

THE
COLORIST'S
COOKBOOK

**"COLORS, LIKE FEATURES, FOLLOW THE
CHANGES OF THE EMOTIONS."**

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***ed 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

0xFFB6A1A0

0xFF0068D1


0xFF8087BA



0xFF5B0CDB

0xFF95CB50


0xFF7B8FA4



0xFF400FD3

0xFF558AB6


0xFF4B62C5



0xFF8AB311

0xFFFF5718

0xFFCD8C14



```

// 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

0xFFCEF124

0xFF7B14EF


0xFFA9AAAA



0xFFE8A56

0xFF02F48D


0xFFA5C674



0xFF195443

0xFFFF0EDF7


0xFFAAB1B4



0xFFA3624C

0xFF058BB1

0xFF737888



```

// 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

0xFFA6B0ED


0xFF594F12

0xFF9197C2

0xFF777787



0xFFA17AC3 0xFF5E853C 0xFF8E7DA3 0xFF778179



0xFFB1B171

0xFF4E4E8E

0xFF97977A

0xFF767685



```

// 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


0xFF297E28

0xFFD681D7

0xFF7E7E7E

0xFF9A7F9A

0xFFB17FB1



0xFF6AC07F

0xFF953F80

0xFF79A17F

0xFF818E7F

0xFF88787F



0xFF78157A

0xFF87EA85

0xFF7C867D

0xFF7FA67F

0xFF81C081



```

// 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

0xFF3E4563

0xFF45435B


0xFF4B4253

0xFF51414A

0xFF563F41

0xFF5C3E35

0xFF613C25



0xFF0E7105

0xFF246747


0xFF315C64

0xFF3B507B

0xFF44428E

0xFF4B2F9E

0xFF530CAE



0xFF44FE74

0xFF66FA88


0xFF80F699

0xFF96F2A9

0xFFA8EEB8

0xFFB9EAC5

0xFFC9E5D2



0xFFC3B9BE

0xFFB6B8B9


0xFFA8B7B5

0xFF9AB6B1

0xFF89B5AD

0xFF77B4A8

0xFF60B2A3



0xFF226832

0xFF658C3B


0xFF8BA944

0xFFA8C24B

0xFFC1D752

0xFFD7EB59

0xFFEFCFE5F



```
// 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


0xFFCB10F9 0xFF38CD13



0xFF06CE06 0xFFEA06EA



0xFF0CE2D4 0xFFE51220



```
// 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();

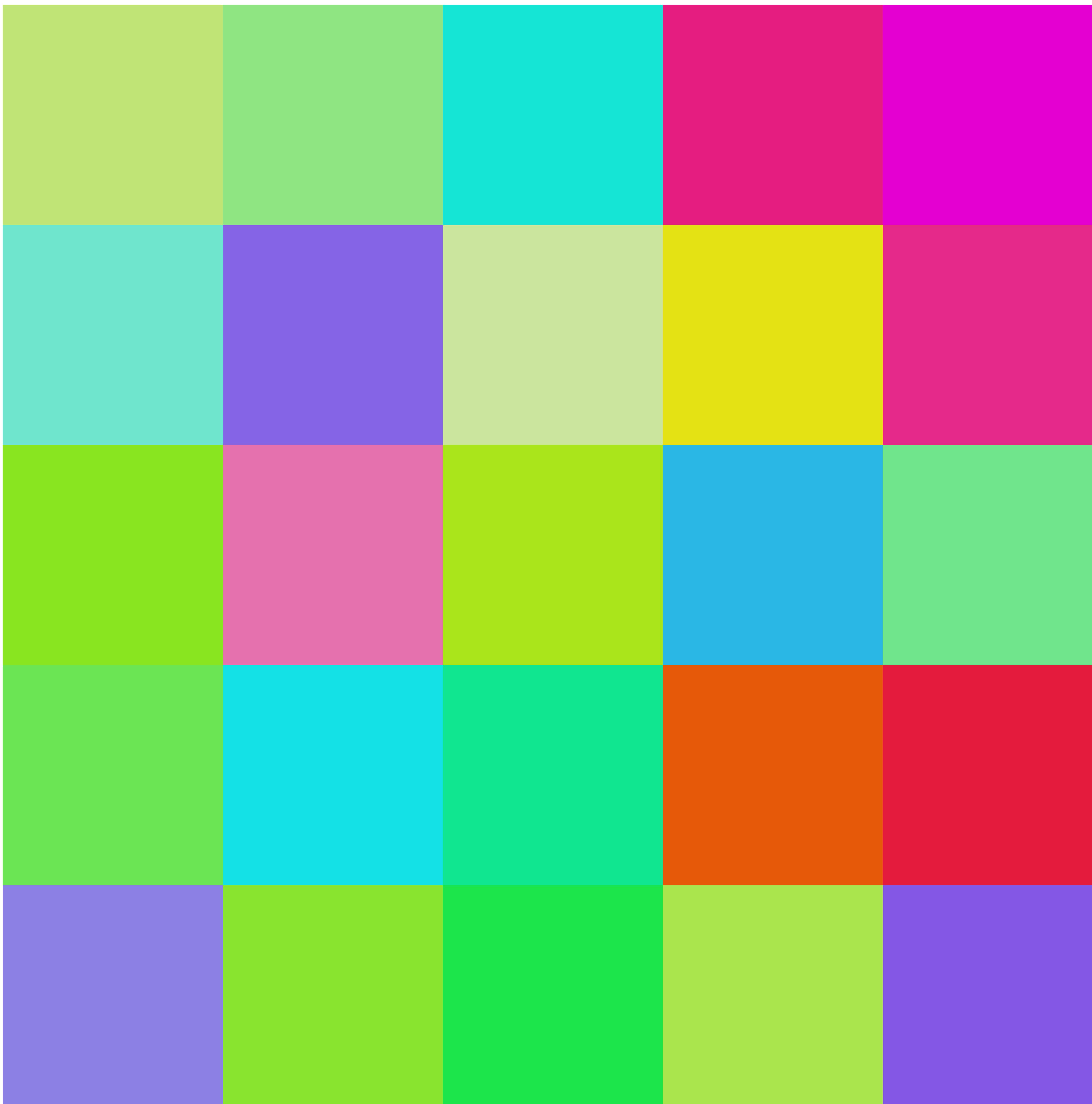
// glaze with the background color, which is opposite from the foreground color
// add a touch of randomness
color bg = color((hue(col1)+180)%360, random(90,100) , random(80,100));
fill(bg);
rect(0, 0, pgwidth,pgheight);

// 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);
```

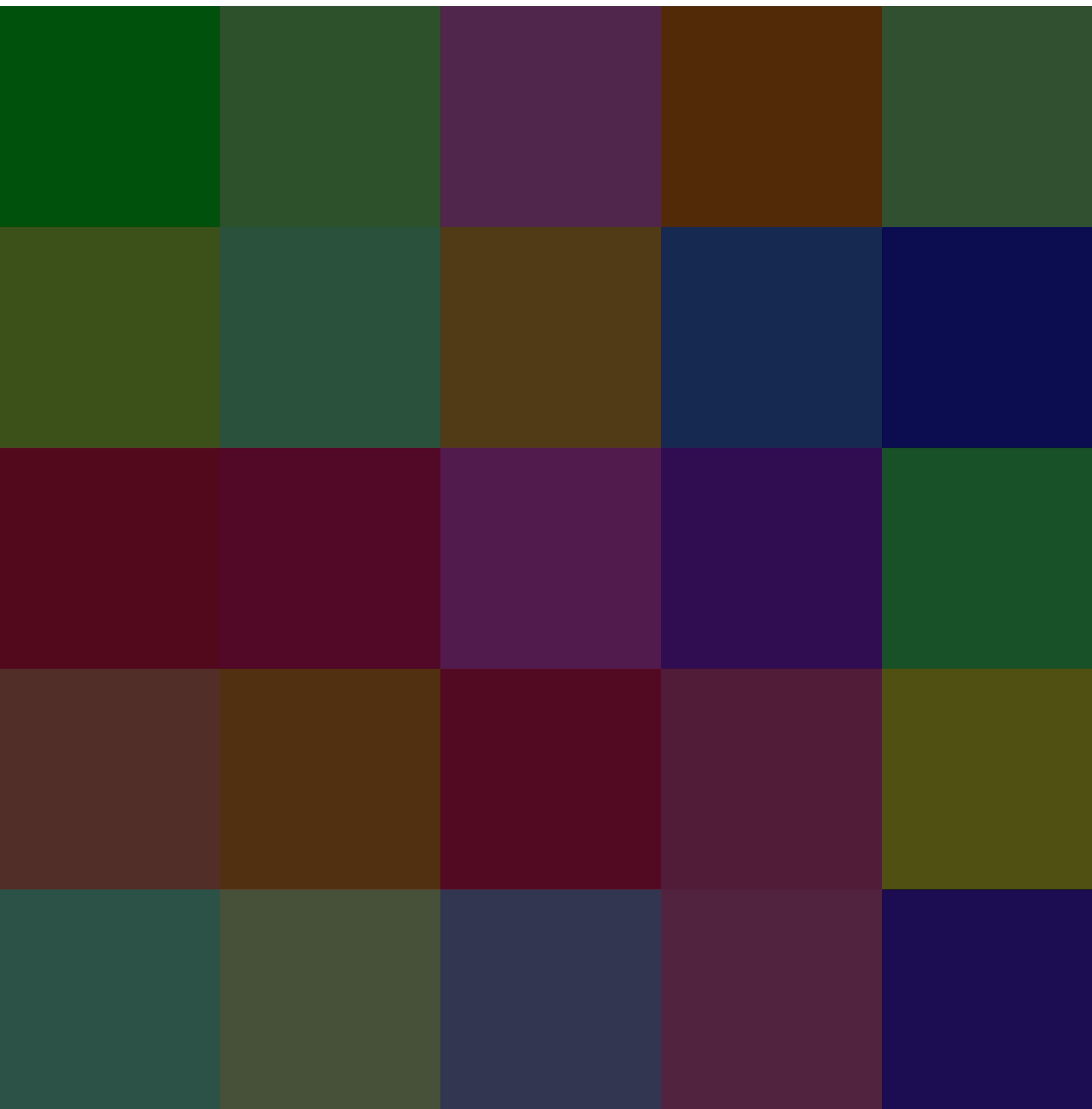
```
float recth = random(200, min(800, pgheight-posY)) ;  
  
// the slices can overlap with each other,  
// but they must be thin  
// like prosciutto  
fill(col1, 50);  
rect(posX, posY, rectw, recth);  
  
}  
  
// for balance, top with a few thin slices of the background color  
for(int i = 0; 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)

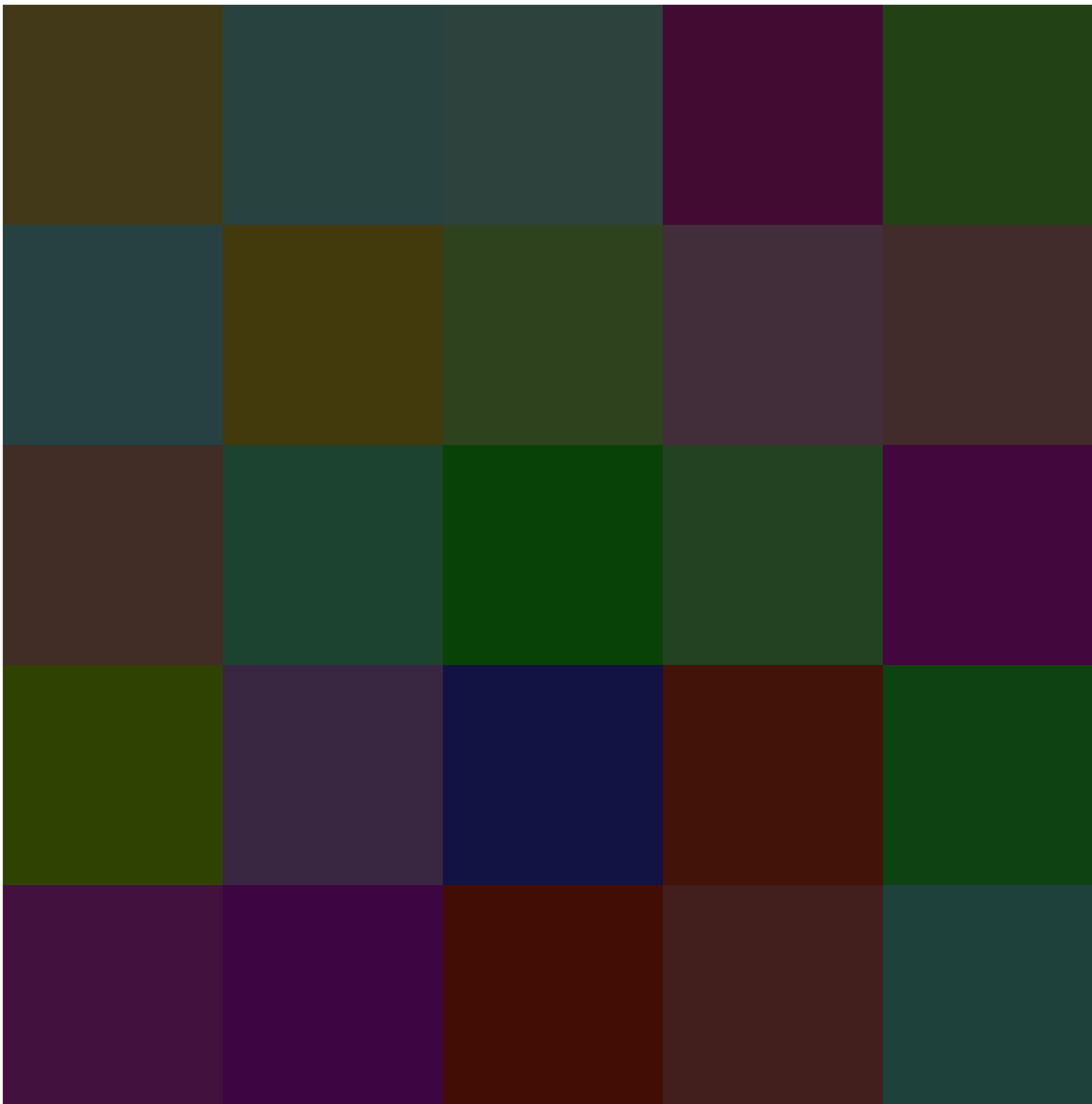
value: 90



value: 32



value: 26



```

// 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();

```

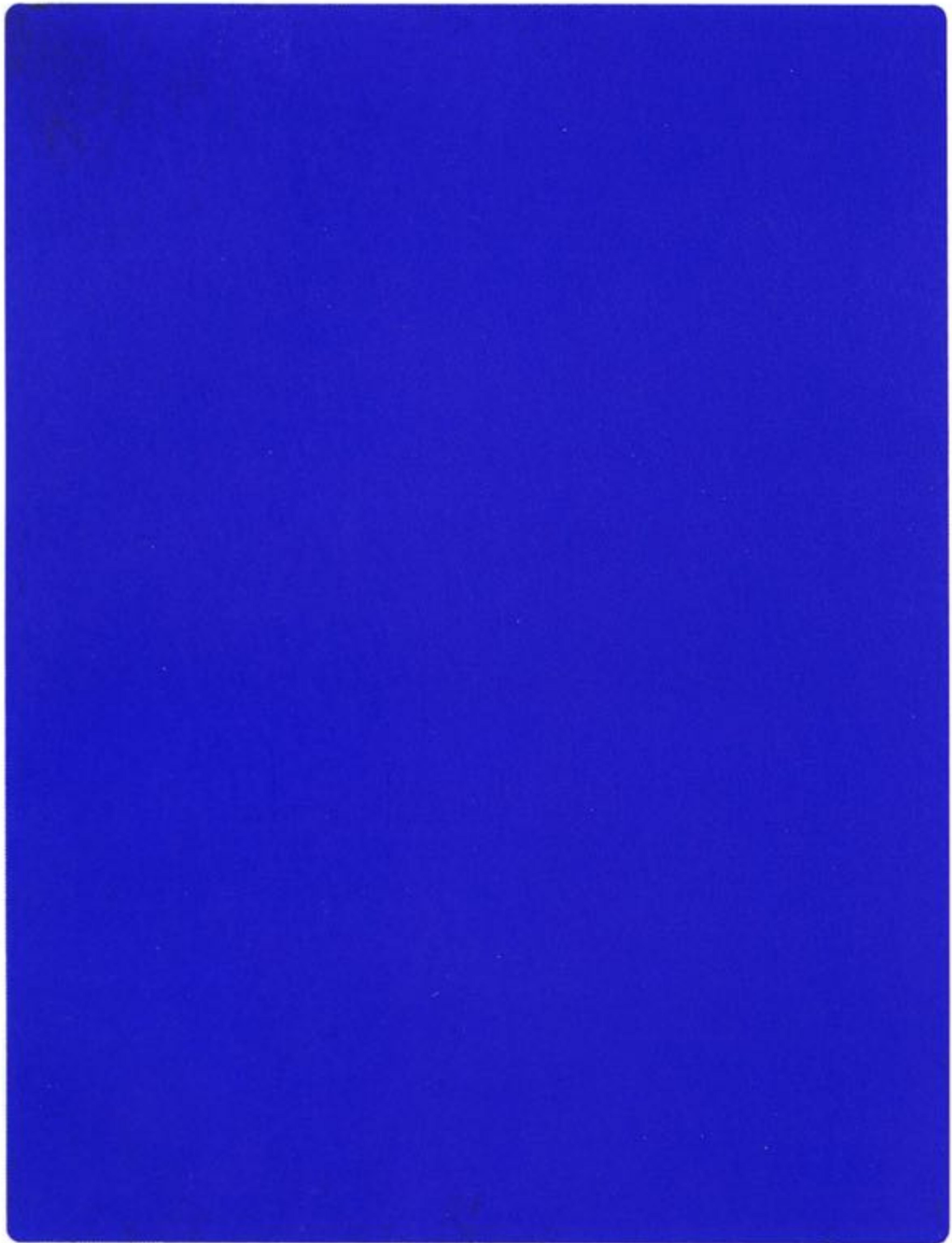
APPENDIX: PALETTES

artist:
Paul Klee



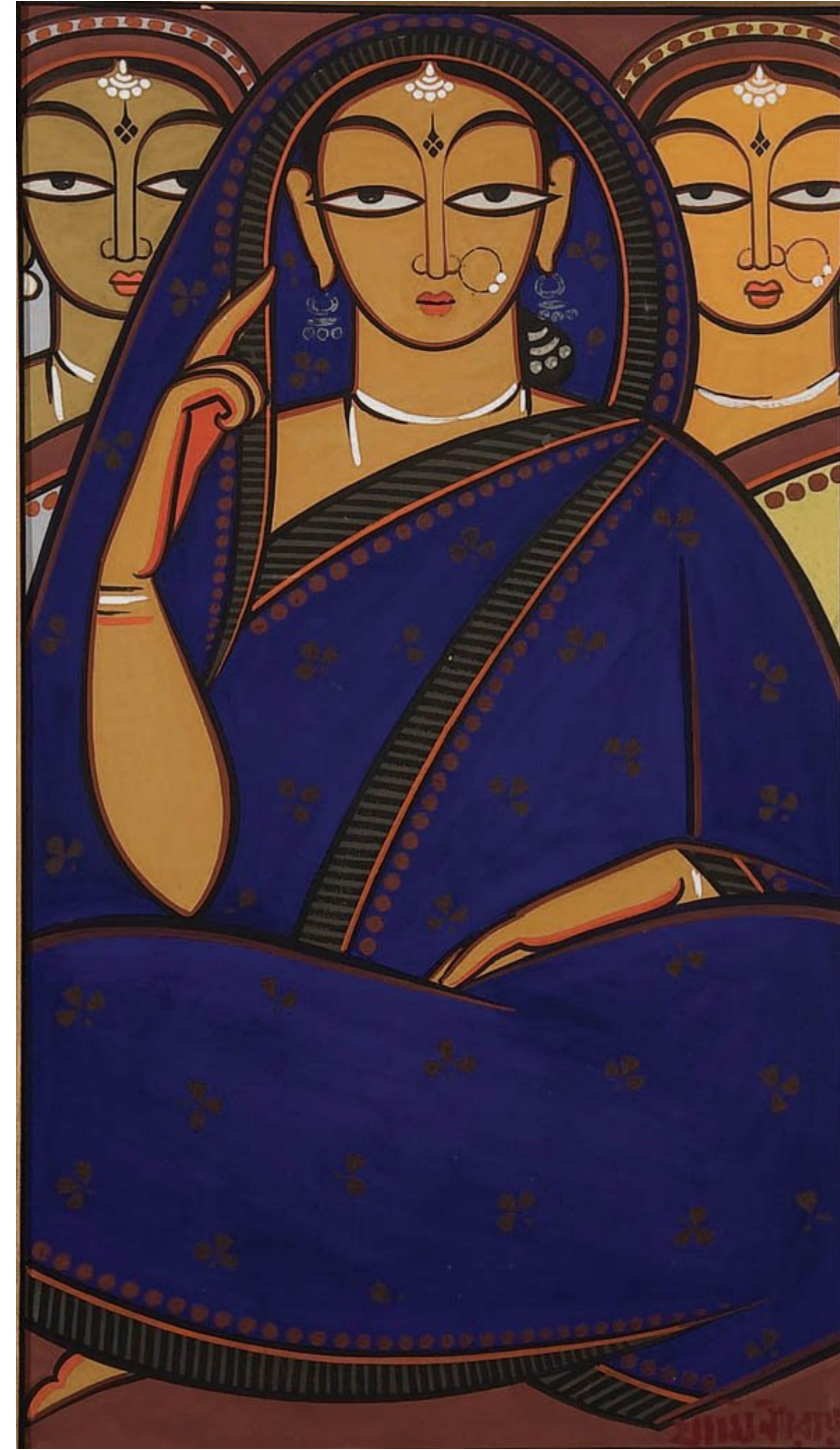
0xFF724F33
0xFF53442B
0xFF383923
0xFFAA241F
0xFFC33919
0xFFCE8104
0xFF8E857A
0xFFD50E23

artist:
Yves Klein



0xFF201BC4

artist:
Jamini Roy



0xFF231D38
0xFF3A282F
0xFF693F30
0xFF936843
0xFFB28149
0xFFCB985A
0xFFBC803E
0xFFD38E43

artist:
Jenny Saville



0xFF546B5D
0xFF242019
0xFF6F9385
0xFF8CA488
0xFF191710
0xFFA77F4C
0xFFCBAAE72
0xFFE9DDAA
0xFF805937
0xFF5E402C
0xFFBB7040

artist:
Sara Cwynar



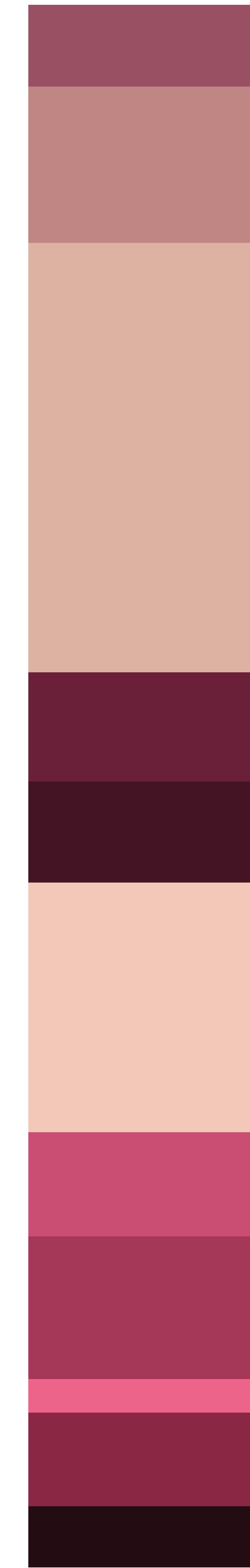
0xFFFFAC9E

0xFFDD725F

0xFFB84C3B
0xEE8D3526
0xFFED7961

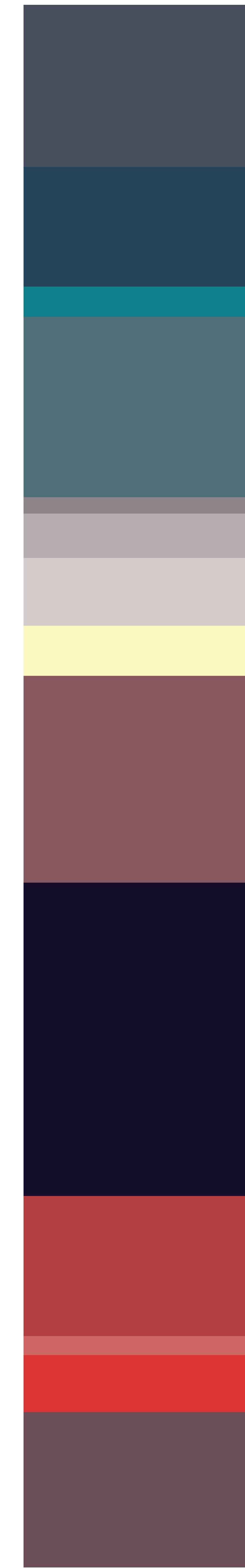
0xFFD14C3D

artist:
Shintaro Ohata



0xFF985062
0xFFBF8684
0xFFDDDB2A2
0xFF6A2039
0xFF431424
0xFFFF2C9B9
0xFFCA4E73
0xFFA53759
0xFFEC6589
0xFF8A2744
0xFF230C12

artist:
James Jean



0xFF484D5B

0xFF24445A

0xFF0D808C
0xFF507079

0xFF8D8589
0xFFB6AEEA

0xFFD3CCCC

0xFFFFAF8C0
0xFF86575D

0xFF150E2A

0xFFB23E40

0xFFCE6464
0xFFDA3434

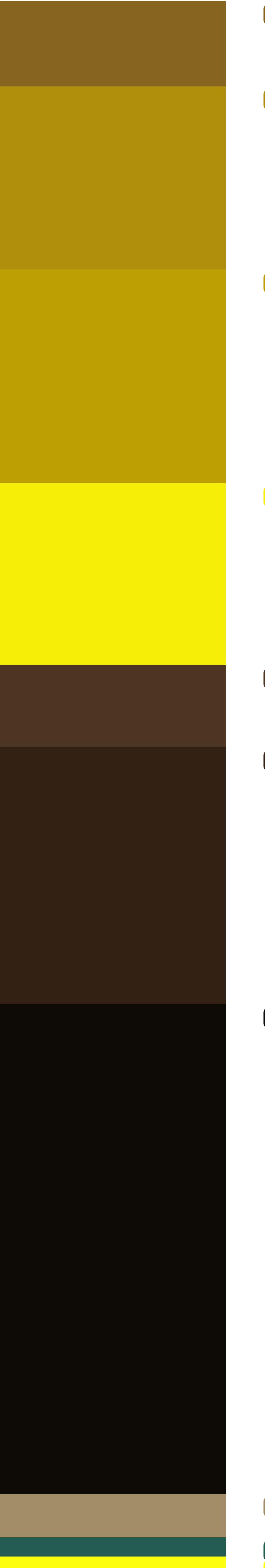
0xFF6A4D56

artist:
Deborah Zlotsky

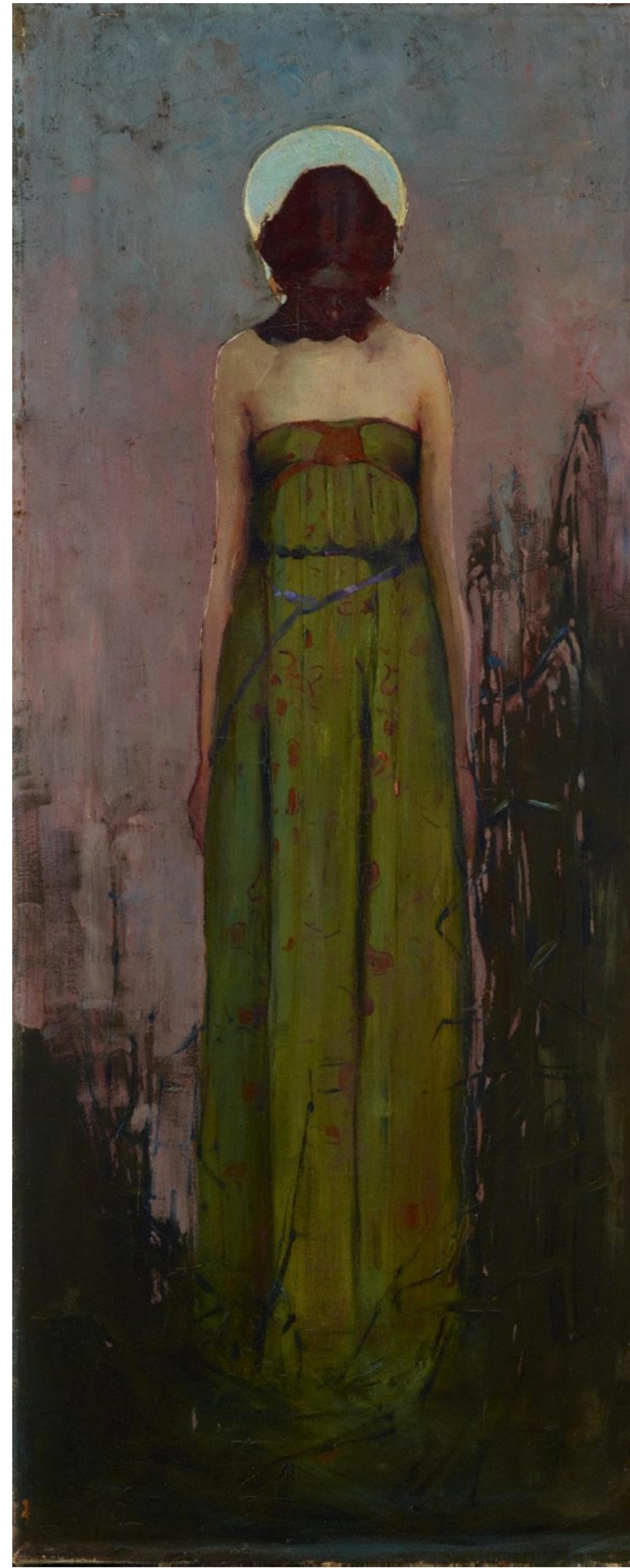


0xFF5093B3
0xFF7D98AB
0xFFA5AFAD
0xFFCC9262
0xFF1F729E
0xFFAFC1D1
0xFFE8B7D3
0xFFFF99A039
0xFFC28C14
0xFF665F43
0xFF4B7A21
0xFF2D482C
0xFF0F3958
0xFF70993B
0xFF44597C

artist:
Piotr Jablonski



artist:
Sydney Long



0xFF69605B



0xFF514E44

0xFF96876D

0xFF2F211A

```
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(palettespos, ypos, 300, rheight);

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

```

```
    }  
}  
  
popMatrix();  
  
}
```

**A SPECIAL THANKS TO CLAYTON MERRELL AND RAFAEL ABREU-CANEDO
FOR YOUR GUIDANCE AND SUPPORT.**

