

THE
COLORIST
COOKBOOK


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

CHAPTER 01: 3 COLORS INTO 4 PART I

0xFF6FA247

0xFF59D013


0xFF64BA33



0xFF6914F8

0xFF3B3639


0xFF5528B3



0xFF040991

0xFF9B0F2C


0xFF6D0C6B



0xFFAC6F61

0xFFC1327A

0xFFB6566E



```

// 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

0xFFDA9A1D

0xFF9F0E8C


0xFFBE6D65



0xFF3496A7

0xFFD6A041


0xFF9B9B7E



0xFFE94811

0xFFFFCDA41


0xFFFF2A22F



0xFFA654A9

0xFF6A36AF

0xFF8B46AC



```

// 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

0xFF45403D

0xFFBABFC2

0xFF787878

0xFF8C8E8F

0xFF9DA0A2




0xFFC896C4

0xFF37693B

0xFFA688A4

0xFF928190

0xFF7A7979




0xFF63480B

0xFF9CB7F4

0xFF78778A

0xFF828BAC

0xFF8B9BC8



```

// 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

0xFFDA8CD9

0xFFC793C6


0xFFB299B1

0xFF9AA09A

0xFF7DA67E

0xFF59AC5B

0xFF06B216



0xFFD48642

0xFFD38B6D


0xFFD2918C

0xFFD296A5

0xFFD19CBB

0xFFD0A1CF

0xFFCFA6E1



0xFF66EDA9

0xFF8BE3AE


0xFFA8D8B3

0xFFC0CDB8

0xFFD6C2BC

0xFFEAB6C1

0xFFFFDA8C5



0xFF61BD68

0xFF7CC25F


0xFF93C855

0xFFA6CE4A

0xFFB8D33E

0xFFC8D82D

0xFFD7DD12



0xFFD11D80

0xFFD64C8A


0xFFDB6793

0xFFDF7D9C

0xFFE48FA5

0xFFE9A0AD

0xFFEDAFB5



```
// 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


0xFF9FE80B

0xFF6418FD




0xFF032BCC

0xFFD3AA06



0xFFCE0DEB 0xFF1DDE01



```
// 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();

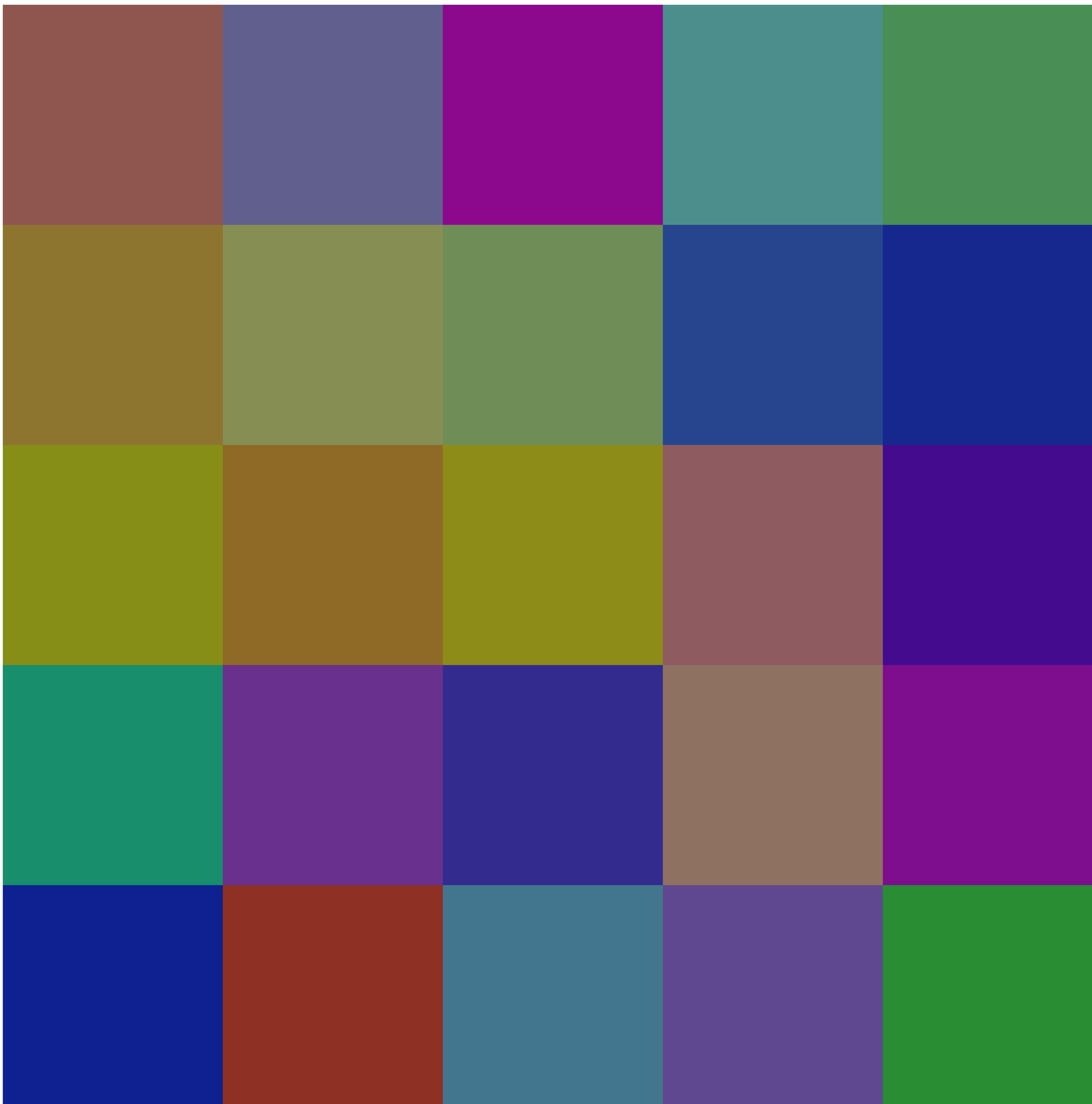
// glaze with the background color, which is opposite from the foreground color
// add a touch of randomness
color bg = color((hue(col1)+180)%360, random(90,100) , random(80,100));
fill(bg);
rect(0, 0, pgwidth,pgheight);

// 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);
```

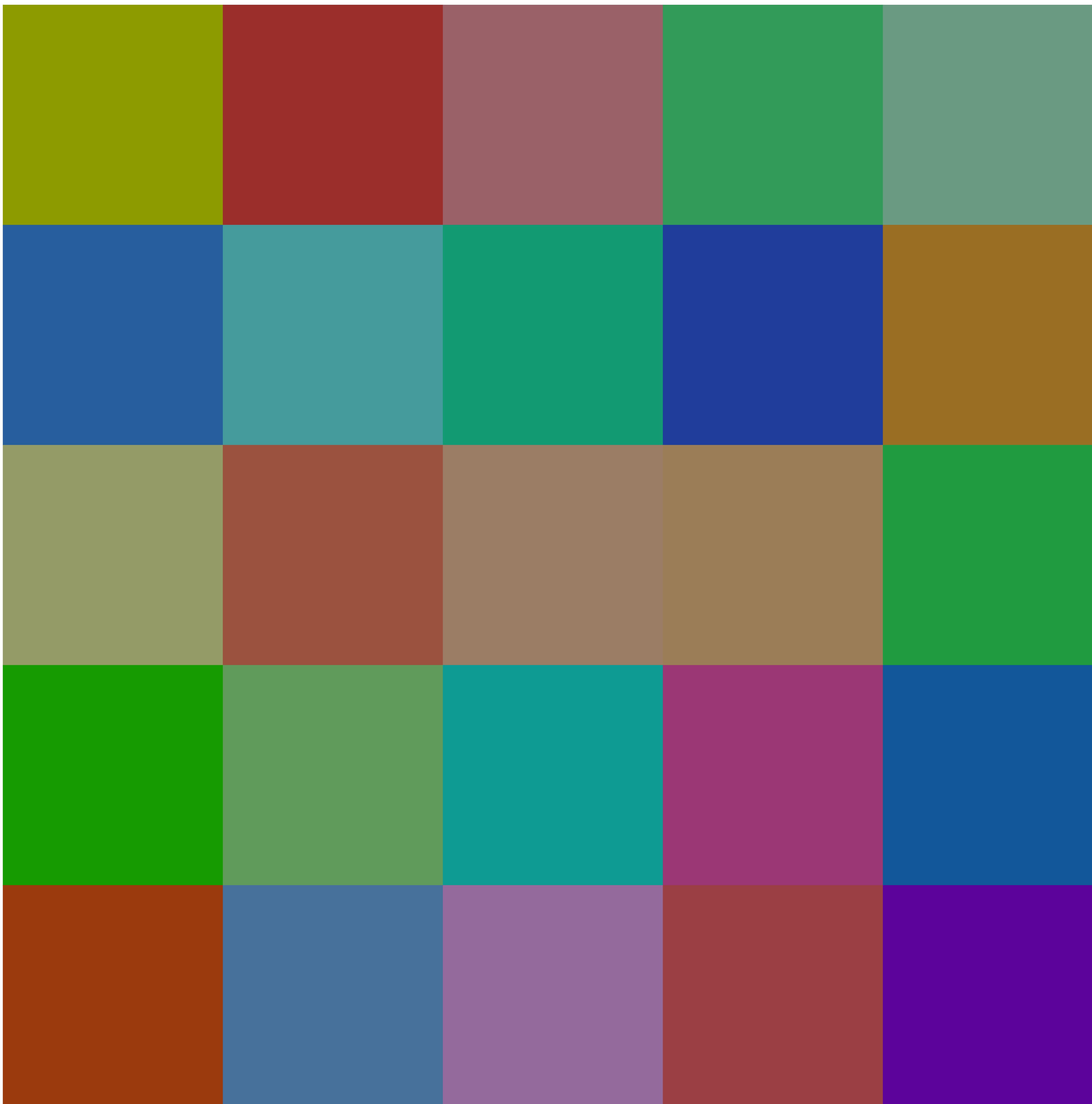
```
float recth = random(200, min(800, pgheight-posY)) ;  
  
// the slices can overlap with each other,  
// but they must be thin  
// like prosciutto  
fill(col1, 50);  
rect(posX, posY, rectw, recth);  
  
}  
  
// for balance, top with a few thin slices of the background color  
for(int i = 0;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

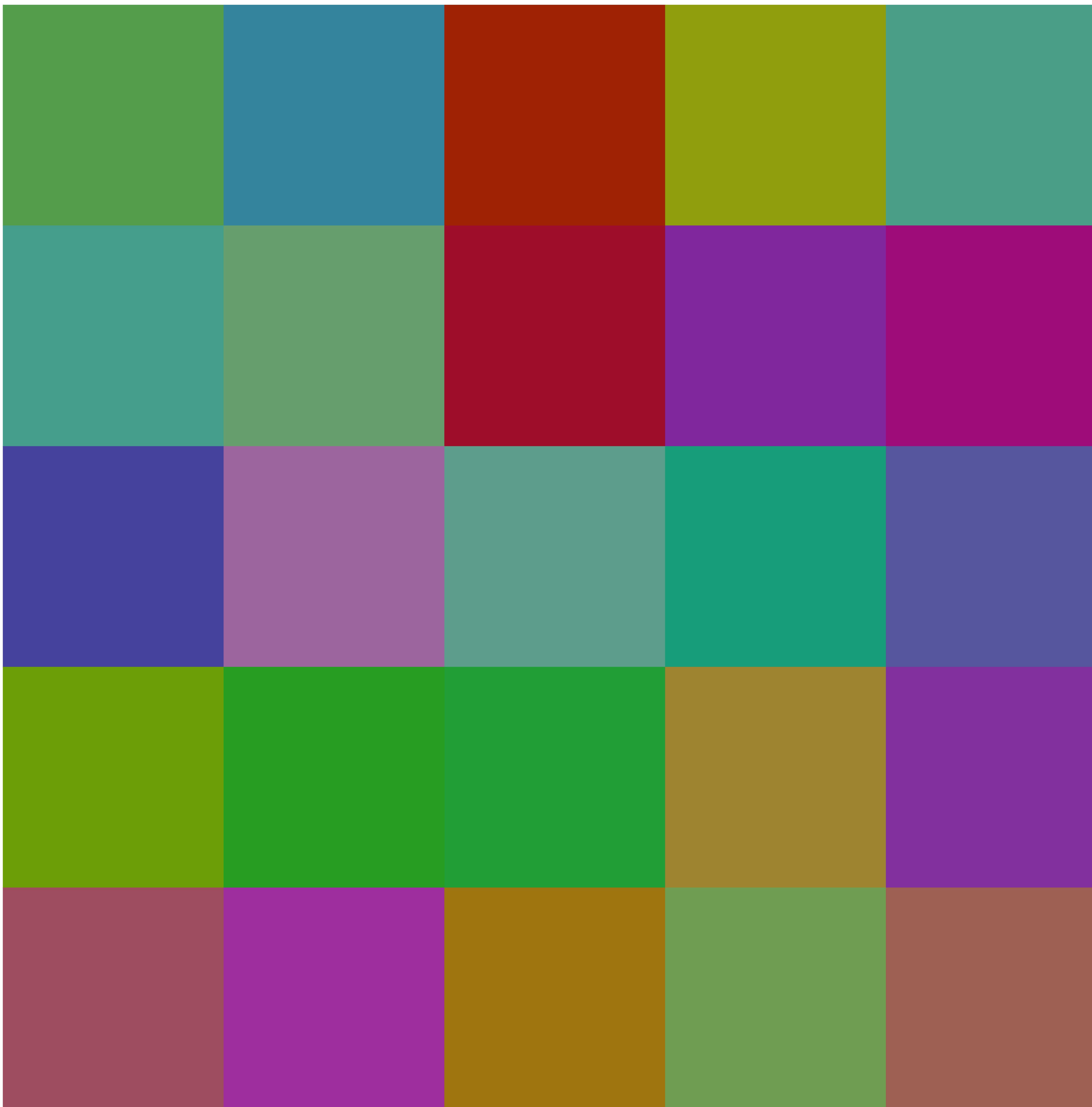
value: 56



value: 61



value: 62



```

// 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:
Shintaro Ohata



0xFF9F9893
0xFFD7D3CA
0xFF867C79
0xFFE7E2D8
0xFF5D6AA3
0xFF3A4083
0xFF314DA4
0xFF212B6D
0xFF859FD7
0xFFFFCAF4
0xFF6E5D48

artist:
Jamini Roy



0xFFFFAC74D
0xFF9C3C3F
0xFF88393A
0xFF4E3130
0xFF251F22
0xFF1D1A1F
0xFFE1D9D0

artist:
Deborah Zlotsky



0xFFDDDDBA
0xFFBBCF8A
0xFF95A767
0xFF6B7944
0xFFB6BE20
0xFFD8CA4D
0xFF548137
0xFF91991B
0xFF386936
0xFFFFCC00
0xFFB67B25
0xFF526245
0xFFFF2E828
0xFFFFA76C15

artist:
Linda Vachon



0xFFA50202
0xFF91100D
0xFF96AB93
0xFFC1BB9E
0xFF1D1B2A
0xFF7CA78E
0xFF4F6756
0xFF0F1C2F
0xFFH7B099
0xFF162640

artist:
Tatsuyuki Tanaka



0xFFAF9888

0xFFD4C0B7
0xFFFFE6E9
0xFF8B7667

0xFF624E41

0xFF423126

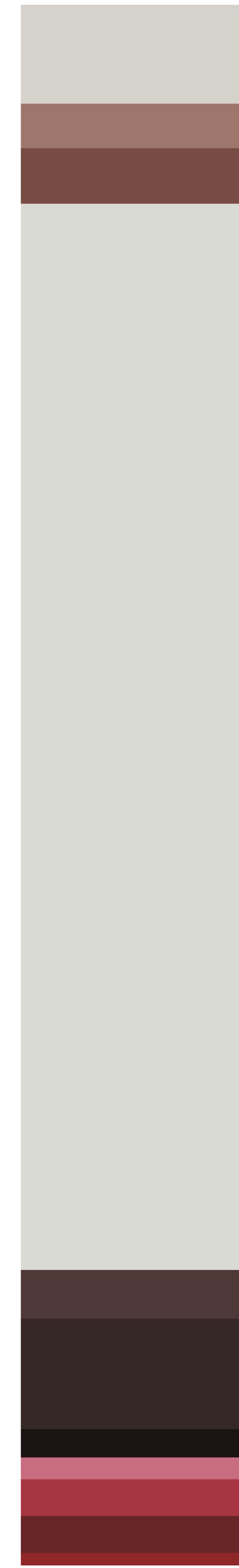
0xFF211811

0xFF0F0E08

artist:
Piotr Jablonski



artist:
Cy Twombly



artist:
Jenny Saville



0xFFE1CCCE

0xFFC2A5AC

0xFF9C8EA2

0xFF7B81A1

0xFF6780A7

0xFF503D59

0xFF8EA1C2

0xFF30202C

0xFF4A5A8F

0xFF140B0C

0xFFABB2CC

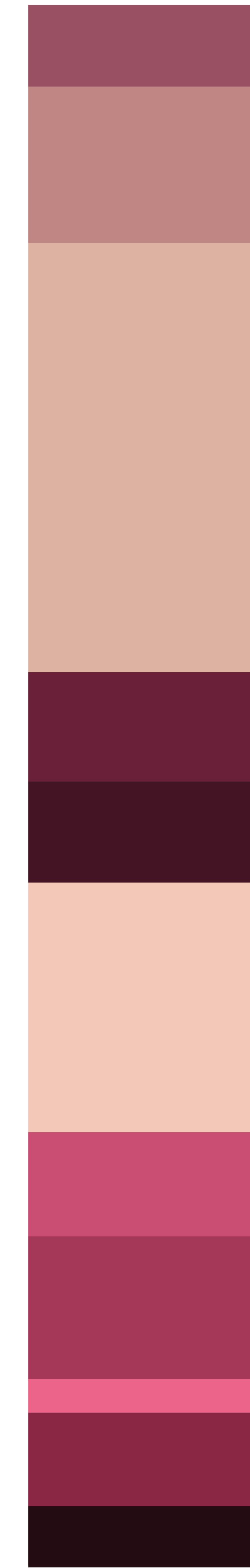
0xFFDFADA2

artist:
Sean Soong



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

artist:
Shintaro Ohata



0xFF985062
0xFFBF8684
0xFFDDB2A2
0xFF6A2039
0xFF431424
0xFFFF2C9B9
0xFFCA4E73
0xFFA53759
0xFFEC6589
0xFF8A2744
0xFF230C12

```
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.**