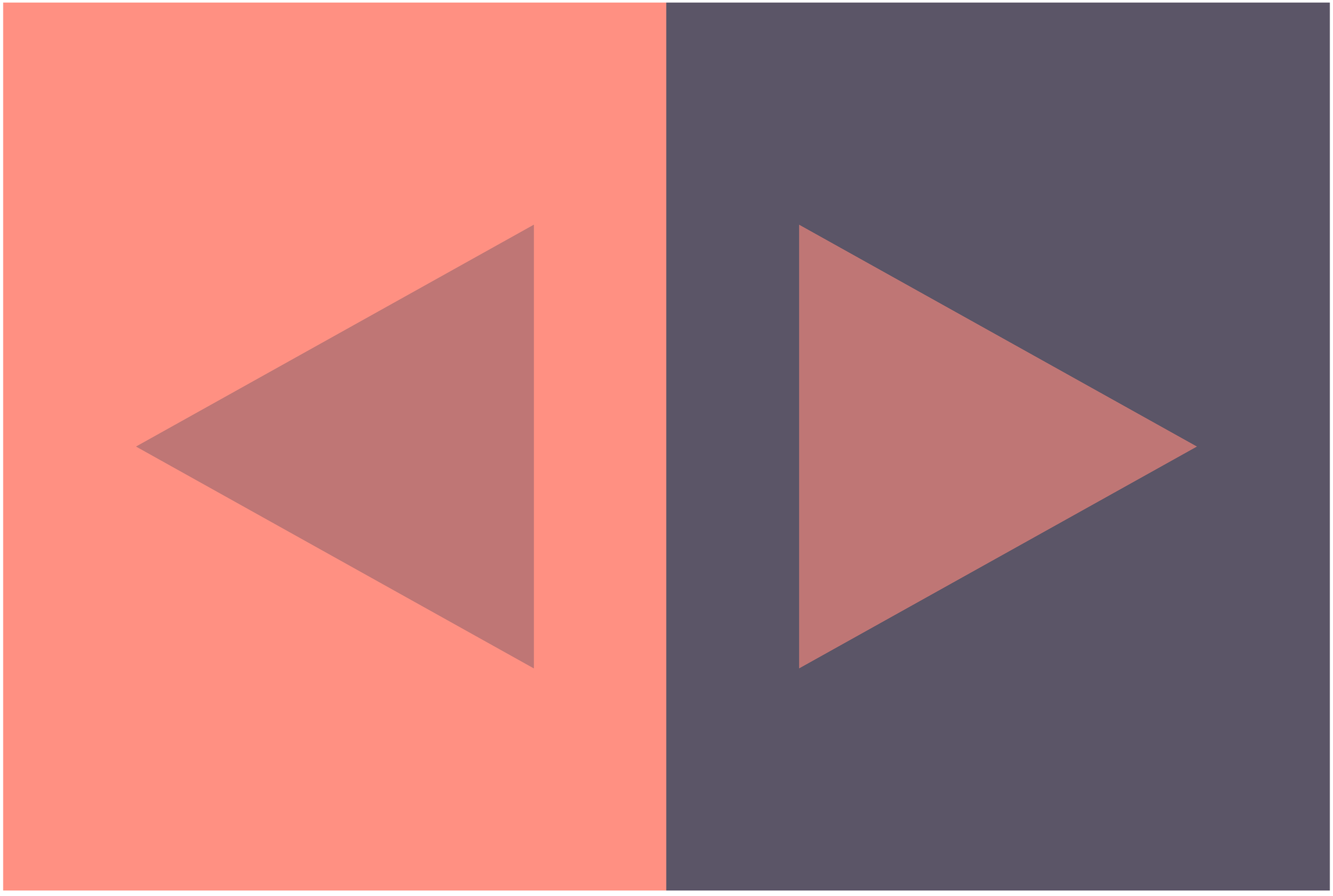
THE COLORIST COOKBOOK

"COOKING IS LIKE PAINTING OR WRITING A SONG. JUST AS THERE ARE ONLY SO MANY NOTES OR COLORS, THERE ARE ONLY SO MANY FLAVORS - IT'S HOW YOU COMBINE THEM THAT SETS YOU APART."

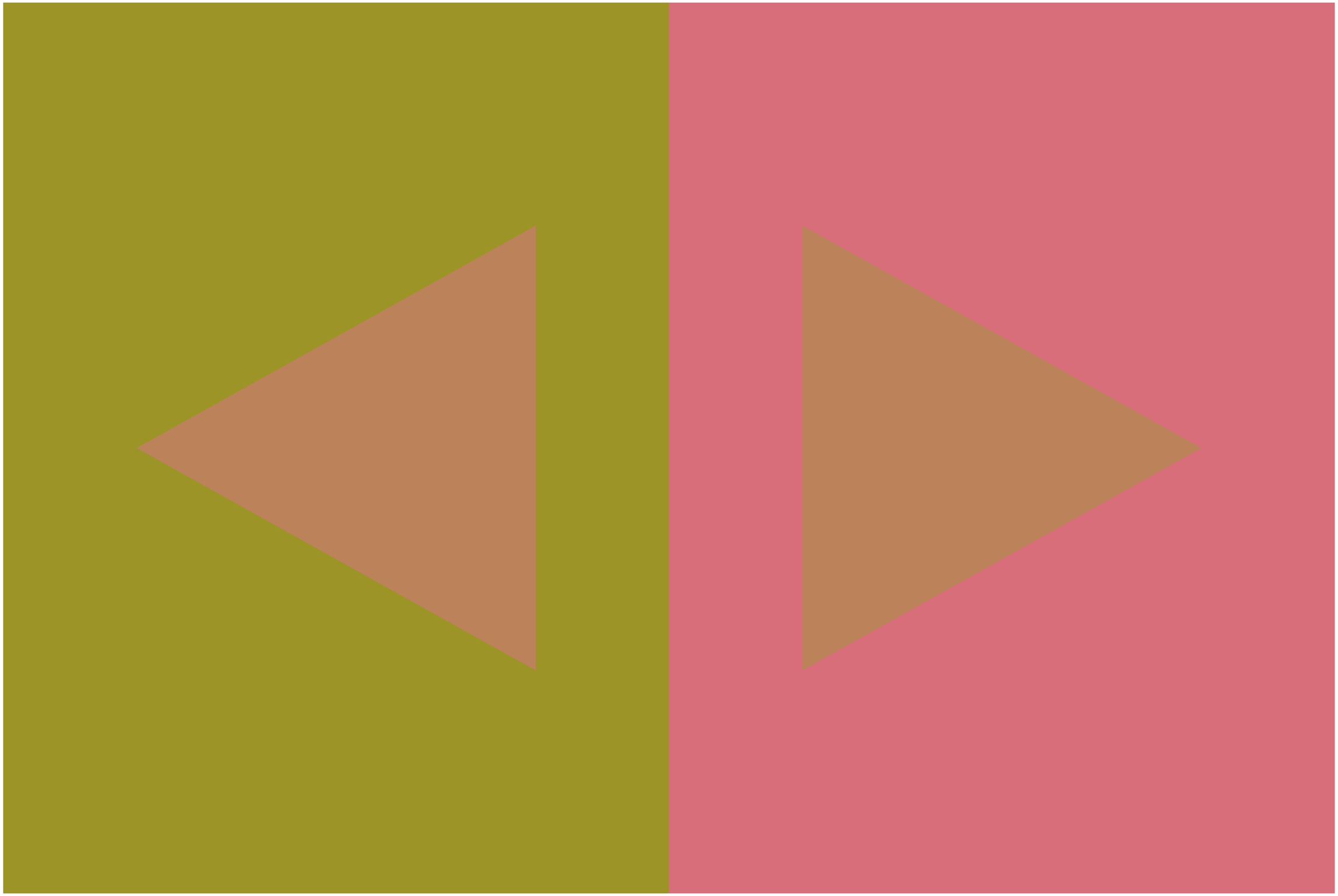
CHAPTER 01: 3 COLORS INTO 4 PART I

0xFFFF9082 0xFF5B5567 0xFFBF7675

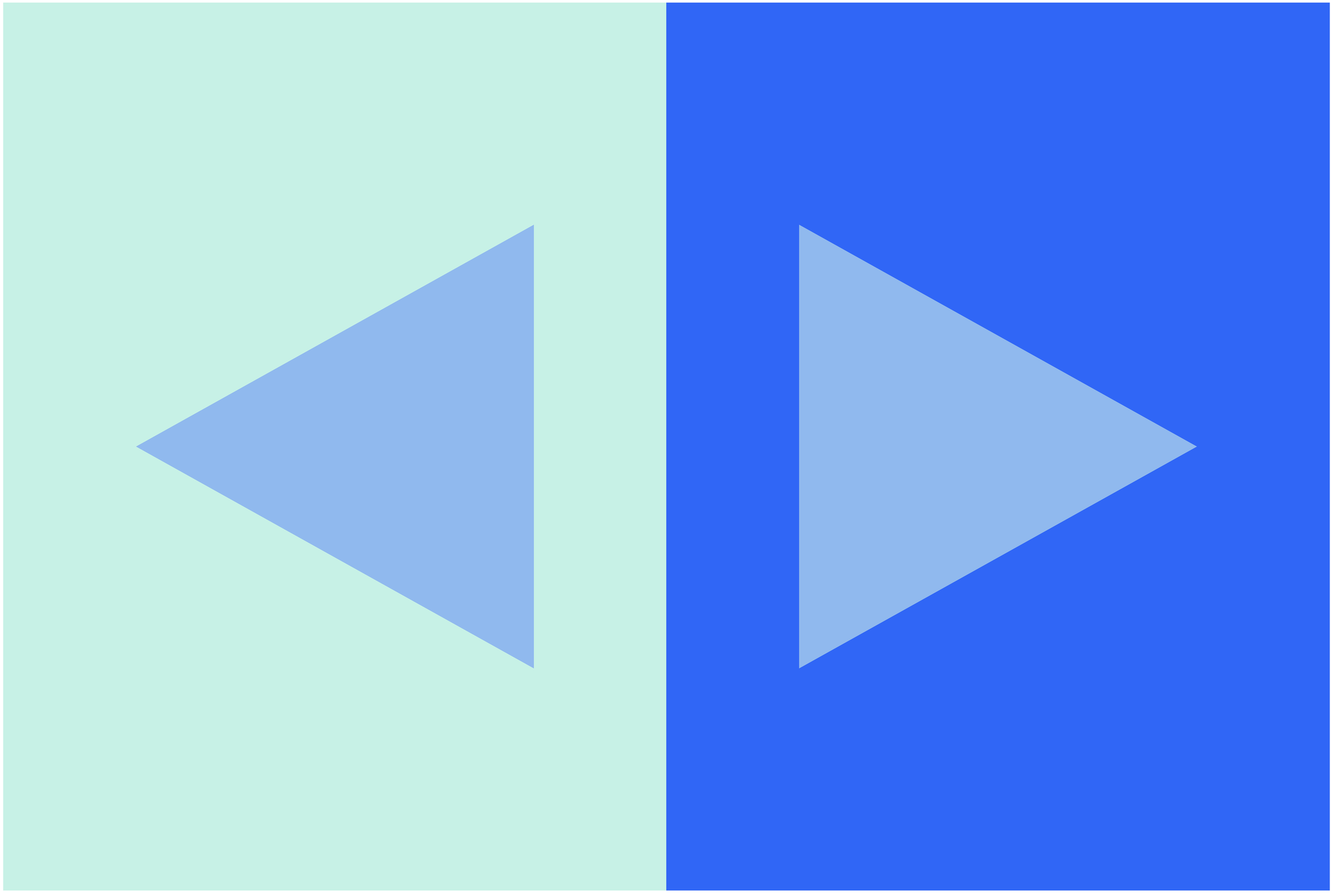


0xFF7C9B99 0xFF728490 0xFF778F94

0xFF9C9426 0xFFD86E7A 0xFFBC825A



0xFFC7F1E6 0xFF3066F6 0xFF90B9EE

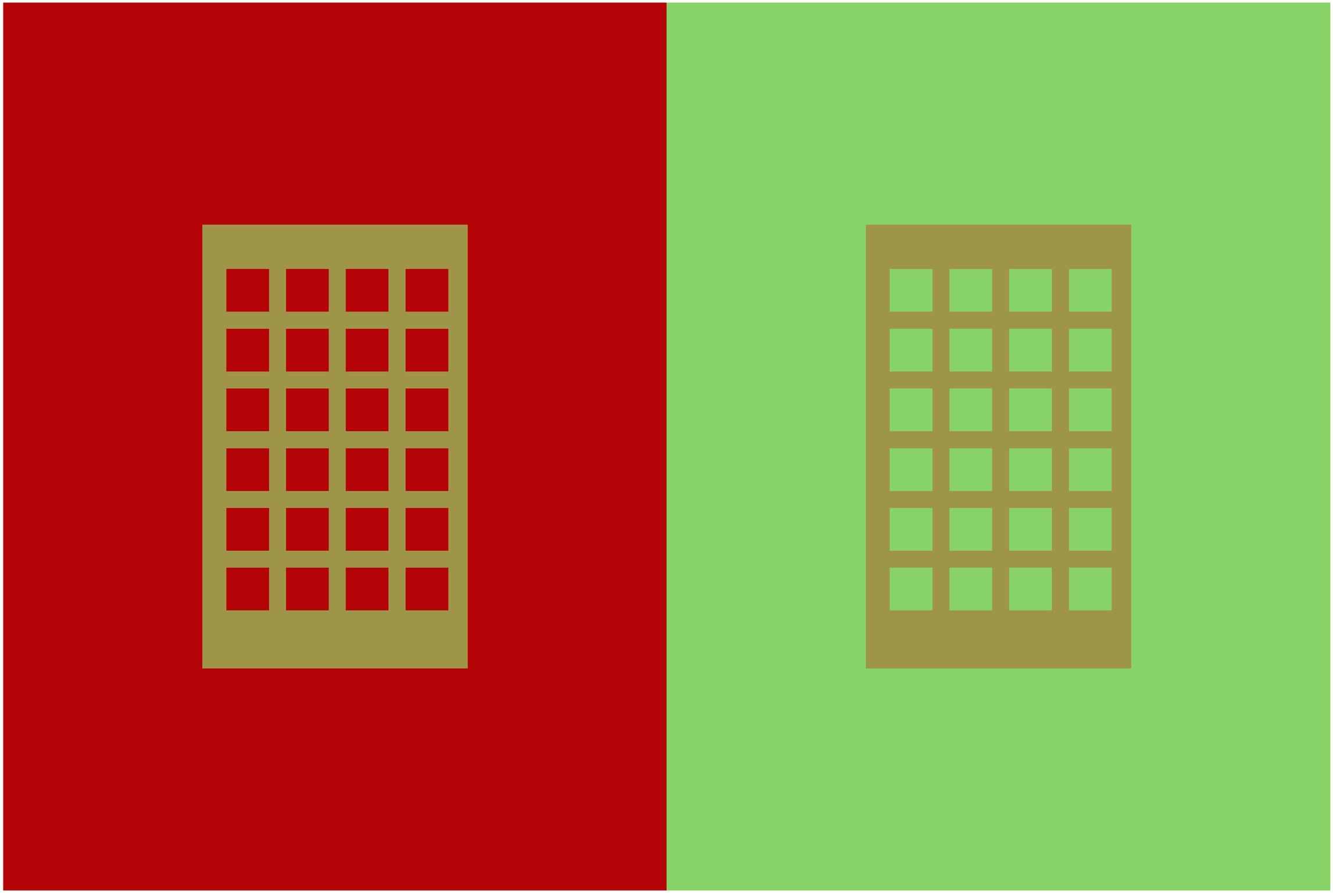


```
// recipe for making 3 colors look like 4

// 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);
```

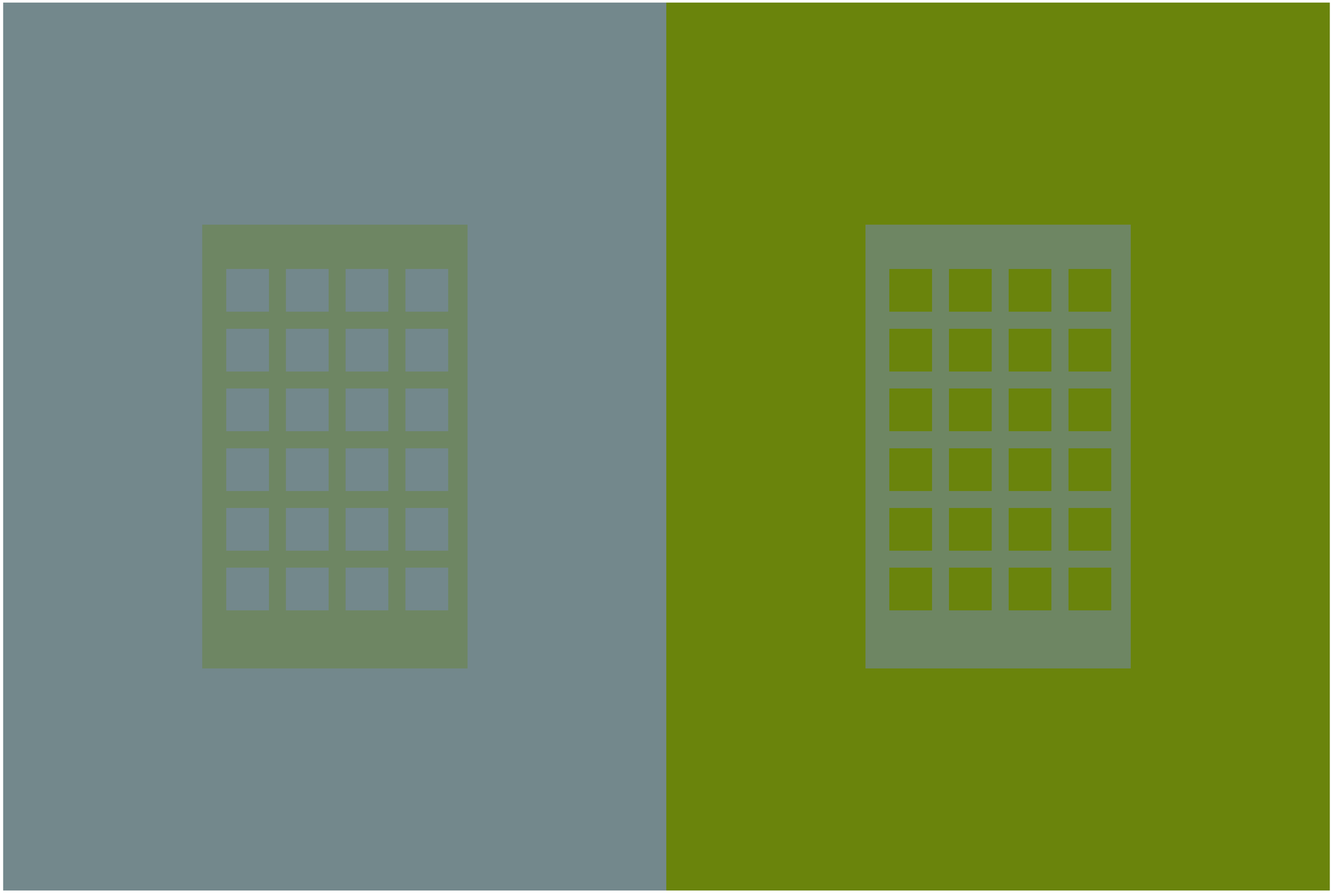
```
// arrange the first two colors beside each other
fill(col1);
stroke(col1);
rect(0,0,pgwidth/2f,pgheight);
fill(col2);
stroke(col2);
rect(pgwidth/2f,0,pgwidth/2f,pgheight);

// 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

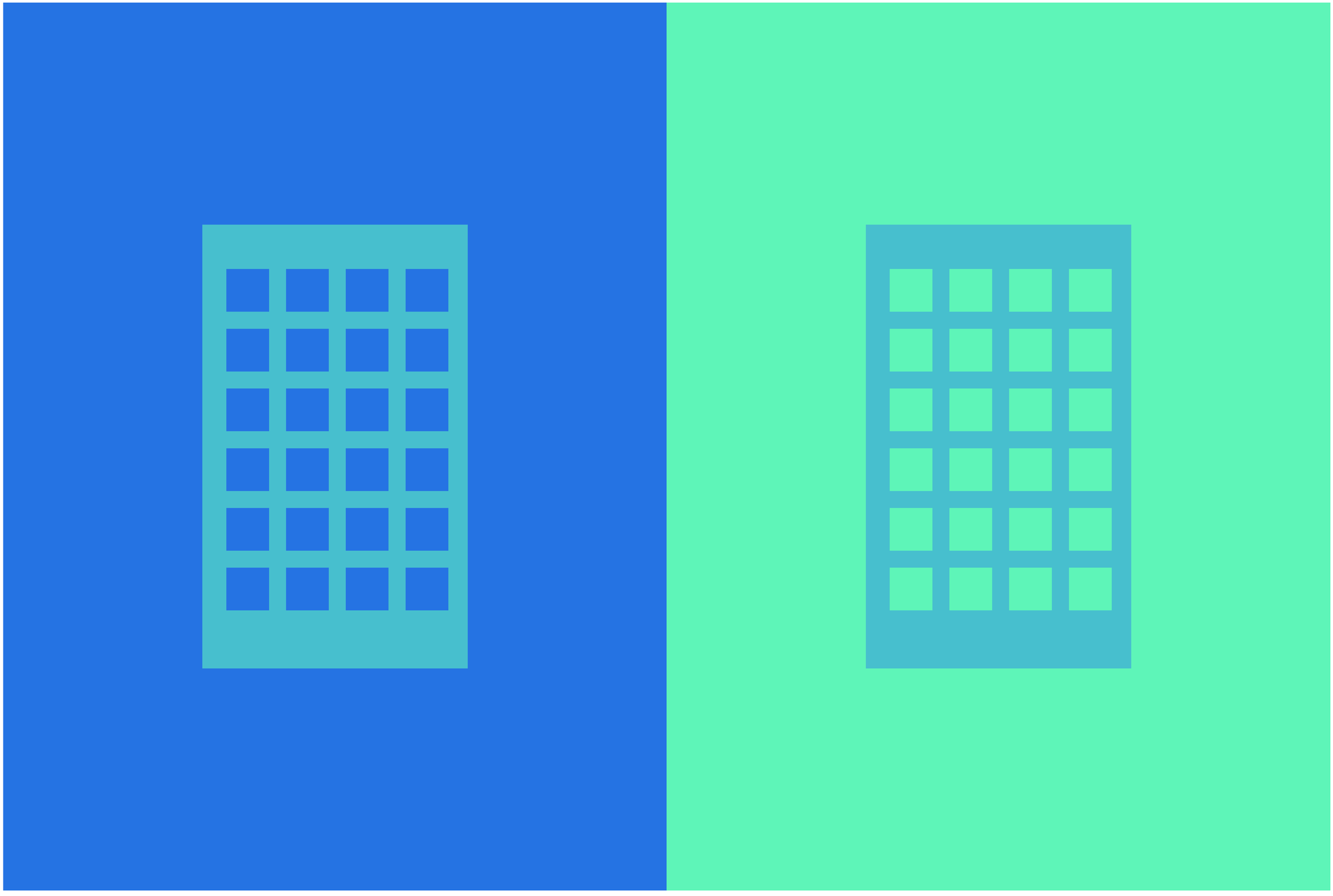
0xFFB40408 0xFF88D367 0xFF9F9549



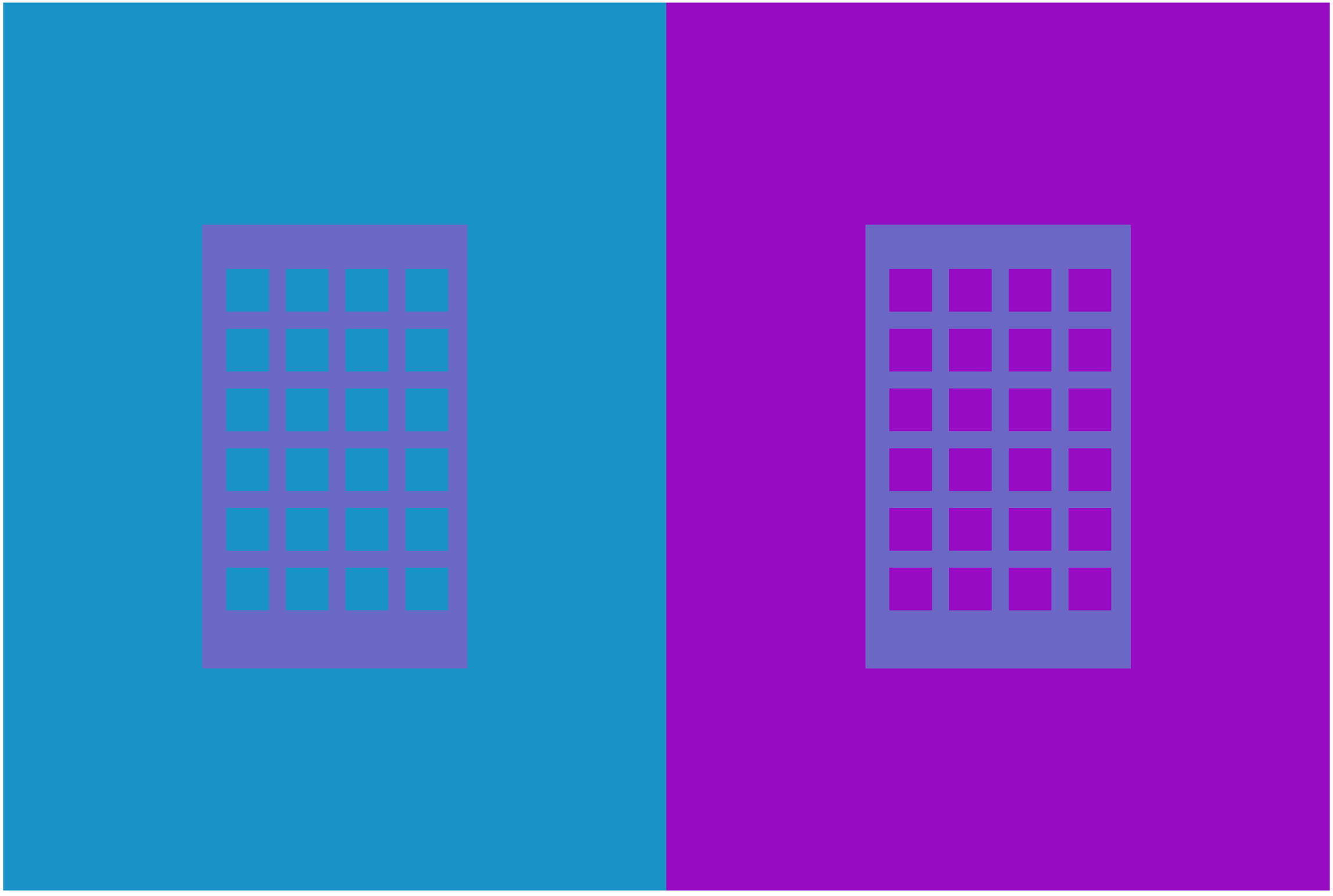
0xFF73888C 0xFF6A840C 0xFF6E8663



0xFF2573E3 0xFF5EF5B8 0xFF47BFCE



0xFF1993C7 0xFF960BC3 0xFF6B68C5



```
// 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);
```

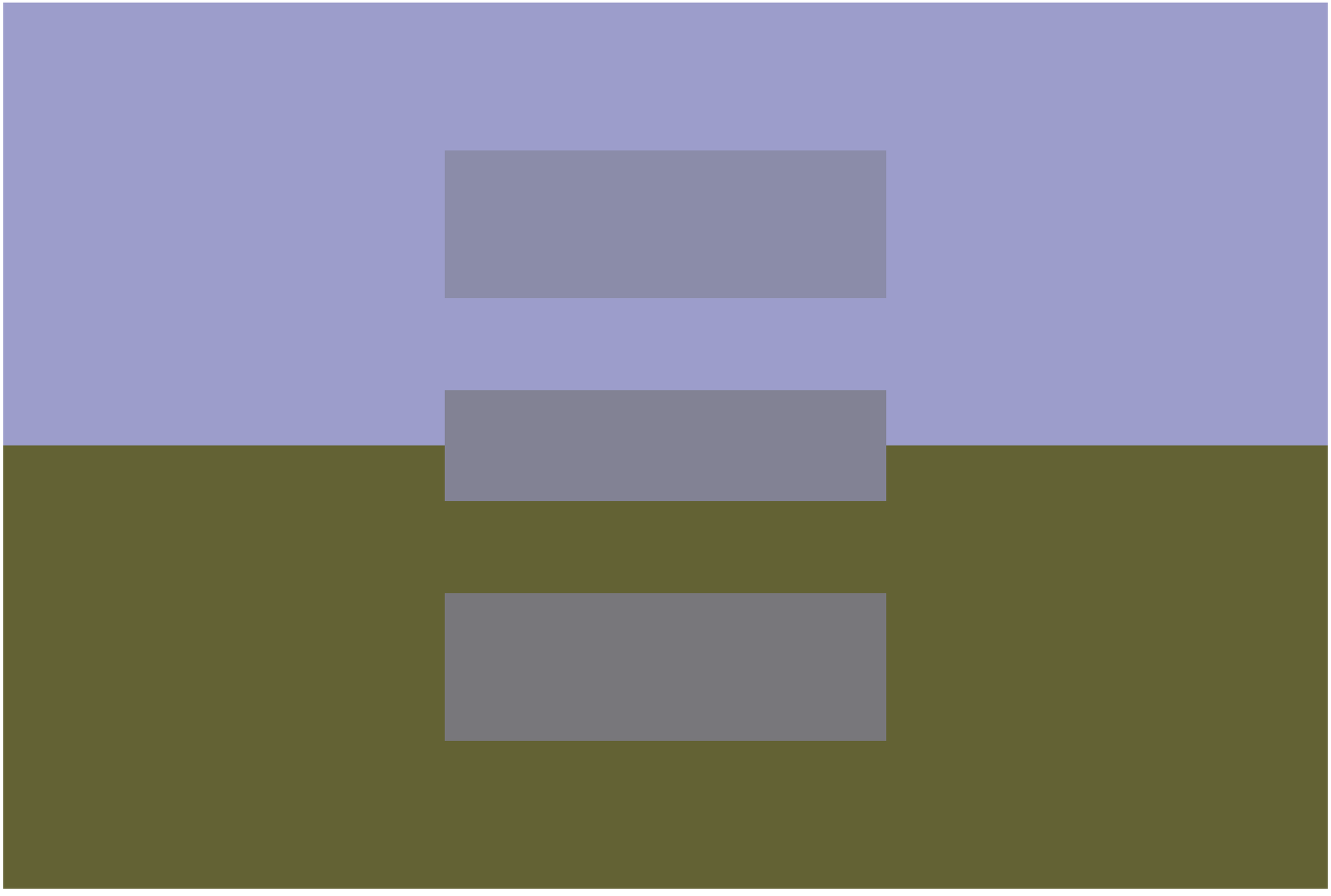
```
pushMatrix();

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: 5 COLORS INTO 3

0xFF54A15C 0xFFAB5EA3 0xFF768E77 0xFF868384 0xFF94778F

0xFF4B1EB5 0xFFB4E14A 0xFF77829A 0xFF89A08A 0xFF99B977

0xFF9C9DCB 0xFF636234 0xFF8B8CA9 0xFF828294 0xFF78777B



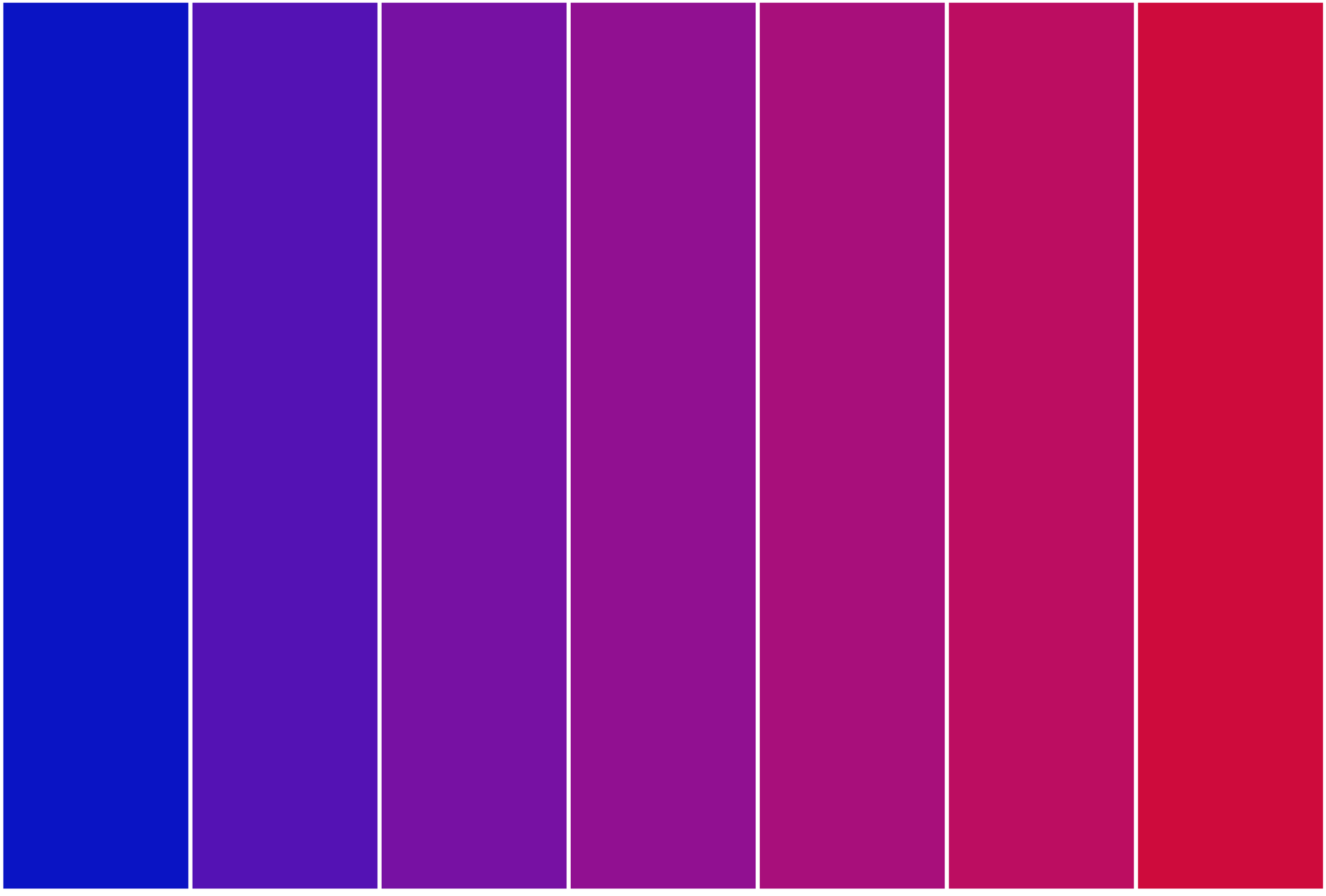
```
// recipe for making 5 colors look like 3
// prepare the first and second colors
SecureRandom random = new SecureRandom();

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();
```

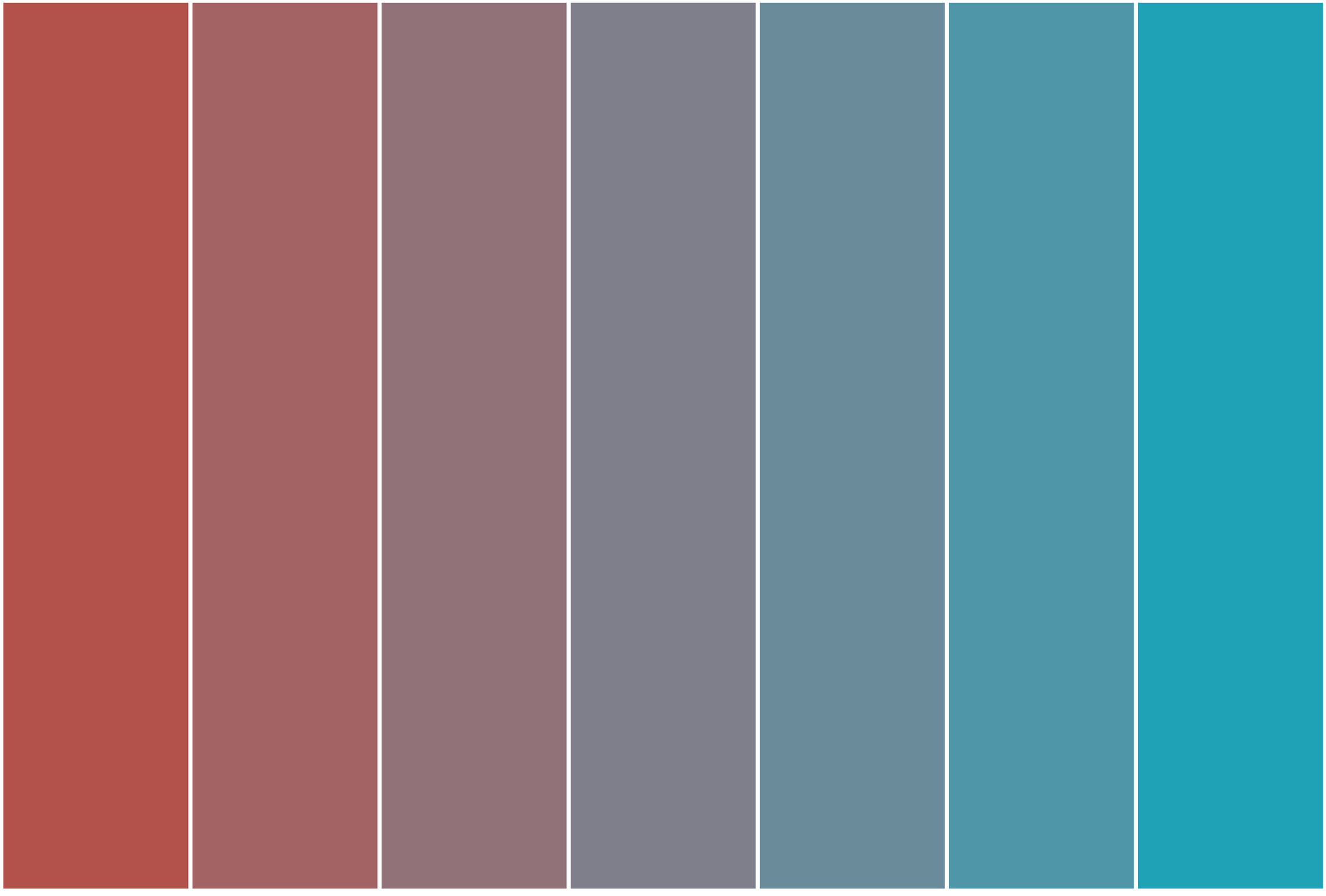
```
fill(mixed2);
rect((pgwidth)/3,  (pgheight*2)/3, pgwidth/3, pgheight/6);

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

popMatrix();
```

CHAPTER 04: COLOR MODULATION

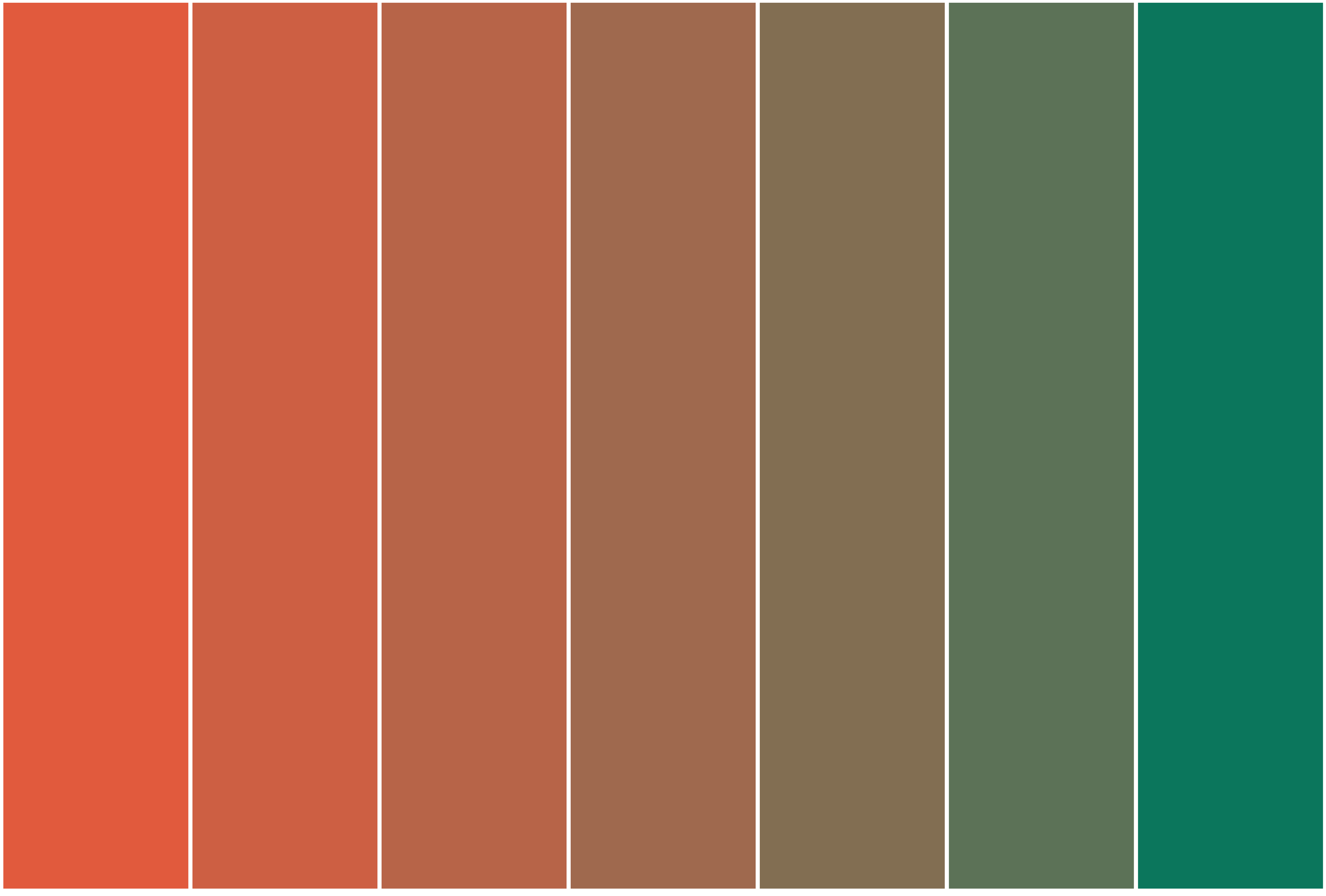
0xFF0A14C4 0xFF5412B4 0xFF7711A3 0xFF911091 0xFFA80F7B 0xFFBC0D61 0xFFCE0B3C



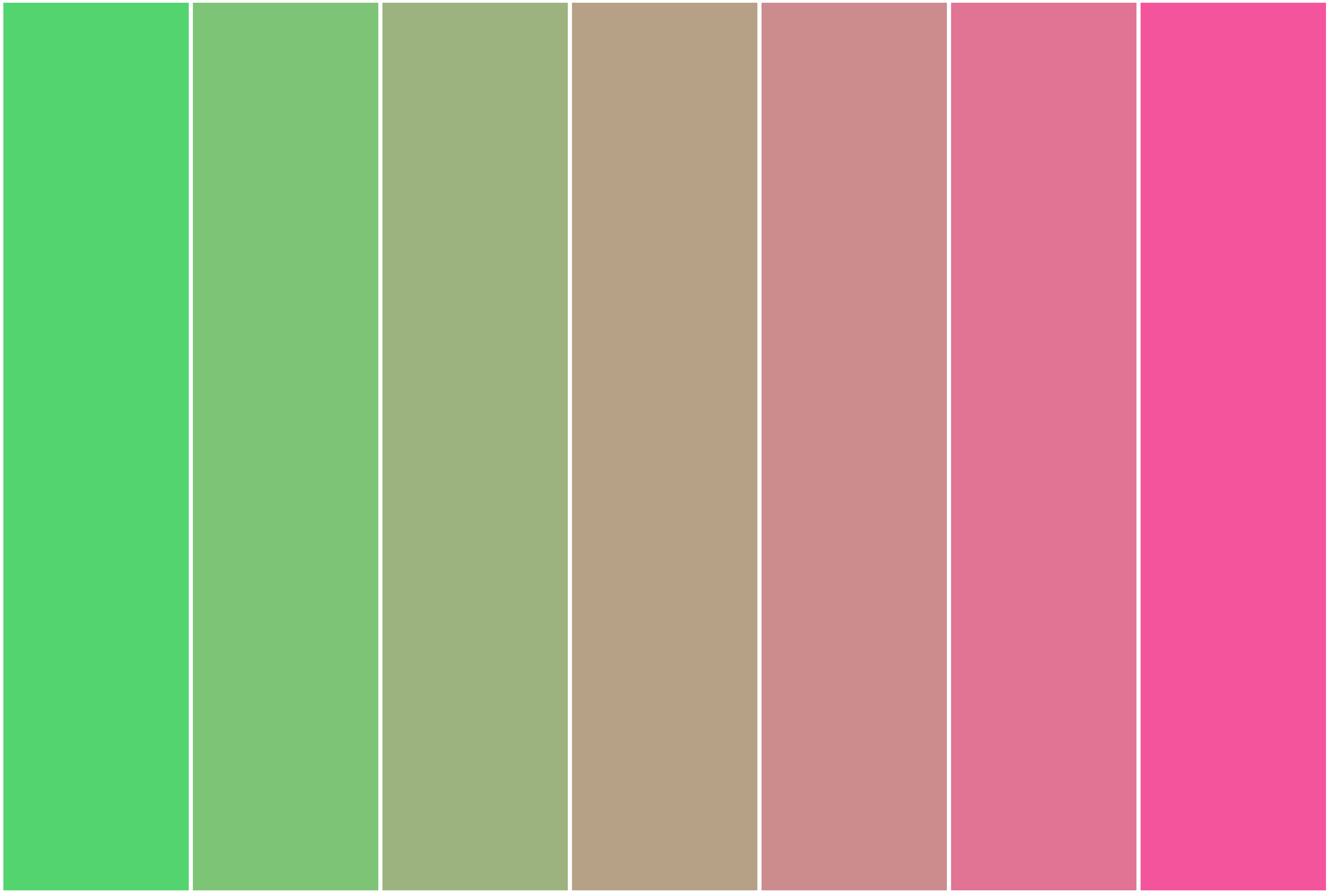
0xFFB2524B 0xFFA36365 0xFF927279 0xFF7F7F8B 0xFF6A8B9A 0xFF4E96A8 0xFF1FA1B6



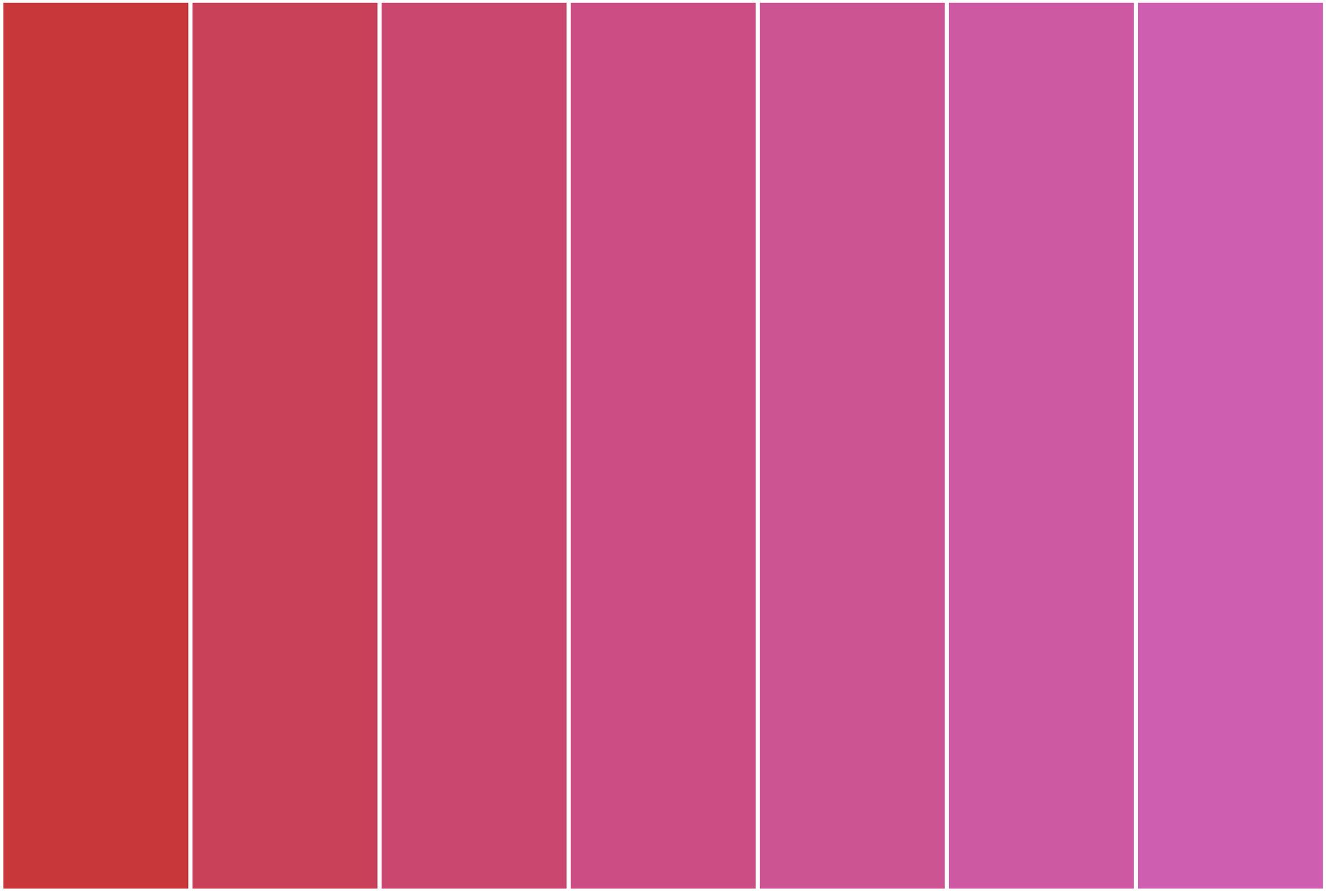
0xFFE15A3D 0xFFCD5F43 0xFFB76448 0xFF9F694E 0xFF826E52 0xFF5C7257 0xFF0B765C



0xFF53D46F 0xFF7DC477 0xFF9CB37F 0xFFB6A187 0xFFCC8C8E 0xFFE17495 0xFFF4549C



0xFFC8383A 0xFFC94059 0xFFCA4770 0xFFCB4D83 0xFFCC5493 0xFFCD59A2 0xFFCE5FB0

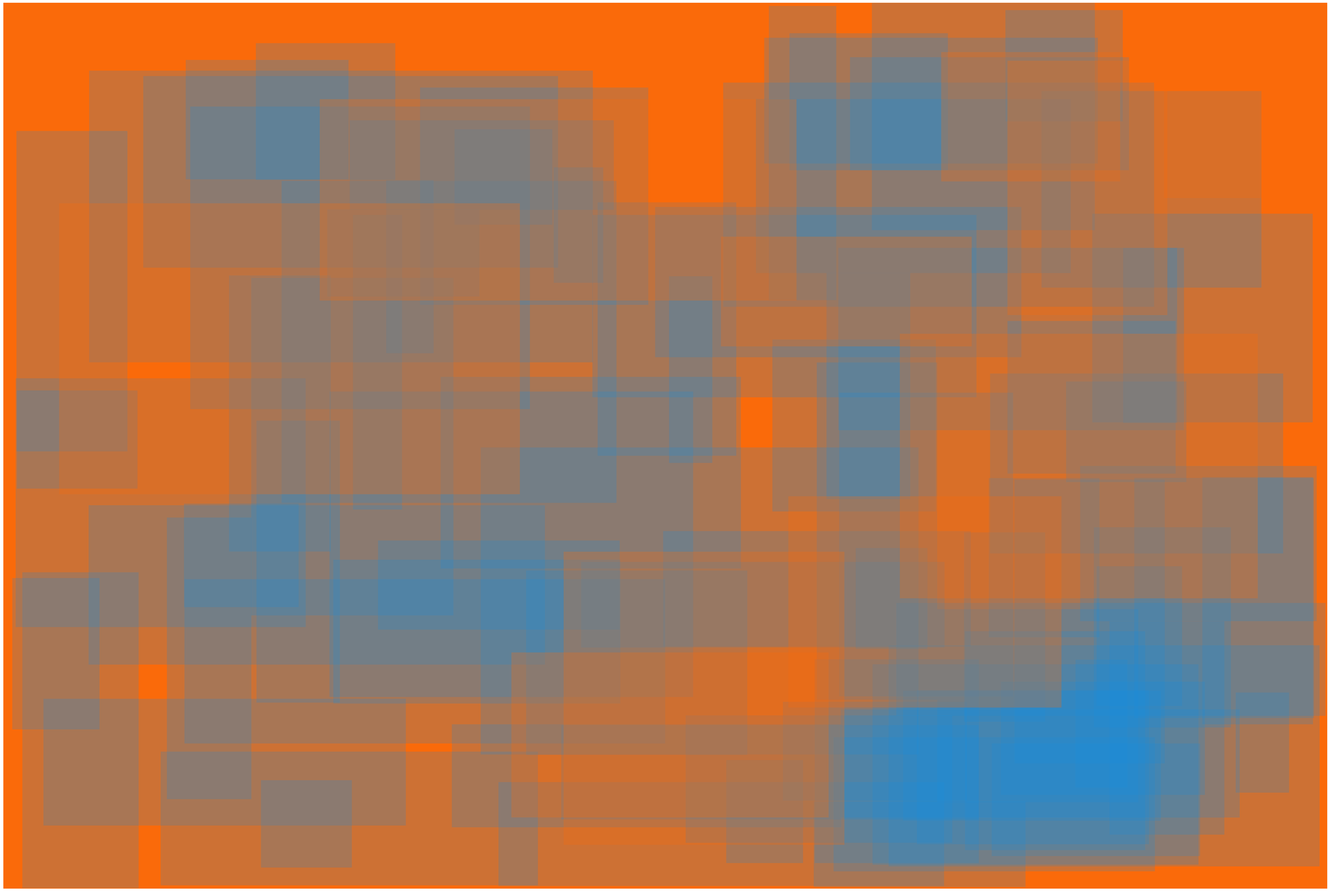


```
// recipe for color modulation

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();
```

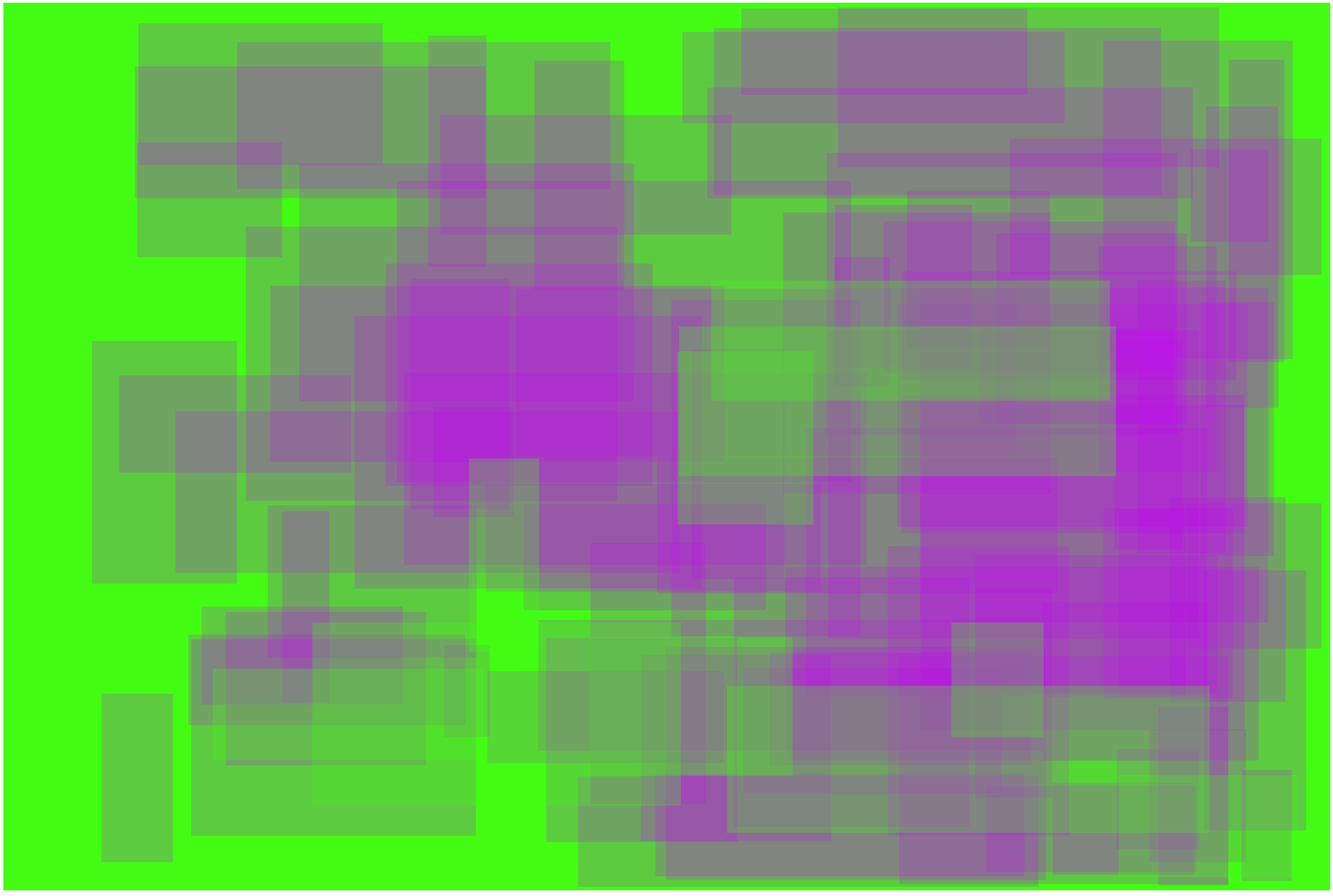
```
for(int i = 0; i < numsteps; i++) {
  // get a weighted average of the red, green, blue channels
  float mixedred = sqrt((sq(red(col1))*c1_weight +sq(red(col2))*c2_weight));
  float mixedgreen = sqrt((sq(green(col1))*c1_weight +sq(green(col2))*c2_weight));
  float mixedblue = sqrt((sq(blue(col1))*c1_weight +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 05: BRIDGING COLORS

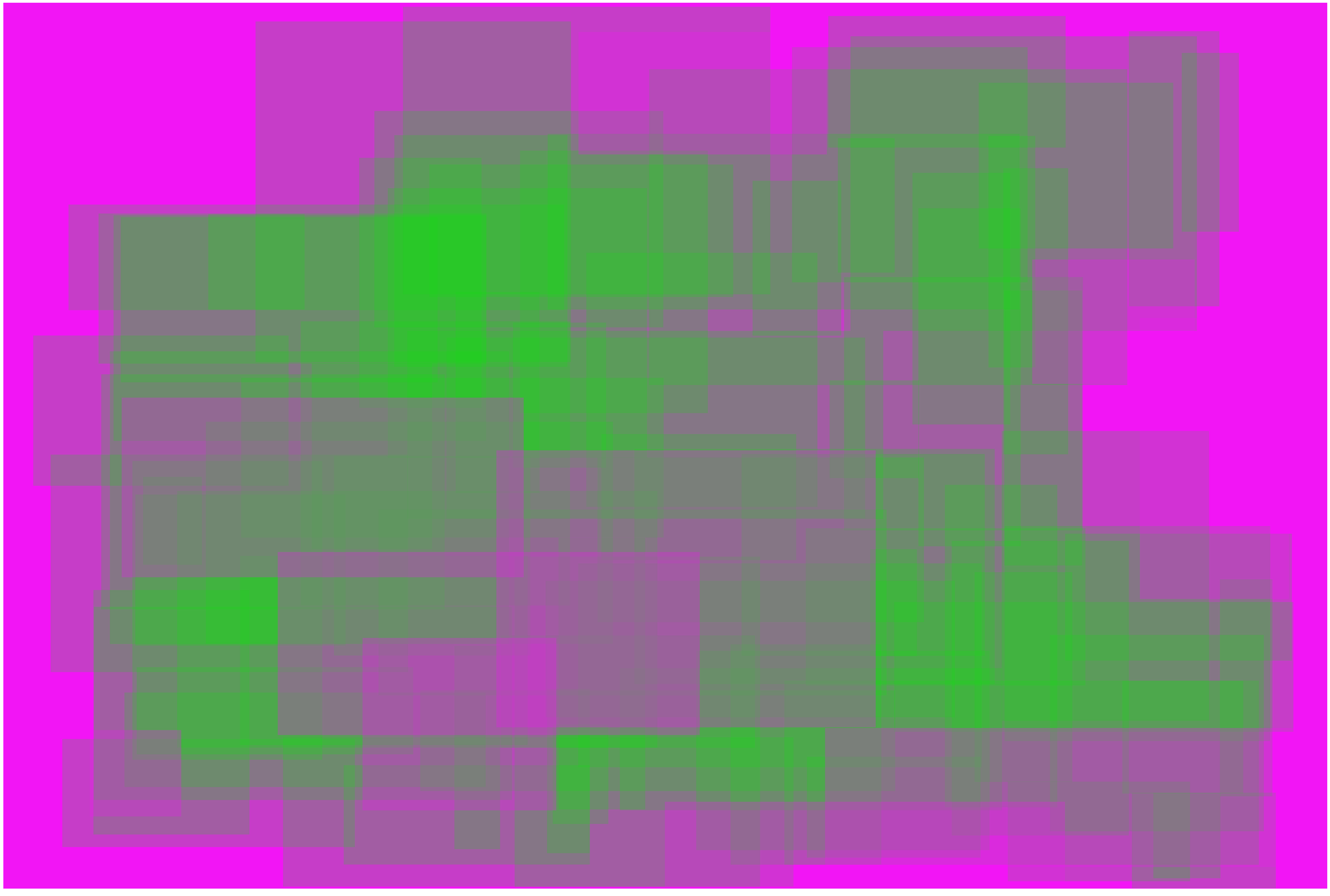
0xFF158FE1 0xFFFA6A0A



0xFFC507F5 0xFF43FC14



0xFF13E110 0xFFF215F5

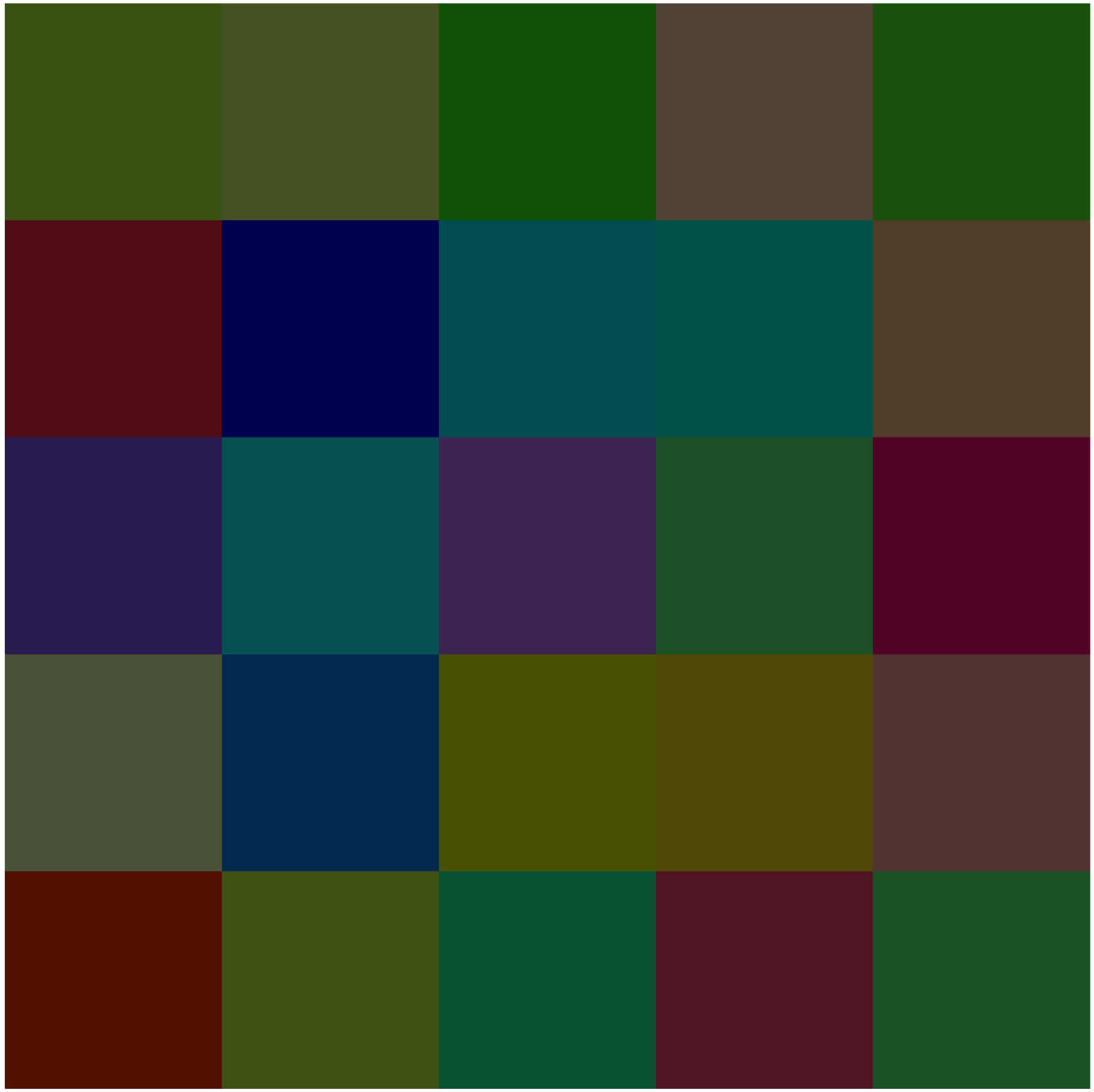


```
// recipe for using bridging colors

// 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);
```

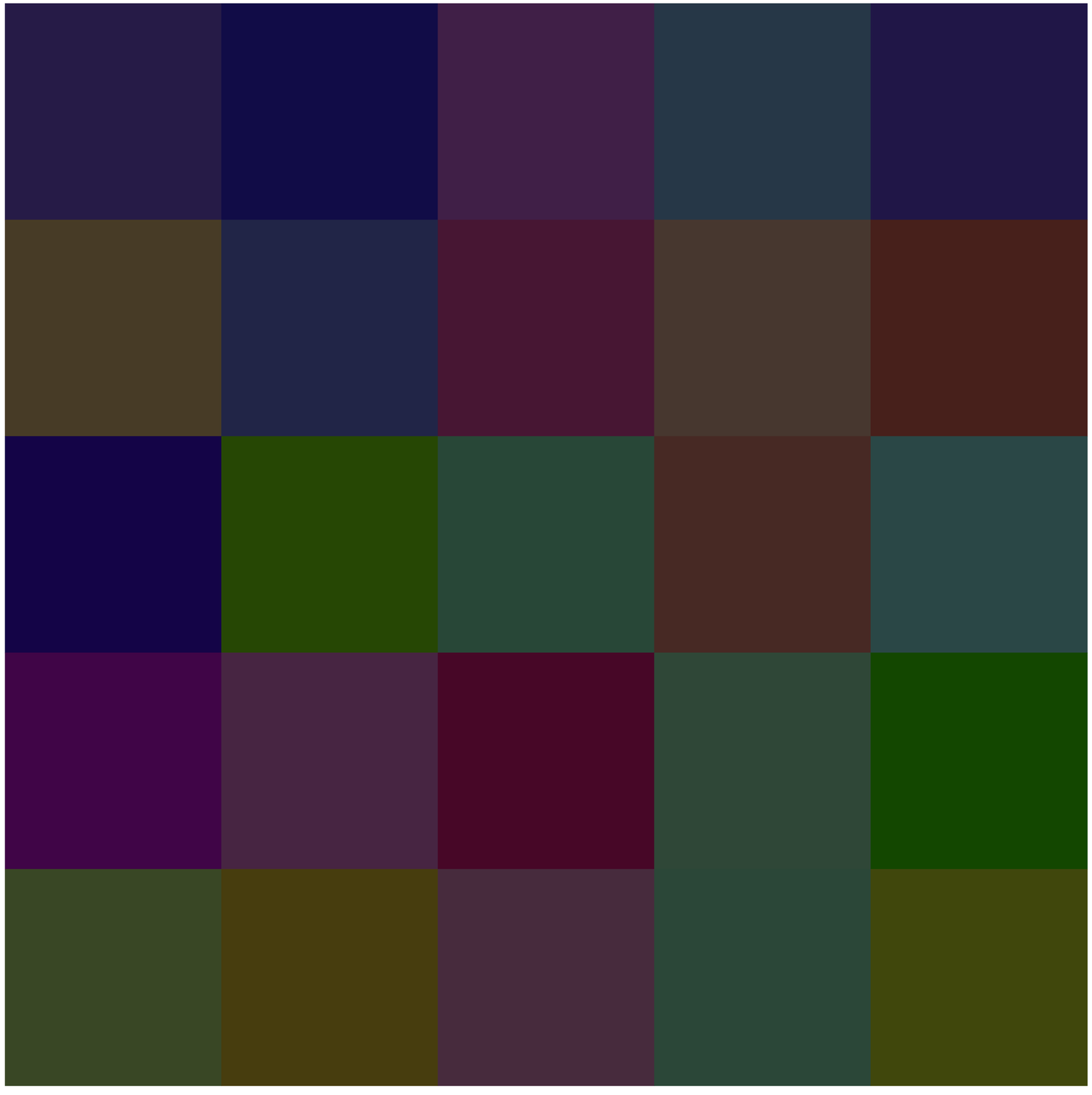
```
    float recth = random(200, min(800, pgheight-posY));

    // 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 06: SAME VALUE STUDIES (AFTER JOHANNES ITTEN)

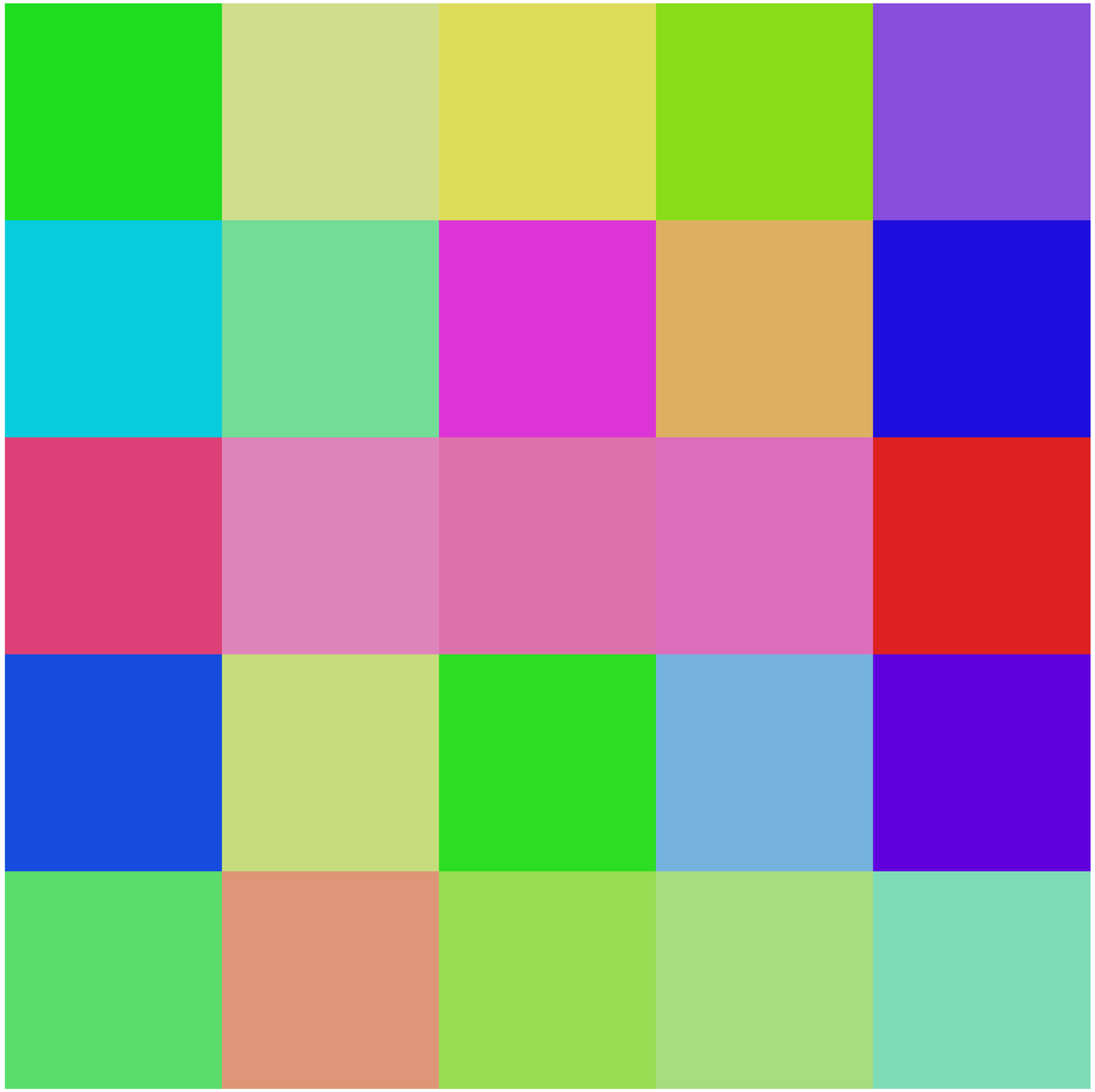
value: 32



value: 28



value: 87



```
// 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:
Sara Cwynar

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

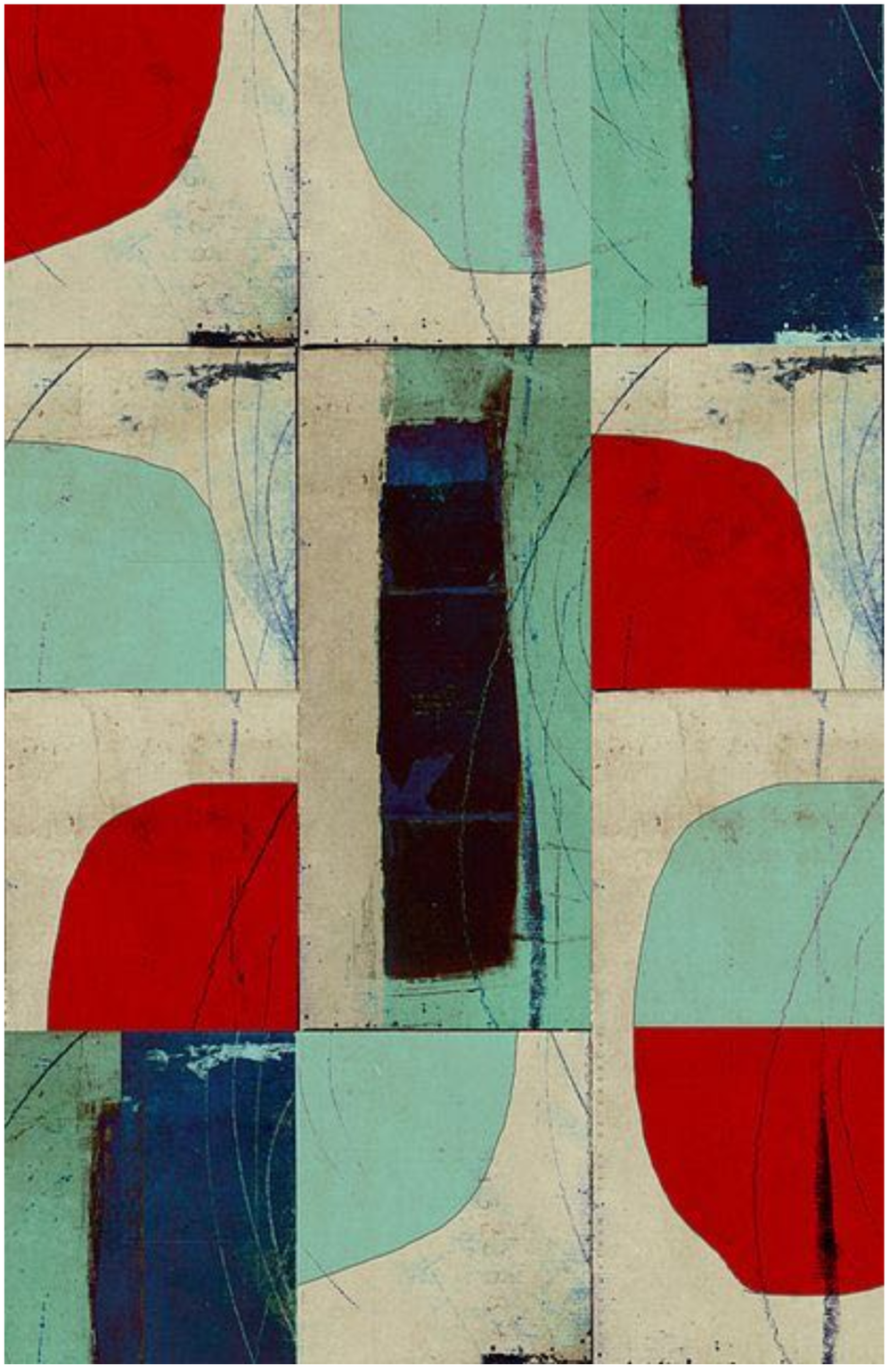
artist:
Sean Soong

0xFF0F1819
0xFF2E3B38
0xFF5E4A3A
0xFF4E6F6D
0xFF888D7E
0xFFBDCCBD
0xFF81583C
0xFF203C3D
0xFFA19C86

artist:
Sara Cwynar

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

artist:
Linda Vachon



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

artist:
Tatsuyuki Tanaka

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

artist:
Jamini Roy

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

artist:
Alphonse Mucha

0xFFEAC6A9
0xFFCE996F
0xFFB16E44
0xFF6F2A0D
0xFF40140A
0xFE1A0806
0xFF933812
0xFFB55527
0xFFD17C4A
0xFFE79469

artist:
Cy Twombly

0xFFD5D2CC

0xFF9E766B
0xFF784B45

0xFFD9DAD4

0xFF4F3936

0xFF352826

0xFF1A1313
0xFFC86C7F
0xFFA83640

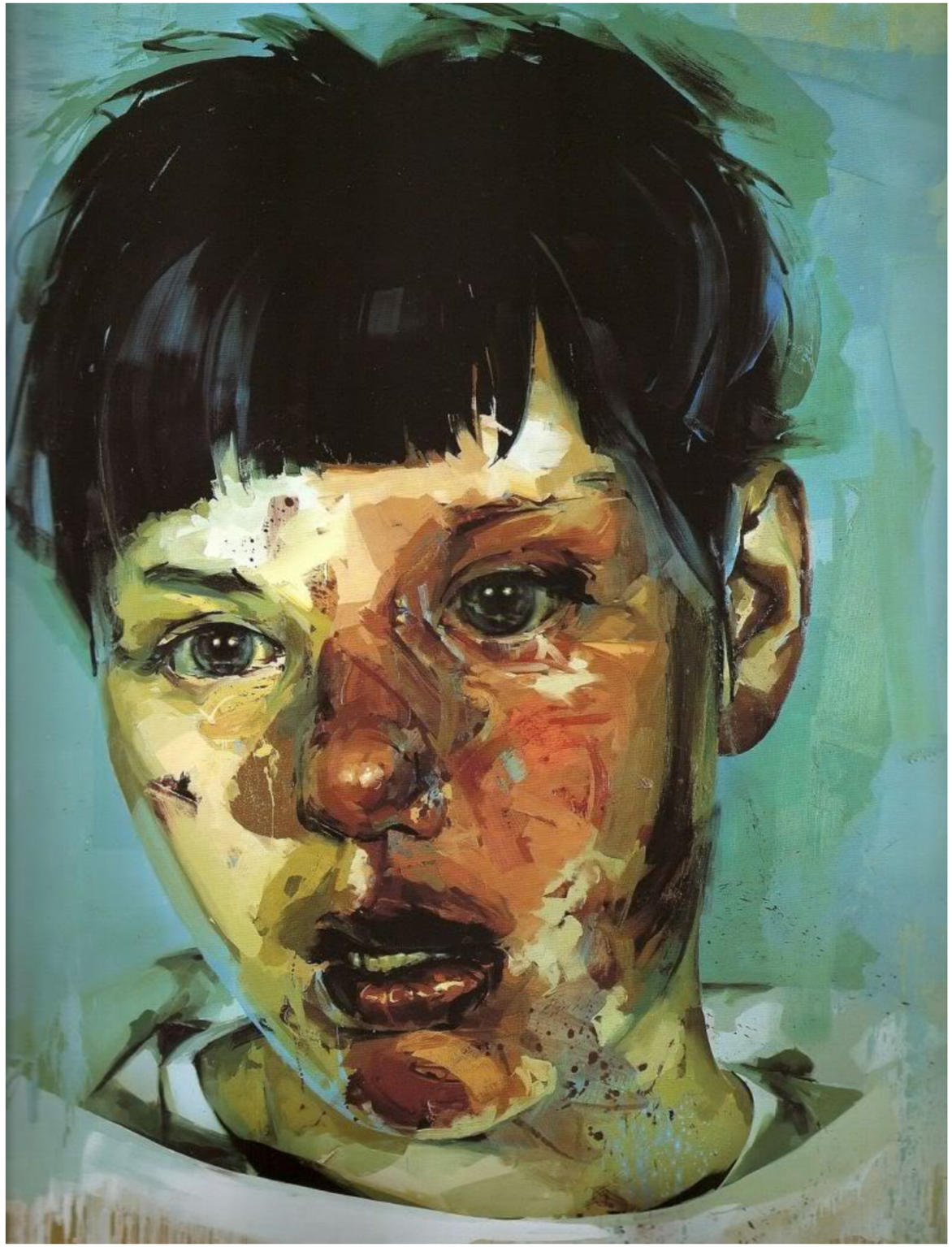
0xFF662527

0xFF8F2627

artist:
Shintaro Ohata

0xFF9F9893
0xFFD7D3CA
0xFF867C79
0xFFE7E2D8
0xFF5D6AA3
0xFF3A4083
0xFF314DA4
0xFF212B6D
0xFF859FD7
0xFFA1CAF4
0xFF6E5D48

artist:
Jenny Saville



0xFF546B5D
0xFF242019
0xFF6F9385
0xFF8CA488
0xFF191710
0xFFA77F4C
0xFFCBAE72
0xFFE9DDAA
0xFF805937
0xFF5E402C
0xFFBB7040

```
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();

}
```