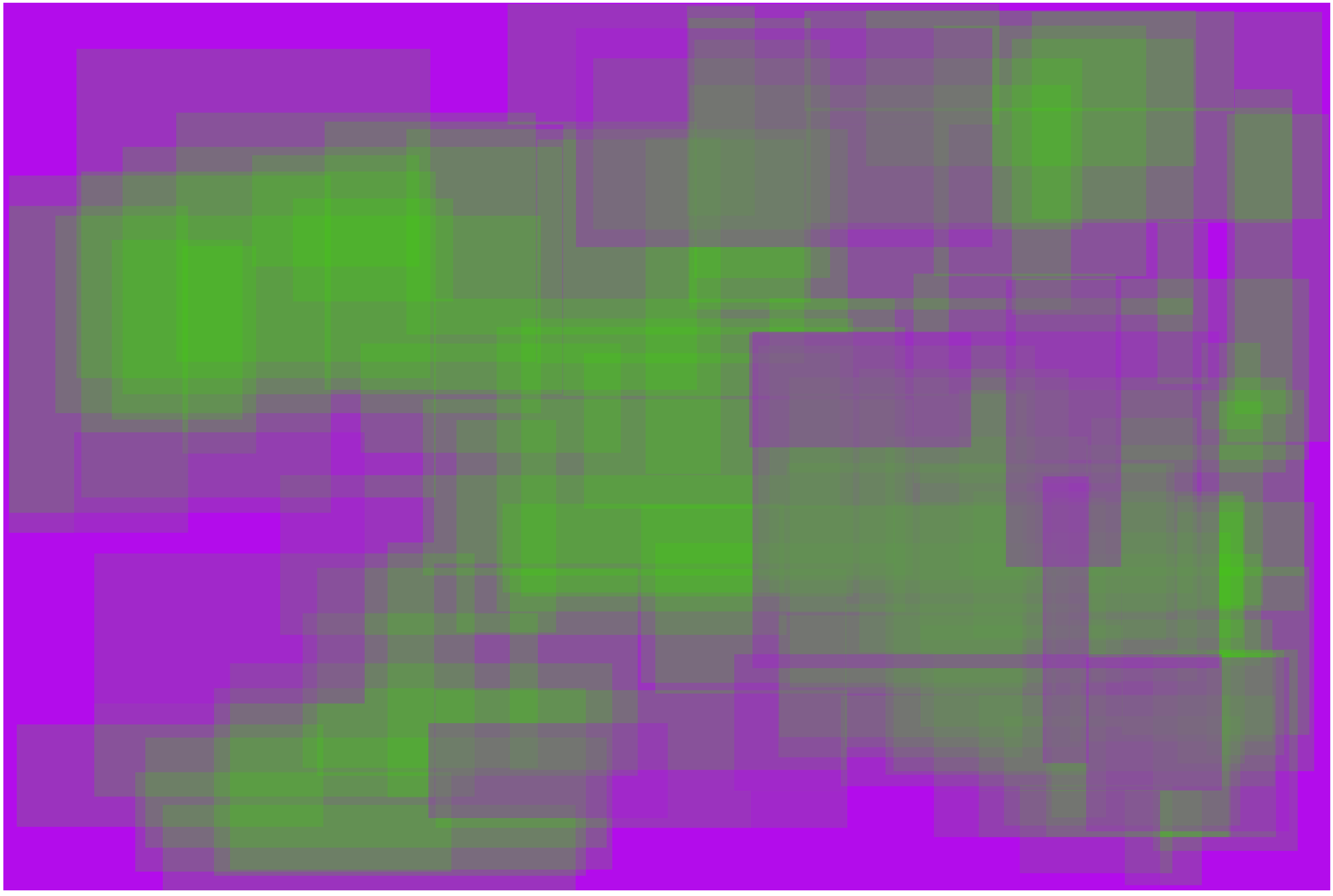
```
for(int i = 0; i < numsteps; i++) {
  // get a weighted average of the red, green, blue channels
  float mixedred = sqrt((sq(red(col1))*c1_weight +sq(red(col2))*c2_weight));
  float mixedgreen = sqrt((sq(green(col1))*c1_weight +sq(green(col2))*c2_weight));
  float mixedblue = sqrt((sq(blue(col1))*c1_weight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

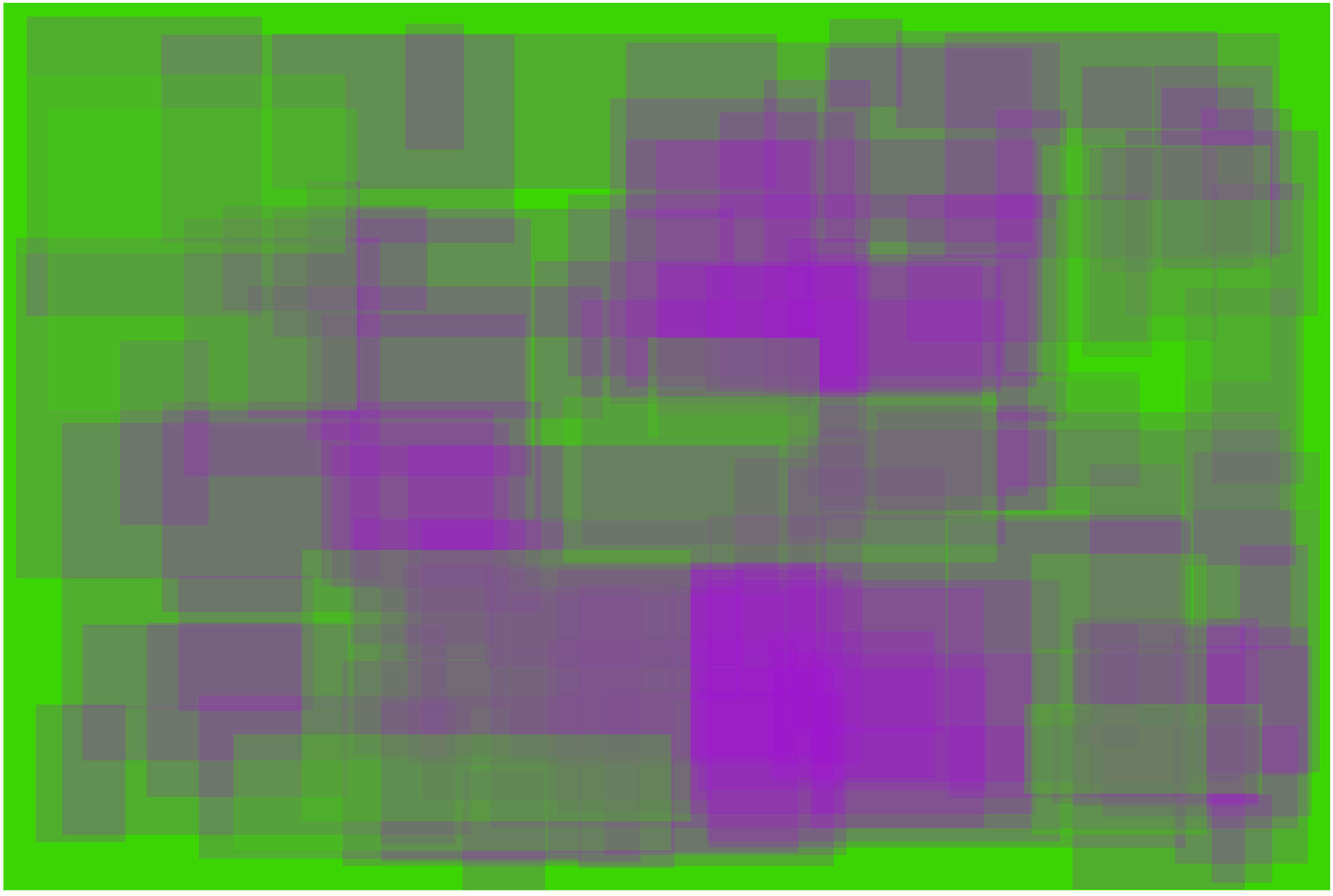
0xFF06FE0F 0xFFD203CA



0xFF3CD608 0xFFB30CEB



0xFFA610DC 0xFF3BD504

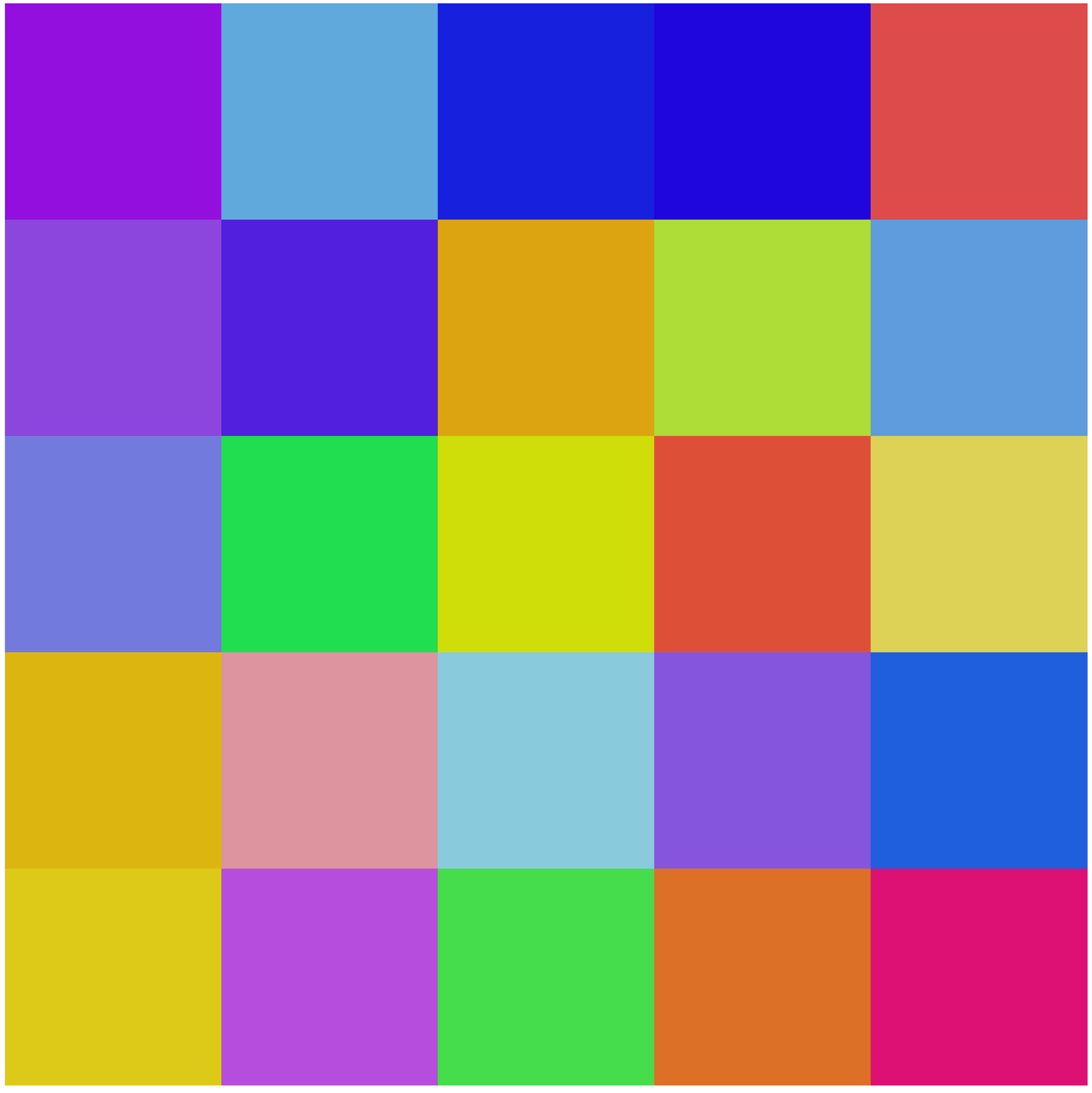


```
// recipe for using bridging colors

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

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

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
// add a touch of randomness
color bg = color((hue(col1)+180)%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);
```

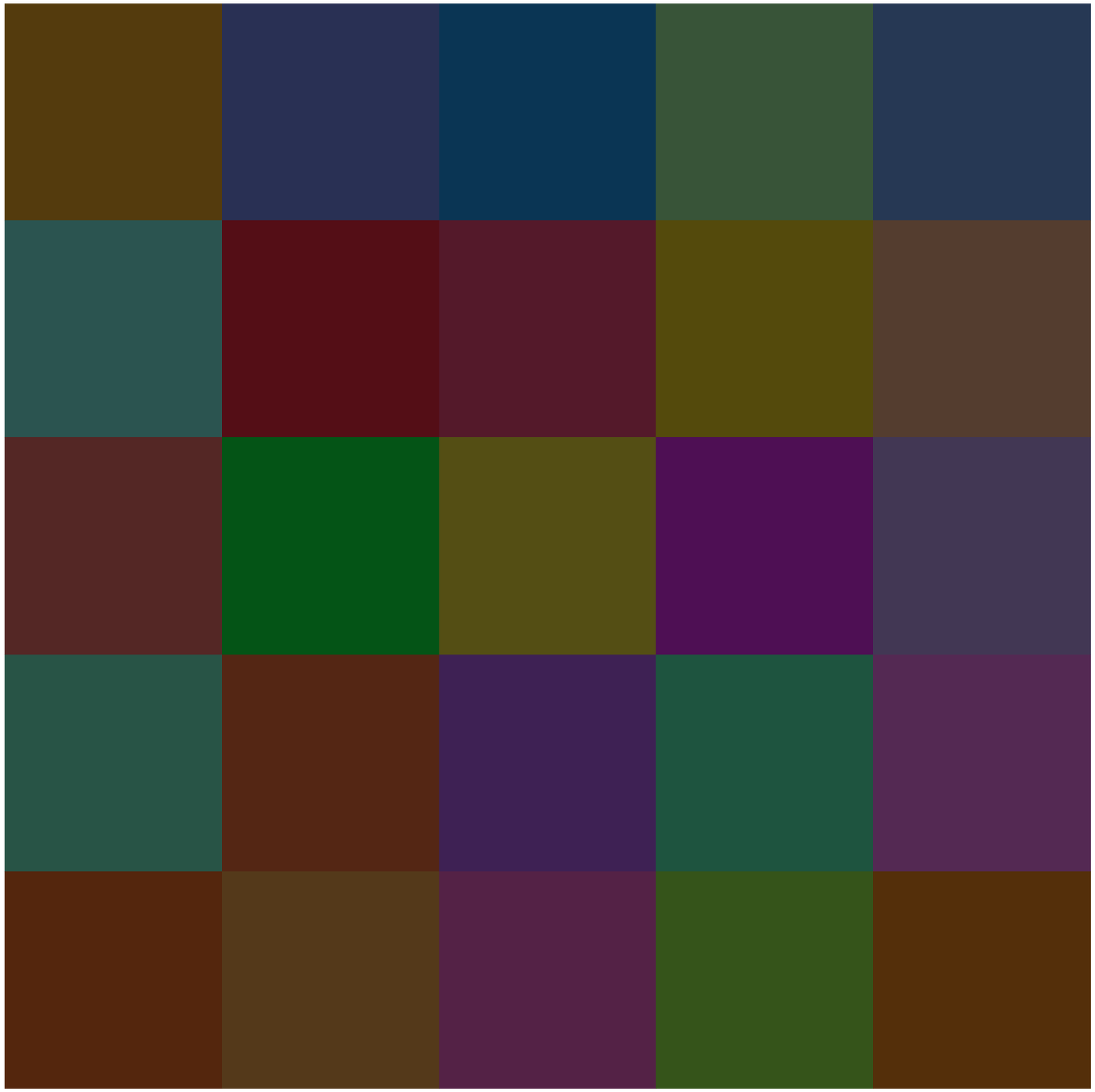
```
    float recth = random(200, min(800, pgheight-posY));

    // 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);

  }

  popMatrix();
  colorMode(RGB);
```

CHAPTER 07: SAME VALUE STUDIES (AFTER JOHANNES ITTEN)

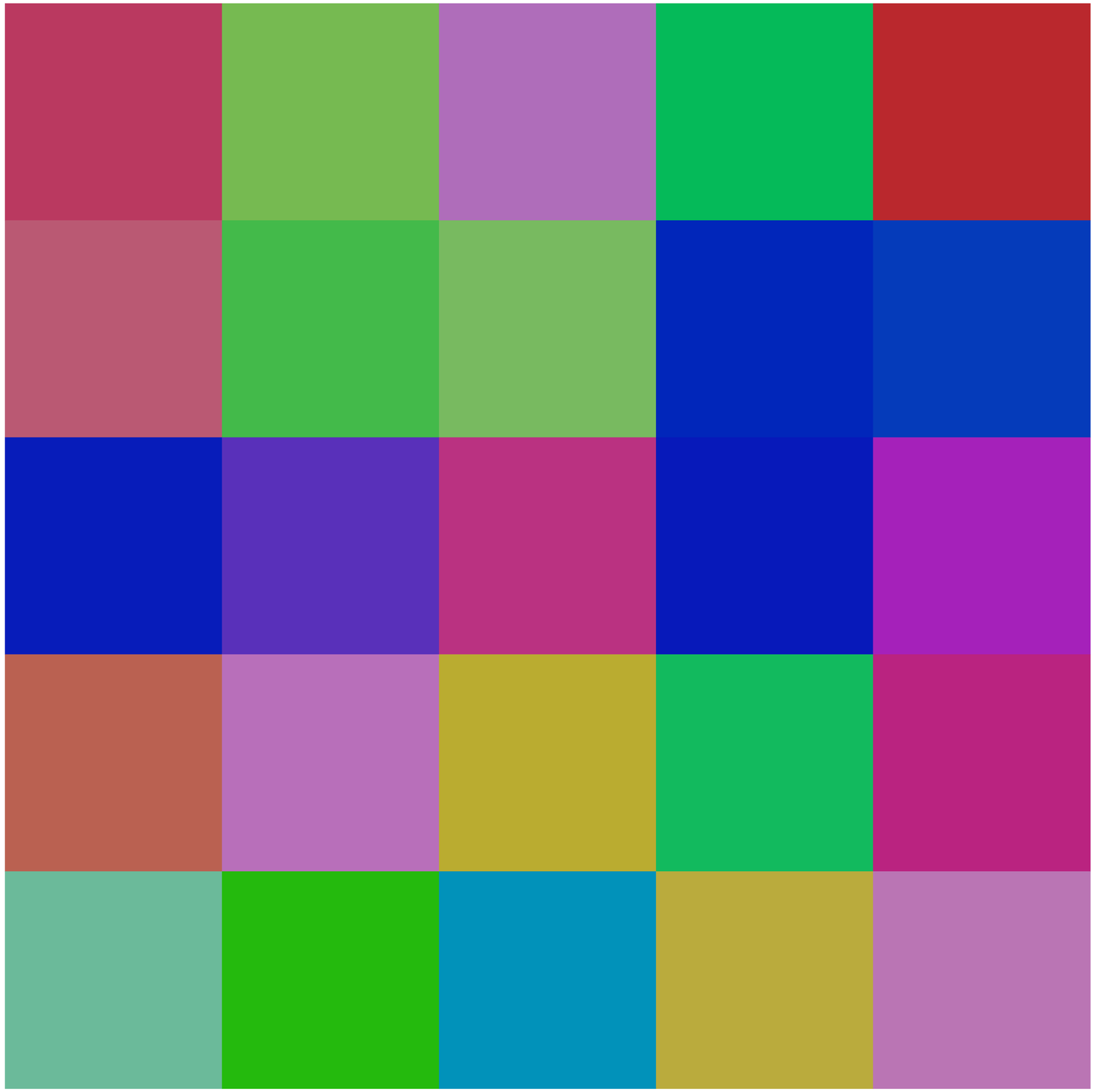
value: 87



value: 33



value: 73



```
// recipe for Johannes Itten studies

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

int min = 20;
int max = 100;
int value = random.nextInt(max-min+1)+min;

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

// sprinkle squares in a matrix, randomly
for(int row = 0; row < 5; row++) {
  for(int col = 0; col < 5; col++) {
    color squarecol = color(random.nextInt(361), random.nextInt(71)+30, value);

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

}

popMatrix();
```

APPENDIX: PALETTES

artist:
Paul Klee

0xFF724F33
0xFF53442B
0xFF383923
0xFFAA241F
0xFFC33919
0xFFCE8104
0xFF8E857A
0xFFD50E23

artist:
Sara Cwynar

0xFF8EB0BA
0xFF679AB0
0xFF4385A4
0xFF13729B
0xFF046590
0xFFB7C1C3
0xFFDBD7D2
0xFF032E5A
0xFF37ACD5

artist:
Sara Cwynar

0xFFF4AC9E
0xFFDD725F
0xFFB84C3B
0xFF8D3526
0xFFED7964
0xFFD14C3D

artist:
James Jean

0xFF484D5B
0xFF24445A
0xFF0D808C
0xFF507079
0xFE8D8589
0xFFB6AEAE
0xFFD3CCCA
0xFFFAF8C0
0xFF86575D
0xFF150E2A
0xFFB23E40
0xFECF6464
0xFFDA3434
0xFF6A4D56

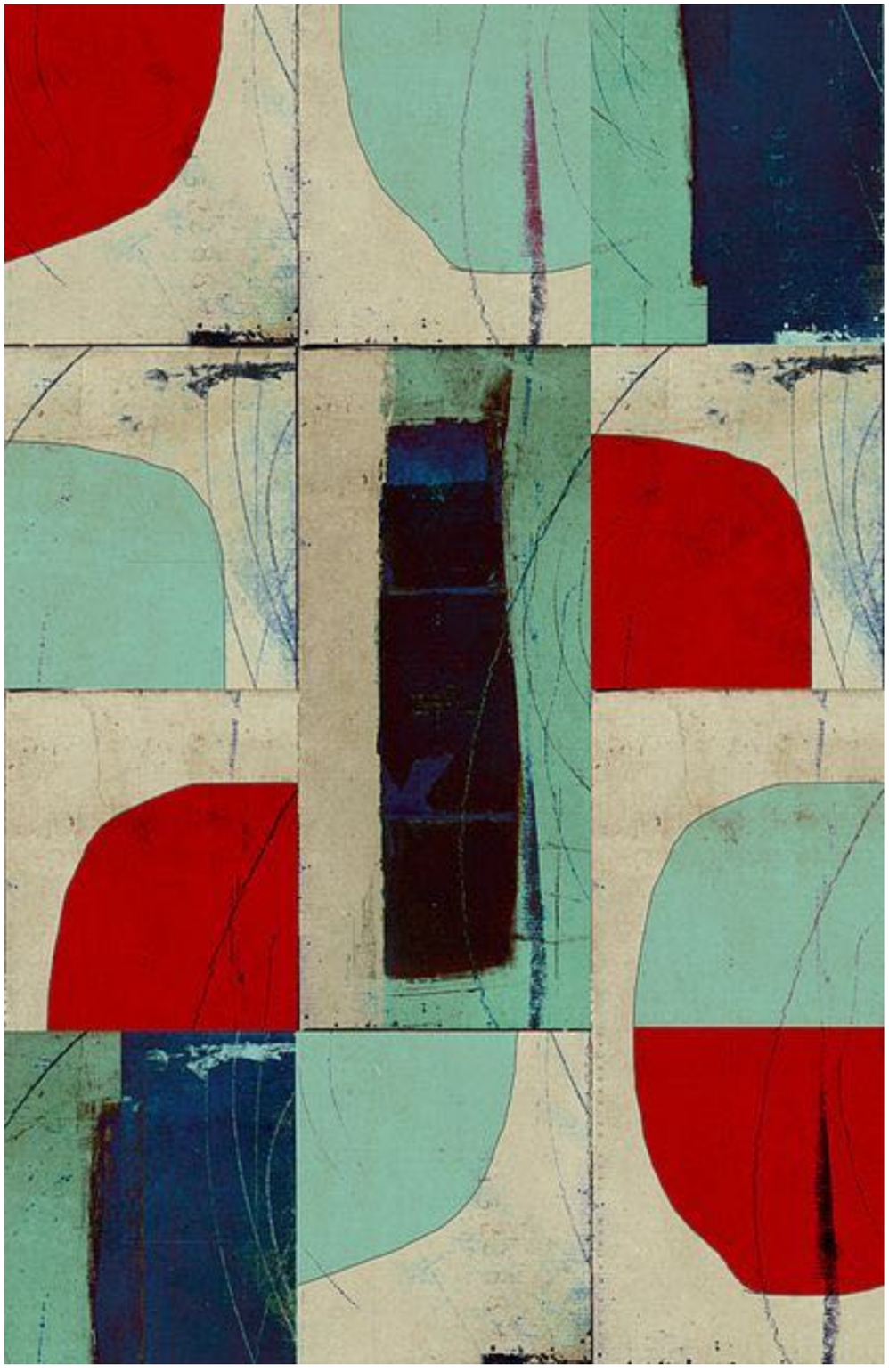
artist:
Sean Soong

0xFF0F1819
0xFF2E3B38
0xFF5E4A3A
0xFF4E6F6D
0xFF888D7E
0xFFBDCCBD
0xEE81583C
0xFF203C3D
0xFFA19C86

artist:
Jenny Saville

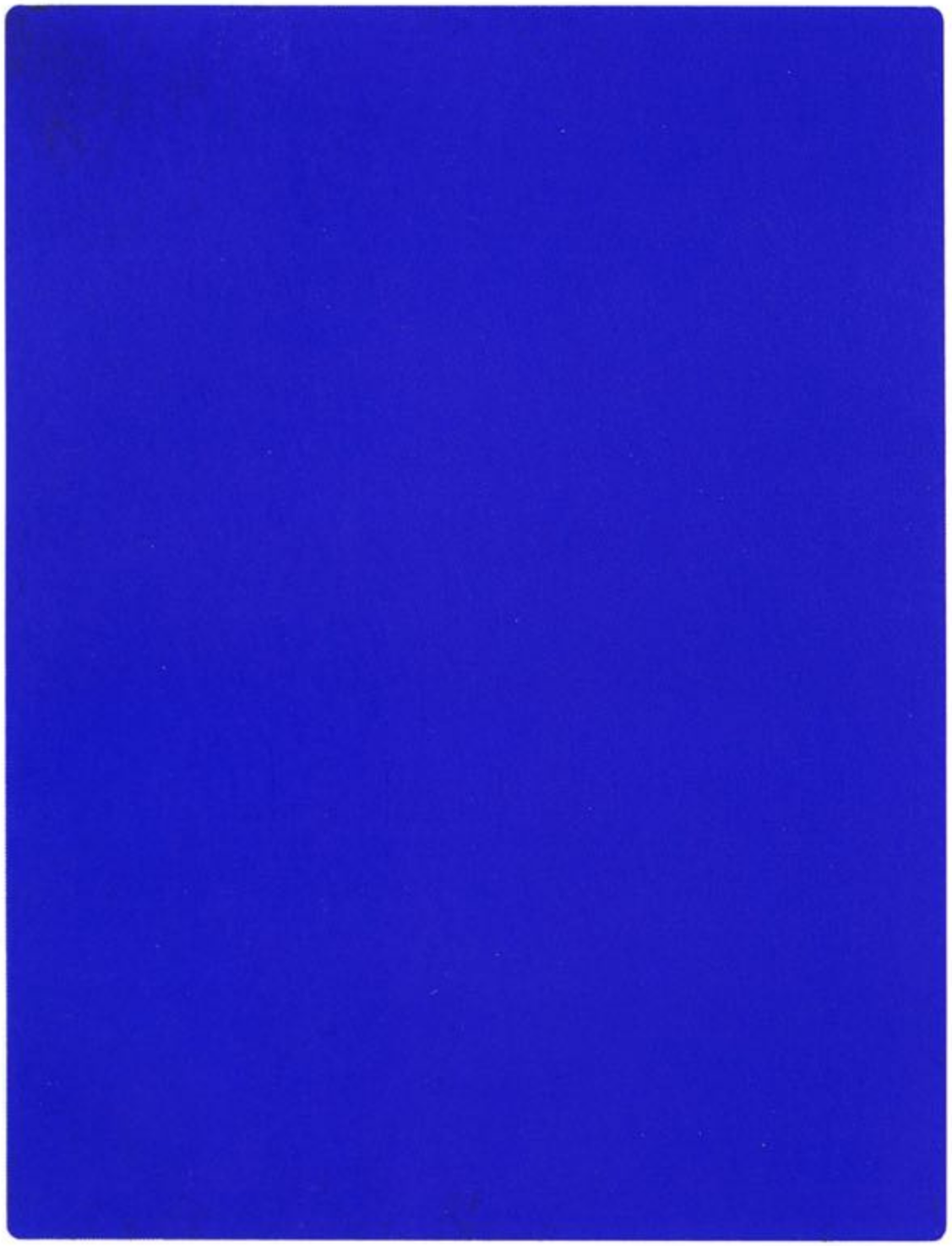
0xFFE1CCCE
0xFFC2A5AC
0xFF9C8EA2
0xFF7B81A1
0xFF6780A7
0xFF503D59
0xFF8EA1C2
0xFF30202C
0xFF4A5A8F
0xFF140B0C
0xFFABB2CC
0xFFDFADA2

artist:
Linda Vachon



0xFFA50202
0xFF91100D
0xFF96AB93
0xFFC1BB9E
0xFF1D1B2A
0xFF7CA78E
0xFF4F6756
0xFF0F1C2F
0xFFA7B099
0xFF162640

artist:
Yves Klein



0xFF201BC4
0xFF201BC4

artist:
Tatsuyuki Tanaka

0xFFAF9888
0xFFD4C0B7
0xFF8B7667
0xFF624E41
0xFF423126
0xFF211811
0xFF0F0E08

artist:
Jamini Roy

0xFFFAC74D
0xFF9C3C3F
0xFF88393A
0xFF251F22
0xFF1D1A1F
0xFFE1D9D0

```
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 RAFAEL ABREU-CANEDO
FOR YOUR GUIDANCE AND SUPPORT.