


THE
COLORIST
COOKBOOK

**"I TRY TO APPLY COLORS LIKE WORDS THAT
SHAPE POEMS, LIKE NOTES THAT SHAPE
MUSIC."**

0xFF319AB0

0xFFCA634F


0xFF868576



0xFF8E4E87

0xFF6CB67B


0xFF7D7C79



0xFF7B569A

0xFF8AAD68

0xFF818873



```
// recipe for simple simultaneous contrast

// prepare the first color
float r_col1 = random(2,84) + random(2,84) + random(2,84);
float g_col1 = random(2,84) + random(2,84) + random(2,84);
float b_col1 = random(2,84) + random(2,84) + random(2,84);

color col1 = color(r_col1, g_col1, b_col1);

// prepare the color opposite to the first color
// season with a touch of randomness
color col2 = color(255 - r_col1 + random(-7,7) ,
                    255 - g_col1 + random(-7,7) ,
                    255 - b_col1 + random(-7,7)) ;

// evenly mix the first two colors to create
// the 'middle' color
// (recommended) season with randomness
color mid = color((r_col1+red(col2))/2f + random(-15,15) ,
                   (g_col1+green(col2))/2f + random(-15,15) ,
                   (b_col1+blue(col2))/2f + random(-15,15)) ;

// 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/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();
```

0xFF675E4C

0xFF9899AF


0xFF887B8C



0xFF74B180

0xFF864A81


0xFF738189



0xFFCF627B

0xFF299881

0xFF7B737C



```

// recipe for holed simultaneous contrast

// prepare the first color
float r_col1 = random(2,84) + random(2,84) + random(2,84);
float g_col1 = random(2,84) + random(2,84) + random(2,84);
float b_col1 = random(2,84) + random(2,84) + random(2,84);

color col1 = color(r_col1, g_col1, b_col1);

// prepare the 'opposite' color to the first color
// season with a touch of randomness
color col2 = color(255 - r_col1 + random(-7,7) ,
                    255 - g_col1 + random(-7,7) ,
                    255 - b_col1 + random(-7,7)) ;

// evenly mix the first two colors to create
// the 'middle' color
// (recommended) season with randomness
color mid = color((r_col1+red(col2))/2f + random(-15,15) ,
                   (g_col1+green(col2))/2f + random(-15,15) ,
                   (b_col1+blue(col2))/2f + random(-15,15)) ;

// 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/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);
float rand4 = random(-250,250);
pushMatrix();


```

```
for (int cols = 0; cols < 4; cols++) {  
    rect (pgwidth * 0.668 + 140*cols, pgheight * 0.3 + 140*rows,100,100);  
}  
}  
popMatrix();  
  
popMatrix();
```

0xFF9C8179

0xFF617A83

0xFF747D79



```

// recipe for schotter simultaneous contrast

// prepare the first color
float r_col1 = random(2,84) + random(2,84) + random(2,84);
float g_col1 = random(2,84) + random(2,84) + random(2,84);
float b_col1 = random(2,84) + random(2,84) + random(2,84);

color col1 = color(r_col1, g_col1, b_col1);

// prepare the 'opposite' color to the first color
// season with a touch of randomness
color col2 = color(255 - r_col1 + random(-7,7) ,
                    255 - g_col1 + random(-7,7) ,
                    255 - b_col1 + random(-7,7)) ;

// evenly mix the first two colors to create
// the 'middle' color
// (recommended) season with randomness
color mid = color((r_col1+red(col2))/2f + random(-20,20) ,
                   (g_col1+green(col2))/2f + random(-20,20) ,
                   (b_col1+blue(col2))/2f + random(-15,15)) ;

// 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/2f,pgheight);
fill(col2);
stroke(col2);
rect(pgwidth/2f,0,pgwidth/2f,pgheight);


// arrange squares in a Georg Nees fashion
rectMode(CORNER);
fill(mid);
stroke(col1);
for (int rows = 0; rows < 25; rows++) {
    for (int cols = 0; cols < 15; cols++) {
        pushMatrix();
        translate(pgwidth * 0.1 + 60*cols, pgheight * 0.2 + 60*rows);
        // let squares scatter more near the bottom
        float rotamnt = random(-rows*0.05,rows*0.05);
        rotate(rotamnt);
        rect(0,0,60,60);
        popMatrix();
    }
}

// repeat previous step
rectMode(CORNER);
fill(mid);
stroke(col2);
for (int rows = 0; rows < 25; rows++) {
    for (int cols = 0; cols < 15; cols++) {

```


```
translate(pgwidth * 0.6 + 60*cols, pgheight * 0.2 + 60*rows);  
float rotamnt = random(-rows*0.05,rows*0.05);  
rotate(rotamnt);  
rect(0,0,60,60);  
popMatrix();  
  
}  
}  
  
popMatrix();
```

0xFF135FE6 0xFFD88C06




0xFF137BD9

0xFFFF77B0B




0xFF3F11F3

0xFFB2DC0E




0xFF9E10F8 0xFF62DE14



0xFF830DEF

0xFF77E710



```

// recipe for using bridging colors

// switch working color mode to HSB
// before preparing the foreground color
colorMode(HSB,360,100,100);
color col1 = color(random(0,360), 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);

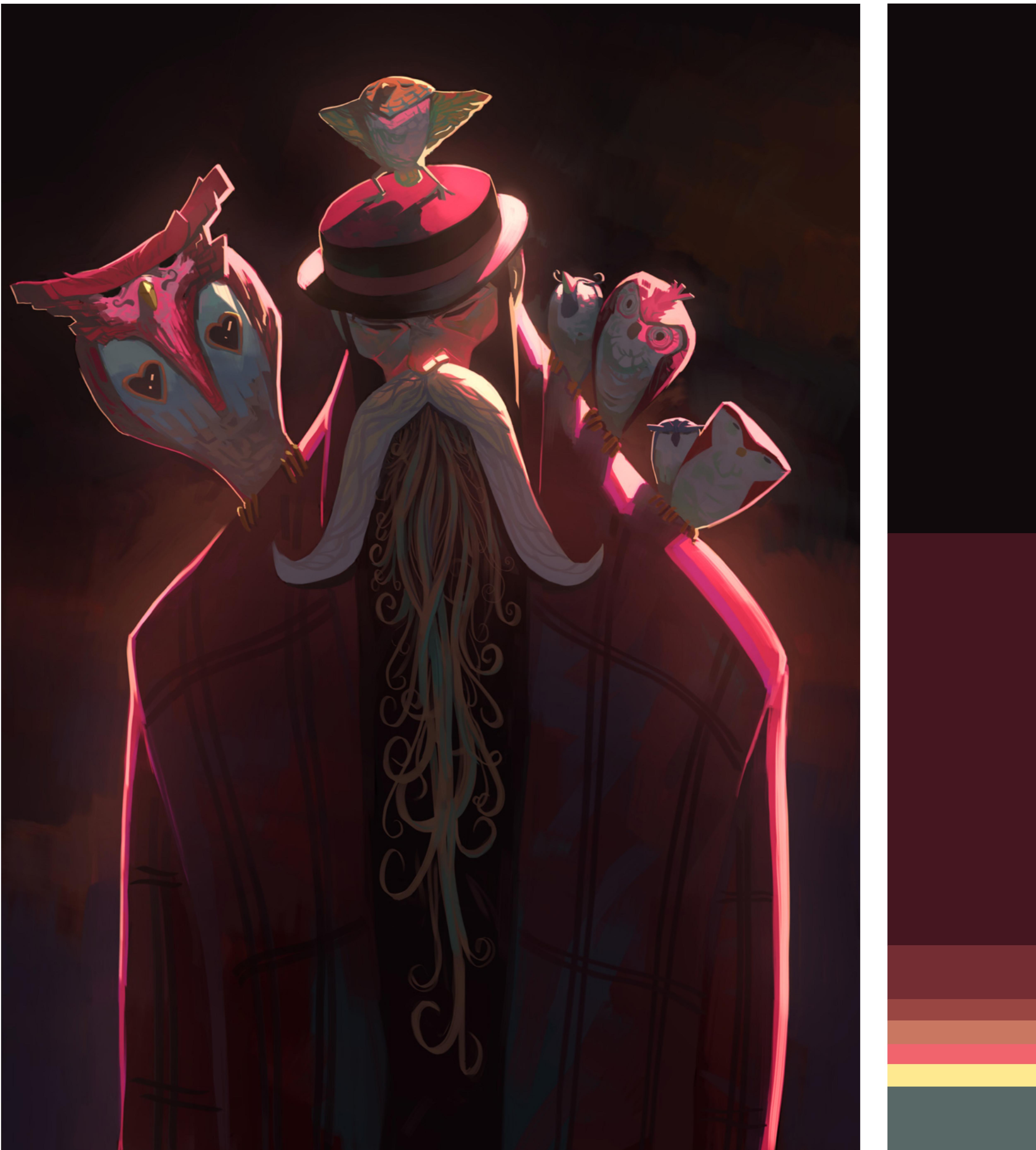
// randomly add 90 slices of the foreground color
for(int i = 0; i < 90; i++) {
    float posX = random(0, pgwidth-200);
    float posY = random(0, pgheight-200);

    float rectw = random(100, min(1200, pgwidth-posX));
    float recth = random(200, min(800, pgheight-posY));

    // the slices can overlap with each other,
    // but they must be thin
    // like prosciutto
}

```

```
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();
```



0xFF100A0C

0xFF44161F

0xFF732D33

0xFF994542

0xFFC97761

0xFFF1636D

0xFFFFEE991

0xFF576867



0xFF9B908C

0xFFB7D3C8

0xFFDBD7CE

0xFF67A0C1

0xFF85E9C8

0xFFFF07D9C

0xFFFFCF4F7

0xFF7E555B

0xFF543C3F

0xFF300F47

0xFFC961AC

0xFF0C0618



0xFFC8B8A8
0xFF9FBDB8
0xFF7BAAAA
0xFFEBD9CA
0xFF5299B6
0xFF0095B9
0xFF475E4C
0xFFFF7805D
0xFF6C5A56
0xFFB27457
0xFF914A33



0xFFDCC6B8

0xFFB39C8E

0xFF8E7066

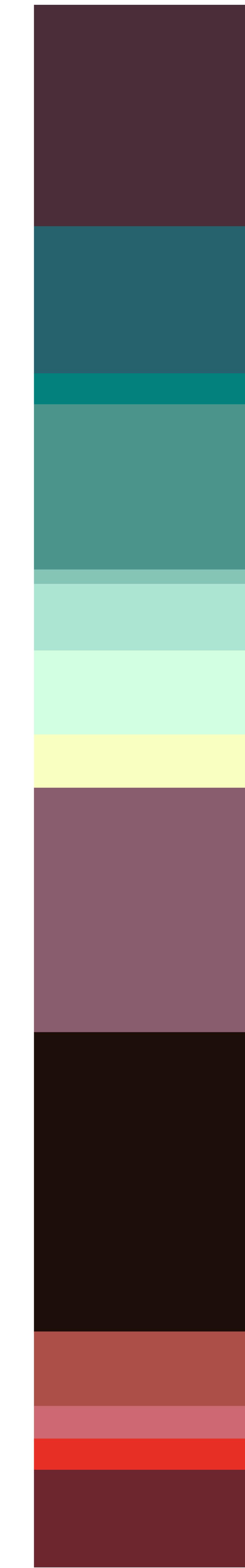
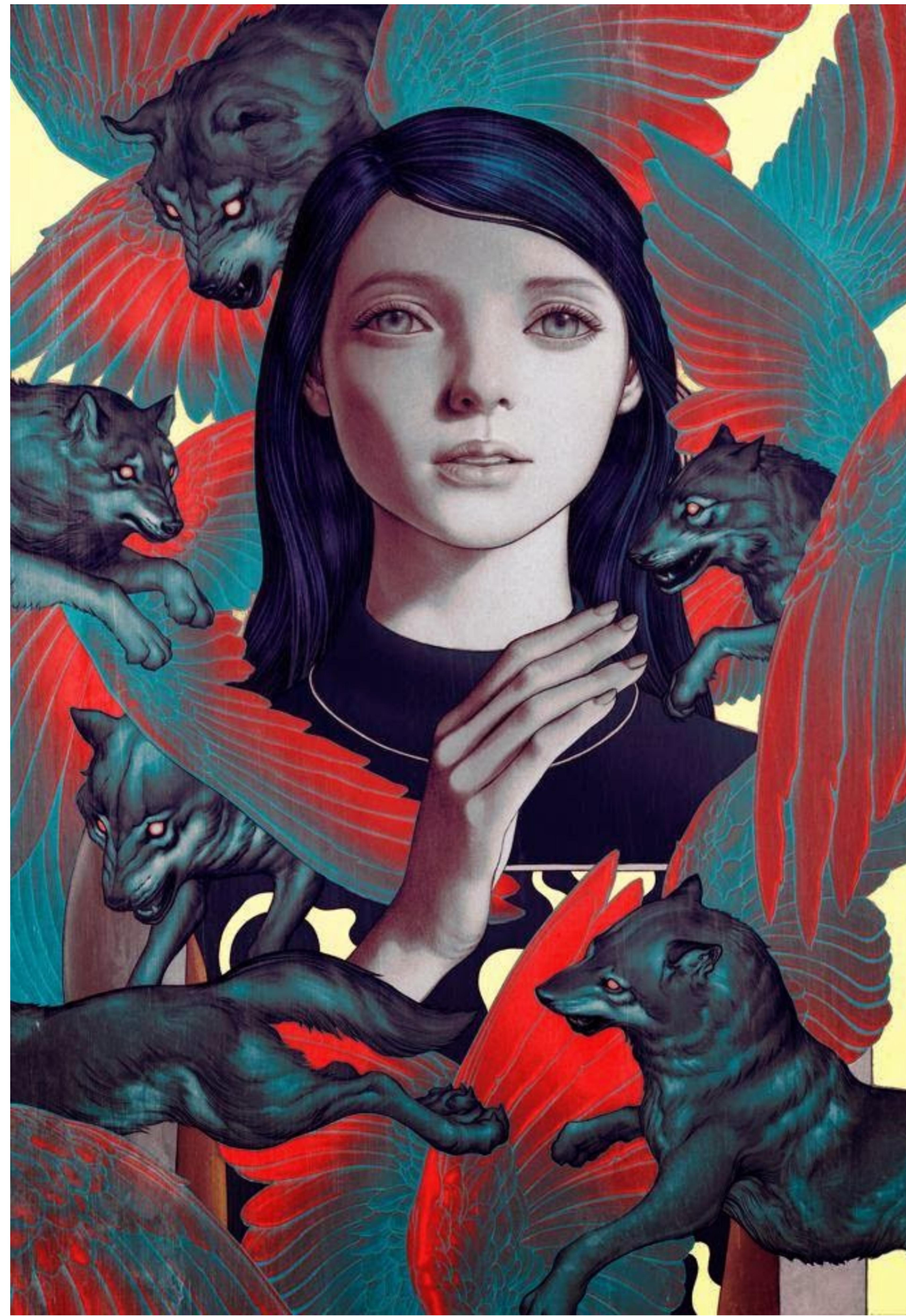
0xFF63463B

0xFF3C221A

0xFF1A0402

0xFFFFCEFE7





0xFF492D39

0xFF26616E

0xFF03817D
0xFF4A948C

0xEE84C5B5
0xFFFFACE6D3

0xFFD2FFE1

0xFFFFBFFC1

0xFF8A5D6E

0xFF1D0E09

0xFFAC4F48

0xFFCD6973
0xFFE82F25
0xFF6D262D

```
// recipe for palette grabber

// prepare initial ingredients
ArrayList<Pixel> colors = new ArrayList<Pixel>();
float tolerance = 50.0;
int totalCount = 0;
float proptolerance = 0.005;
PImage[] images = new PImage[6];

public class Pixel {
    public color pixelcolor;
    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();
    p.pixelcolor = 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);
    }

    int iwidth = images[imgindex].width;
    int iheight = images[imgindex].height;
    images[imgindex].resize(0, (int) pgheight);
    float palettepos = images[imgindex].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 = images[imgindex].get(i, j);

            if (notBlackorWhite(currpix)) {
                if (colors.size() == 0) {
                    addPixel(currpix);
                }
            }
        }
    }
}
```

```

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, colors.get(idx).pixelcolor);
    // 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--;
        }

        prevIdx = idx;
        minDist = currDist;
    }
    else if (!foundMatch) {
        colors.get(idx).count++;
        totalCount++;
        foundMatch = true;
        prevIdx = idx;
        minDist = currDist;
    }
}

if (!foundMatch) {

```

```

        }

    }

}

}

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(images[imgindex], 0, 0);

// place palette beside image
for (int idx = 0; idx < colors.size(); idx++) {
    float prop = colors.get(idx).count / ((float)netCount);
    if (prop >= proptolerance) {
        float rheight = pgheight * prop;

```

```
color paletteColor = colors.get(idx).pixelcolor;  
  
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);  
  
ypos += rheight;  
  
}  
}  
  
popMatrix();  
  
}
```

