

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

0xFF973257

0xFFE7DCBF


0xFFC39F94



0xFF278AA6

0xFFC1B3B6


0xFF8B9FAE



0xFF16516A

0xFFFFB2B6A


0xFFB2406A



0xFF115B97

0xFF72CA38

0xFF519C71



```
// 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

0xFFEED586

0xFF408cff


0xFFAEB4CB



0xFFDDA44

0xFFFFB689D


0xFFB1AA78



0xFFDED5B7

0xFF2FAB1D


0xFFA0C183



0xFFFF024B5

0xFF6AE5A0

0xFFB9A3AA



```

// 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

0xFF08F4A8

0xFFFF70B57

0xFF8CC892

0xFFCA8A76



0xFFA0BC4A

0xFF5F43B5

0xFF8D9F77

0xFF77789A




0xFFC8E494

0xFF371B6B

0xFFA6BB87

0xFF7A8379



```

// 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


0xFFCB8E94

0xFF34716B

0xFFA98587

0xFF948081

0xFF7B7A79



0xFF2EA886

0xFFD15779

0xFF7C9281

0xFF97857F

0xFFAD767C



0xFF965E89

0xFF69A176

0xFF887782

0xFF81837F

0xFF798E7B



```

// 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

0xFF3BAB0C

0xFF50A43D


0xFF629D56

0xFF709768

0xFF7D8F79

0xFF898887

0xFF947F94



0xFF495461

0xFF734C67


0xFF91456E

0xFFAA3C74

0xFFC0327A

0xFFD4257F

0FFE61085



0xFF35B154

0xFF3DAF60


0xFF44AE6C

0xFF4AAC76

0xFF50AB7F

0xFF55A988

0xFF5BA891



0xFF31CDBB

0xFF34BDB8


0xFF37ACB6

0xFF3A99B3

0xFF3D83B1

0xFF4069AE

0xFF4345AB



0xFF9EDFA0

0xFF94DD9F


0xFF89DC9F

0xFF7EDB9F

0xFF72DA9F

0xFF64D99F

0xFF53D79F



```
// 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


0xFF3407E7

0xFFC1EF08




0xFF72E60C

0xFF7914D1



0xFF1466F4 0xFFEA9809



```
// 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));

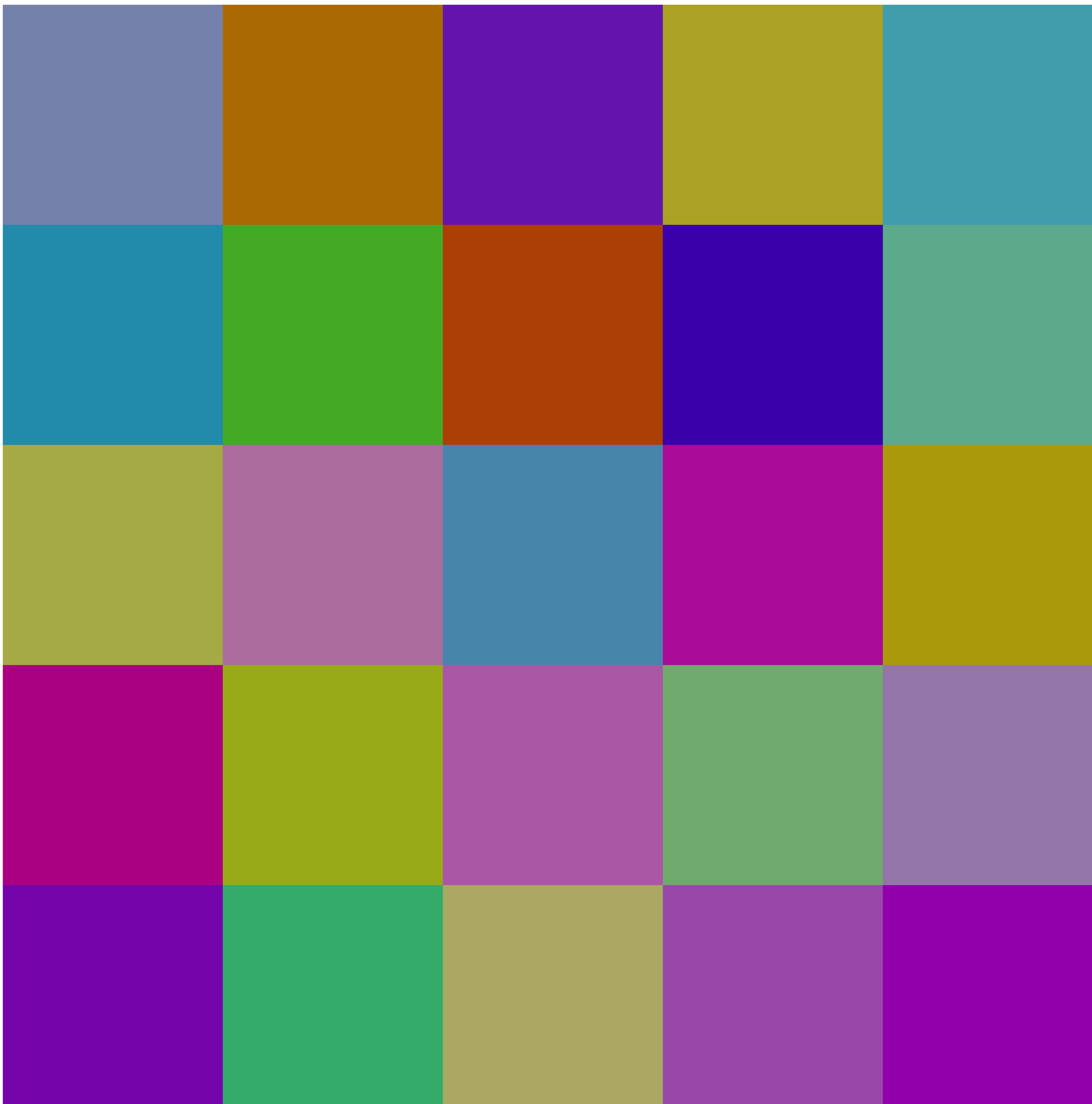
    fill(bg,70);
    rect(posX, posY, rectw, recth);

}

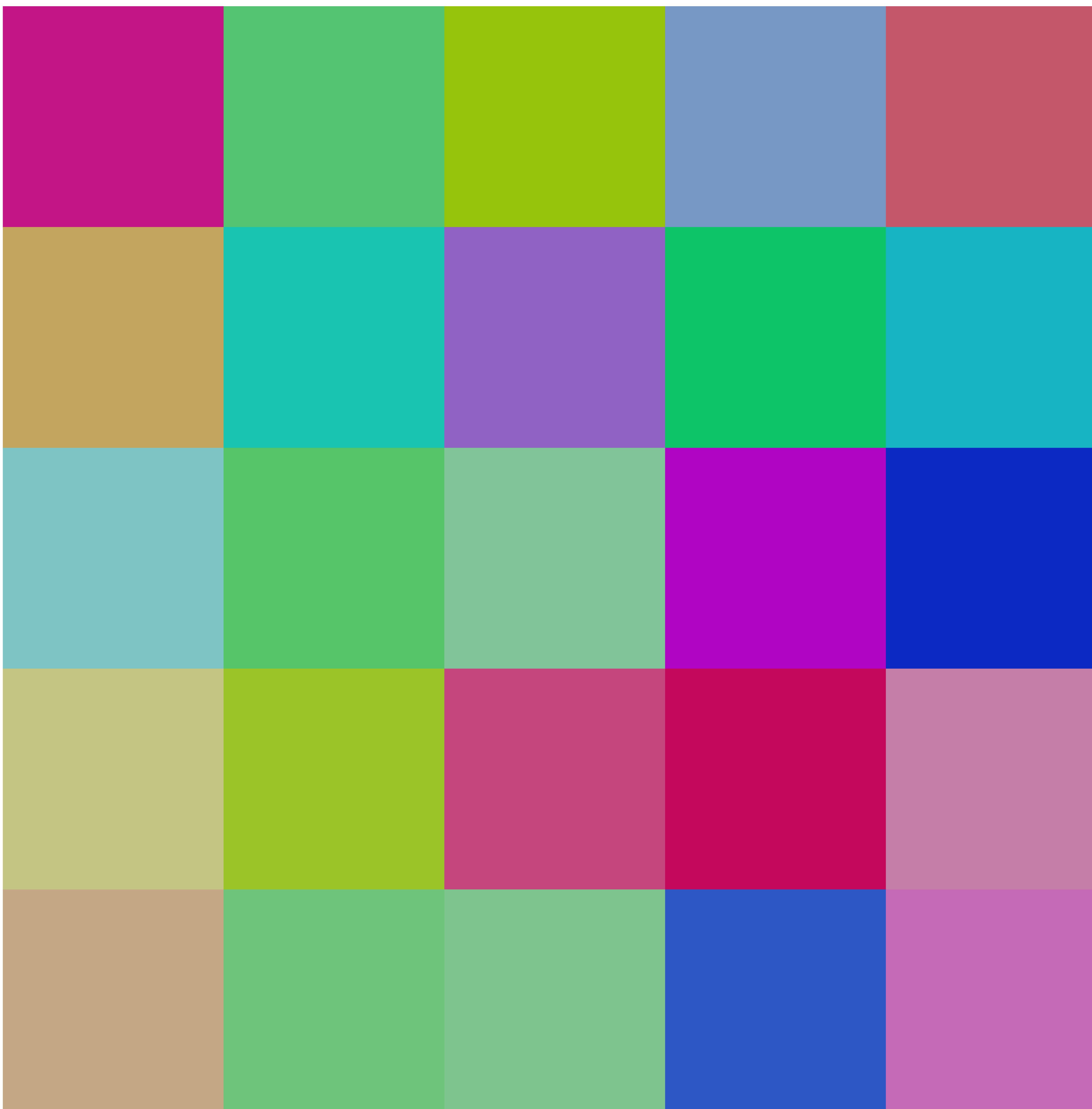
popMatrix();
colorMode(RGB);
```

CHAPTER 07: SAME VALUE STUDIES (AFTER JOHANNES ITTEN)

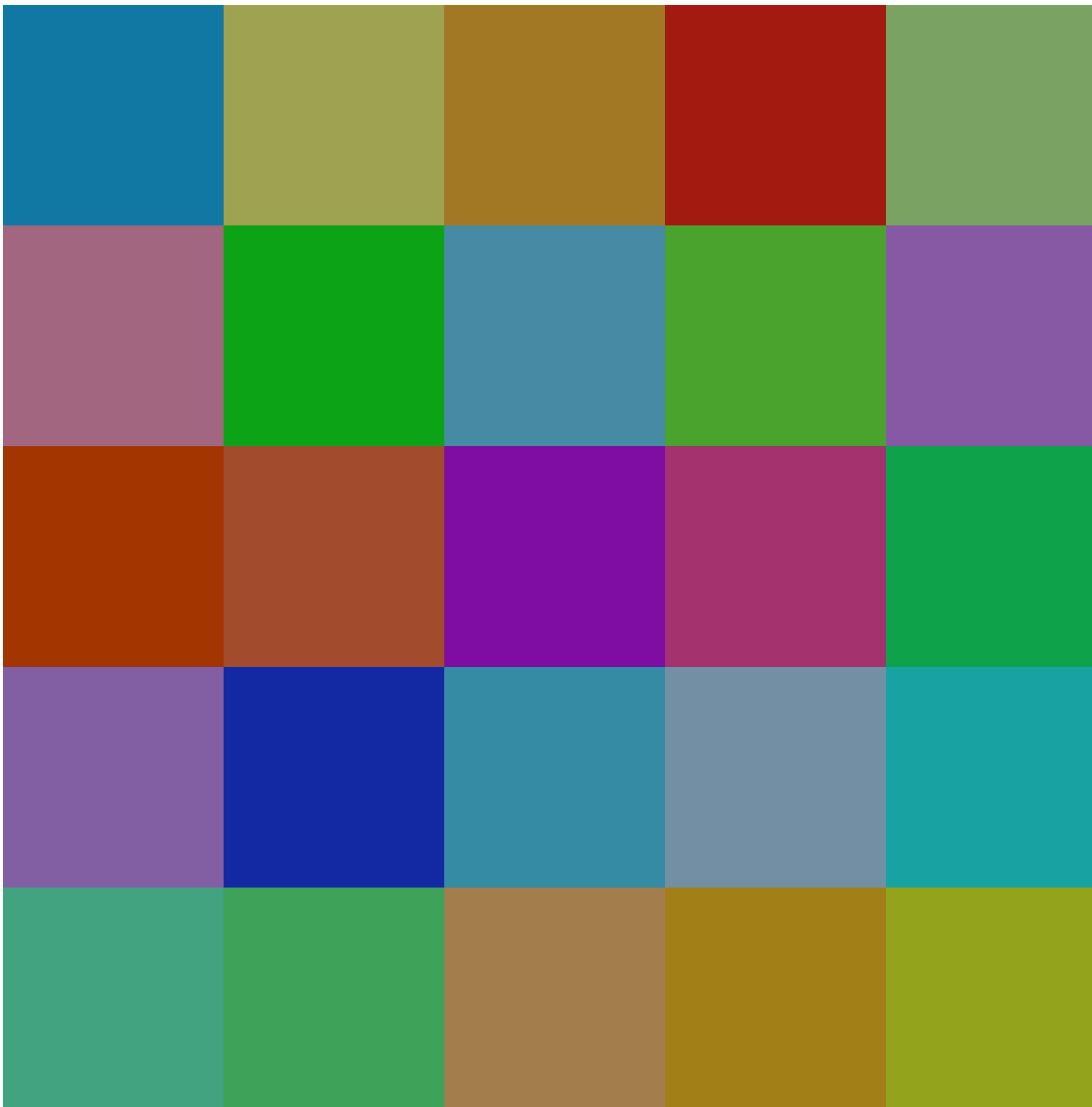
value: 67



value: 77



value: 64



```
// 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:
Tatsuyuki Tanaka



artist:
Jenny Saville



0xFF546B5D
0xFF242019
0xFF6F9385
0xFF8CA488
0xFF191710
0xFFA77F4C
0xFFCBAE72
0xFFE9DDAA
0xFF805937
0xFF5E402C
0xFFFFB7040

artist:
Cy Twombly



0xFFD5D2CC
0xFF9E766B
0xFF784B45
0xFFD9DAD4
0xFF4F3936
0xFF352826
0xFF1A1313
0xFFC86C7F
0xFFA83640
0xFF662527
0xFF8F2627

artist:
Sara Cwynar

0xFFFFAC9E



0xFFDD725F

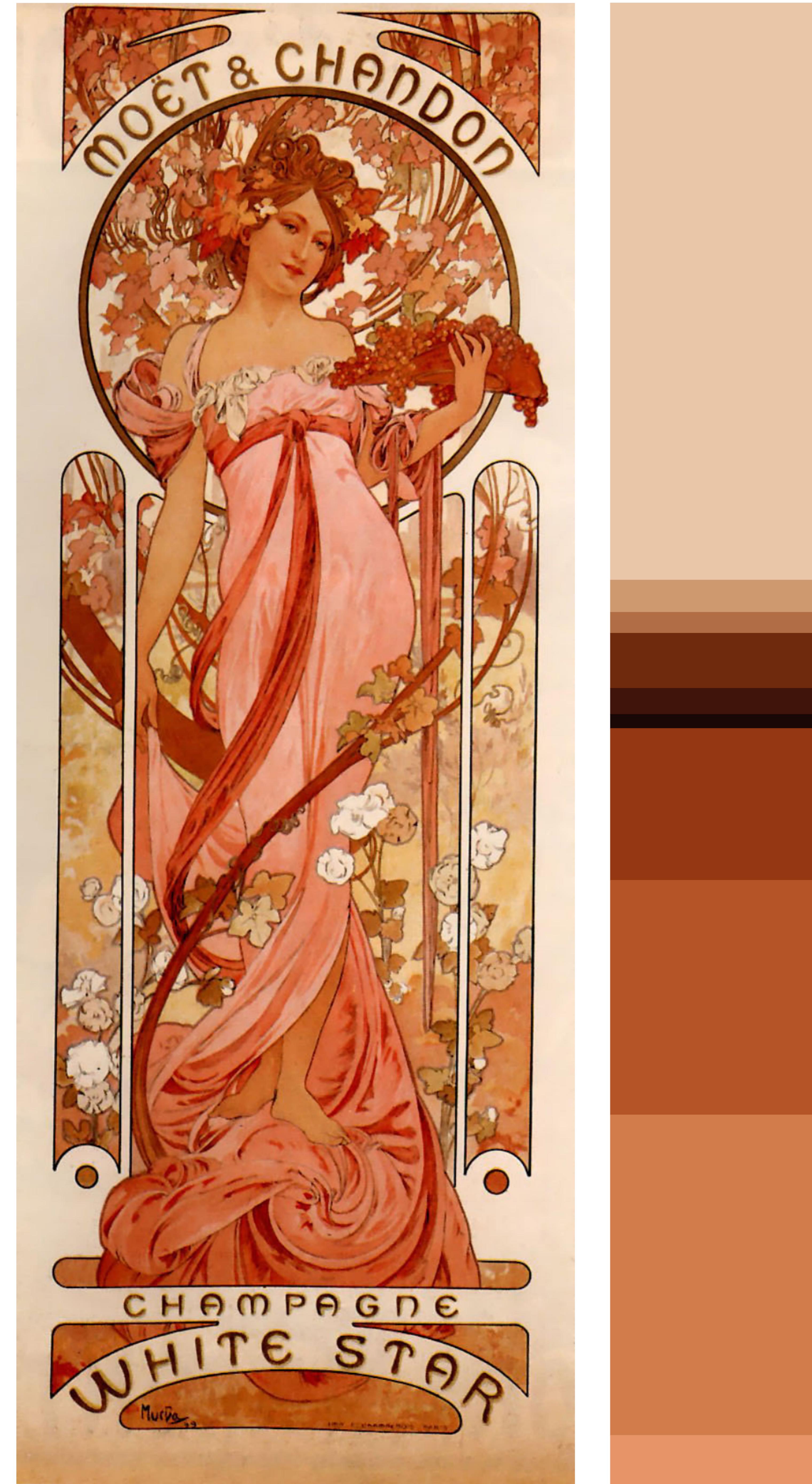
0xFFB84C3B

0xEE8D3526

0xFFED7961

0xFFD14C3D

artist:
Alphonse Mucha



0xFFEAC6A9

0xFFCE996F
0xFFB16E44
0xFF6F2A0D

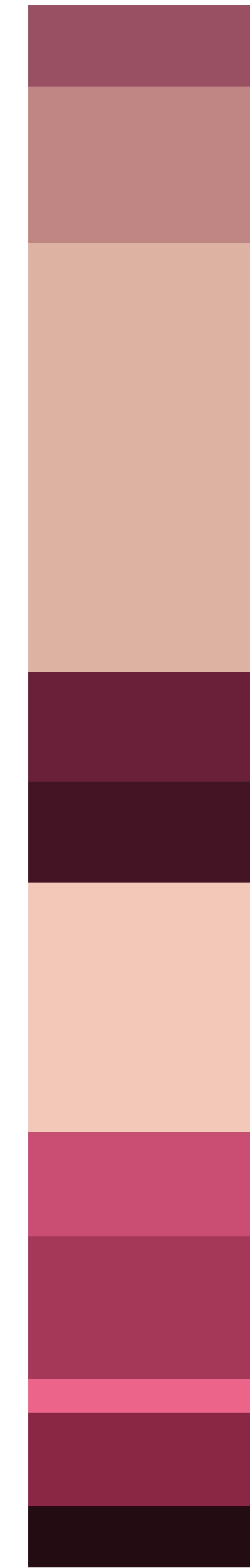
0xFF40140A
0xFF1A00806
0xFF933812

0xFFB55527

0xFFD17C4A

0FFE79469

artist:
Shintaro Ohata



artist:
Deborah Zlotsky



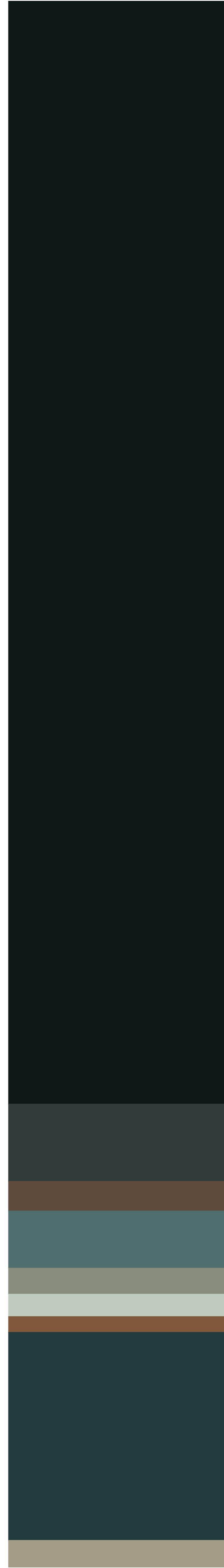
0xFFDDDDBA
0xFFBBCF8A
0xFF95A767
0xFF6B7944
0xFFB6BE20
0xFFD8CA4D
0xFF548137
0xFF91991B
0xFF386936
0xFFFF0000
0xFFB67B25
0xFF526245
0xFFFF2E828
0xFFA76C15

artist:
Shintaro Ohata



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

artist:
Sean Soong



0xFF0F1819
0xFF2E3B38
0xFF5E4A3A
0xFF4E6F6D
0xFF888D7E
0xFFB0CCBD
0xFF81583C
0xFF203C3D
0xFFA19C86

artist:
Deborah Zlotsky



0xFF5093B3
0xFF7D98AB
0xFFA5AFAD
0xFFCC9262
0xFF1F729E
0xFFAFC1D1
0xEE8B7D30
0xFFD99936
0xFFC28C14
0xFF665F43
0xEF4B7A21
0xFF99A039
0xFF2D482C
0xFF0F3958
0xFF7D993B
0xFF44597C

```
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.**