06
색

06.01
도형 칠하기

우리는 이미 직간접적으로 색을 다루었습니다. 도형의 내부를 칠하는 `fill()` 함수, 획의 색을 정하는 `stroke()` 함수를 이용해 원하는 도형을 그렸습니다.

이번 시간에는 그동안 간단하게만 언급하고 지나갔던 색(color)에 대해 자세히 살펴보겠습니다. 지금까지 배웠던 여러 프로그래밍 지식을 한데 섞어 재미있는 그림을 잔뜩 그려봅시다.

background() 함수

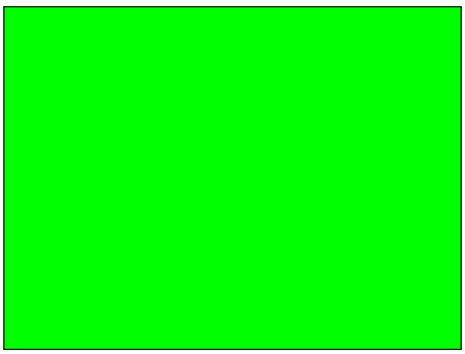
background() 함수를 이용하면 바탕화면의 색을 정할 수 있습니다.

```
//myBackground
void setup() {
  size(480, 360);
}

void draw() {
  //dark gray
  background(50);
}
```

myBackground

background()함수에 RGB값을 전하면 컬러를 표현할 수 있습니다.

```
//myBackgroundColor
void setup() {
   size(480, 360);
}

void draw() {
   //green
   background(0, 255, 0);
}
```

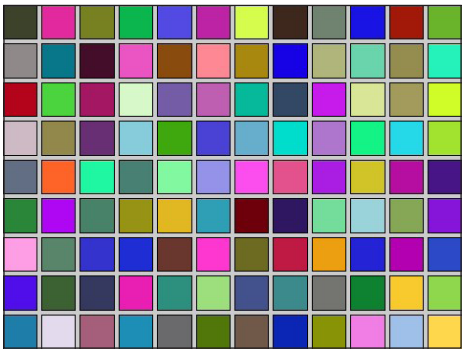


myBackgroundColor

fill() 함수

fill() 함수는 도형의 내부를 정해진 색으로 칠하는 함수입니다. fill() 함수가 실행된 다음에 그려지는 도형은 모드 fill() 함수의 영향을 받습니다.

```
//myFill
void setup() {
   size(480, 360);
   frameRate(1);
}

int nX = 12;
int nY = 9;
float margin = 5.0;

void draw() {
   float tileX = (width - (nX - 1) * margin) / nX;
   float tileY = (height - (nY - 1) * margin) / nY;

   for (int y = 0; y < nY; y++) {
      for (int x = 0; x < nX; x++) {
         float r = random(255);
         float g = random(255);
         float b = random(255);
         fill(r, g, b);
         float posX = x * (tileX + margin);
         float posY = y * (tileY + margin);
         rect(posX, posY, tileX, tileY);
         }
   }
}
```



myFill

획(stroke)

도형의 내부뿐만 아니라 도형의 윤곽선 색도 지정할 수 있습니다.

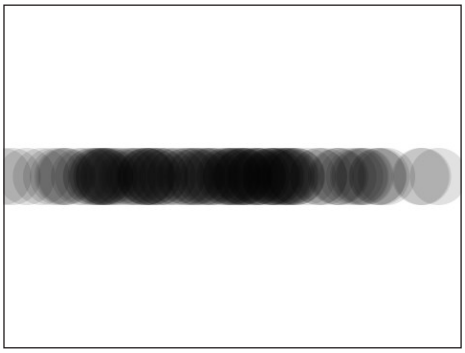
stroke() 함수를 이용하면 가능합니다.

불투명도

이전의 기억을 떠올려 보면 색을 칠할 때 투명도/불투명도를 이야기했던 기억이 있습니다. 보통 알파값이라고 불리는데 알파값이 낮은 색을 칠하면 아래에 칠한 색이 많이 비칩니다. 극단적으로 이야기해서 알파값이 0이라면 투명한 색을 칠하는 것과 같아 아래에 칠한 색이 모두 비칩니다. 반대로 알파값이 255라면 아래의 색을 모두 가려버립니다.

알파값이 낮은 색이라도 여러 번 같은 자리를 칠하면 색이 점점 진해집니다. 아래의 예제를 참고합시다.

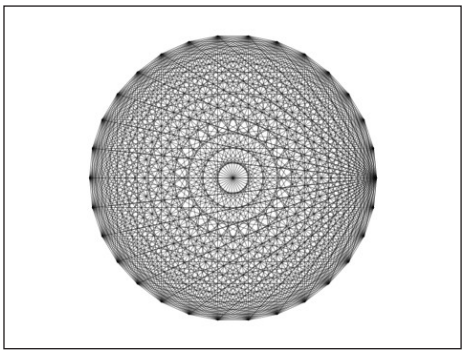
```
//myAlpha
void setup() {
   size(480, 360);
   background(255);
   noStroke();
   frameRate(10);
}

void draw() {
   float sigma = width / 5.0;
   float posX = width / 2.0 + sigma *
randomGaussian();
   float posY = height / 2.0;
   fill(0, 30);
   ellipse(posX, posY, 60, 60);
}
```



myAlpha

불투명도는 stroke()에도 적용이 가능합니다.

```
//myAlphaStroke
int[] posXs = new int[]{390,387,377,361,340,315,286
,256,224,194,165,140,119,103,93,90,93,103,119,14
0,165,194,224,256,286,315,340,361,377,387,390};

int[] posYs = new int[]{180,211,241,268,291,310,323
,329,329,323,310,291,268,241,211,180,149,119,92,
69,50,37,31,31,37,50,69,92,119,149,180};

void setup() {
   size(480, 360);
   noLoop();
}

void draw() {
   stroke(10, 50);
   strokeWeight(0.5);
```

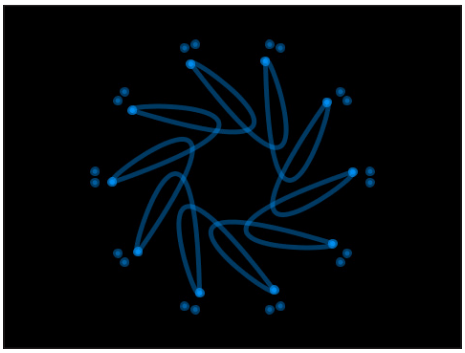


myAlphaStroke

```
   for (int i = 0; i < posXs.length; i++) {
      for (int j = 0; j < posXs.length; j++) {
         line(posXs[i], posYs[i], posXs[j], posYs[j]);
      }
   }
}
```

선으로 도형을 한 번 그려보니 이런 그림도 멋집니다. 조금 더 해볼까요? 이번에도 stroke에 알파값을 적용합니다. 이번에 그릴 그림은 Bezier 곡선을 적용합니다. Bezier 곡선이 무엇인지 일단 넘어가기로 하고 우선 결과물을 감상합시다.

```
///myBezier
void setup() {
   size(480, 360);
   //transparent black background
   background(0, 30);
   noFill();
}

float start = 0.0;
void draw() {
   background(0, 30);
   //pale purple line
   stroke(0, 150, 255, 100);
   strokeWeight(5);

   float innerRadius = min(width, height) * 0.35;
   float outerRadius = min(width, height) * 0.4;

   int n = 10;
   float theta = TWO_PI / n;

   translate(width / 2, height / 2);
   for (int i = 0; i < n; i++) {
      float X0 = innerRadius * cos(start + theta * i);
         float Y0 = innerRadius * sin(start + theta *
i);
         float X1 = outerRadius * cos((-start) +
theta * i);
         float Y1 = outerRadius * sin((-start) +
theta * i);
         float X2 = outerRadius * cos((+start) +
theta * (i + 1));
         float Y2 = outerRadius * sin((+start) +
theta * (i + 1));
         float X3 = innerRadius * cos(start + theta *
(i + 1));
         float Y3 = innerRadius * sin(start + theta *
(i + 1));
```



myBezier

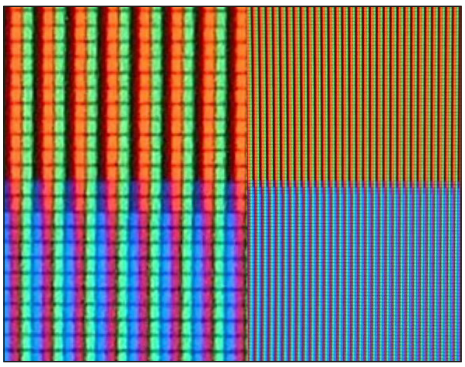
```
    fill(0, 150, 255, 100);
    ellipse(X0, Y0, 5, 5);
    ellipse(X1, Y1, 5, 5);
    ellipse(X2, Y2, 5, 5);
    ellipse(X3, Y3, 5, 5);

    noFill();
    bezier(X0, Y0, X1, Y1, X2, Y2, X3, Y3);
  }
start += 0.01;
}
```

05.02
RGB

색을 정하는 방법은 여러 종류가 있습니다. 가장 자주 접하게 될 혹은 이미 접한 두가지 컬러 시스템에 대해 알아보겠습니다. 우선 이미 익숙해진 RGB방식입니다.

RGB: LCD모니터로 설명하기

LCD모니터를 확대해서 본 적이 있으신가요? 자세히 살펴보면 세가지 다른 색의 점들이 화면에 깔려있는 것을 보실 수 있습니다.


RGB_pixels

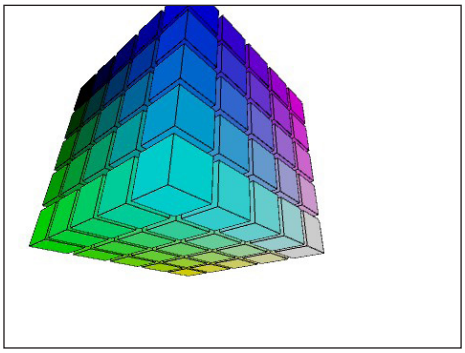
빨강, 녹색, 파랑색 이 세가지 색의 픽셀이 얼마나 밝게 빛나는지에 따라 모니터에서 다른 색으로 표현됩니다. 가령 오렌지색을 표현한다고 하면 빨강과 녹색을 켜고 파랑색 픽셀은 꺼버립니다. 빨강과 녹색빛이 섞이면 오렌지색으로 나타납니다. 비슷한 방법으로 하늘색을 표현하려면 빨강, 녹색 그리고 파랑색 픽셀을 적당한 밝기로 켭니다. 이 세가지 색이 섞이면 하늘색으로 보입니다.

이런 방법을 RGB색상이라고 합니다. Red/Green/Blue의 첫머리를 따서 지칭하는 말입니다. 따로 지정하지 않는다면 프로세싱은 RGB모드에서 작동합니다. 명시적으로 RGB모드를 지정할 때는 `colorMode(RGB)`를 사용합니다. 아래의 예에서 확인할까요?

RGB색상표를 만들어 봅시다. 세가지 축을 사용해야 하니 평면에서는 그

릴 수 없고 3D로 표현이 가능하겠네요. RGB 색상을 만들면서 3D도형도 연습해봅시다. 이번에는 일단 구경하는 것으로 생각합시다. 앞으로 배울 내용을 구경하면서 어떤 것들이 등장하는지 맛만 보는 시간을 가집시다. 모든 것을 알 필요는 없습니다. 필요한만큼 조금씩 하지만 꾸준하게 연습하는 것이 중요합니다.

```
//myRGB
void setup() {
   size(480, 360, P3D);
   background(255);
   colorMode(RGB);
}

float margin = 5.0;
float size = 30;

int nX = 3;
int nY = 3;
int nZ = 3;

int clickX;
int clickY;
int offsetX;
int offsetY;

float rotateX;
float rotateY;
float clickRotateX;
float clickRotateY;
float targetRotateX;
float targetRotateY;

void draw() {
   background(255);
   setView();
   for (int x = 0; x < nX; x++) {
      for (int y = 0; y < nY; y++) {
         for (int z = 0; z < nZ; z++) {
            pushMatrix();
            translate(x * (size + margin), y * (size +
margin), z * (size + margin));
            float r = map(x, 0, nX, 0, 255);
            float g = map(y, 0, nY, 0, 255);
            float b = map(z, 0, nZ, 0, 255);
            fill(r,g,b);
            box(size);
            popMatrix();
         }
      }
   }
}
```



myRGB

```
void mousePressed() {
   clickX = mouseX;
   clickY = mouseY;
   clickRotateX = rotateX;
   clickRotateY = rotateY;
}

void setView() {
   translate(100, 100);
   if (mousePressed) {
      offsetX = mouseX - clickX;
      offsetY = mouseY - clickY;
      targetRotateX = clickRotateX + offsetX /
float(width) * TWO_PI;
      targetRotateY = clickRotateY + offsetY /
float(height) * TWO_PI;
      rotateX += (targetRotateX - rotateX) * 0.25;
      rotateY += (targetRotateY - rotateY) * 0.25;
      }
   rotateX(-rotateY);
   rotateY(rotateX);
}
```

RGB 각 축을 따라 다른 색상이 조합되는 것을 관찰했나요? 마우스를 드래그하며 이리저리 색상표를 살펴볼 수도 있습니다.

05.03
HSB

이제 색을 구성하는 두번째 방법을 배우겠습니다. 미술시간 기분도 납니다. 미술시간에 배웠던 내용을 떠올리며 이것저것 만들어보며 즐거운 시간을 가져봅시다.

기억나실지 모르지만 HSB모드로 색을 지정하는 방법을 이미 연습한 적이 있습니다. 지난 시간 모로코 타일에서 붉은 계통의 타일색을 임의로 뽑을 때 사용을 했습니다. 모두 붉은 계통의 타일이지만 어떤 타일은 더 어둡고 다른 타일은 회색이 많이 비치는 등 분위기가 비슷한 여러 색을 지정할 때 HSB모드를 사용했습니다.

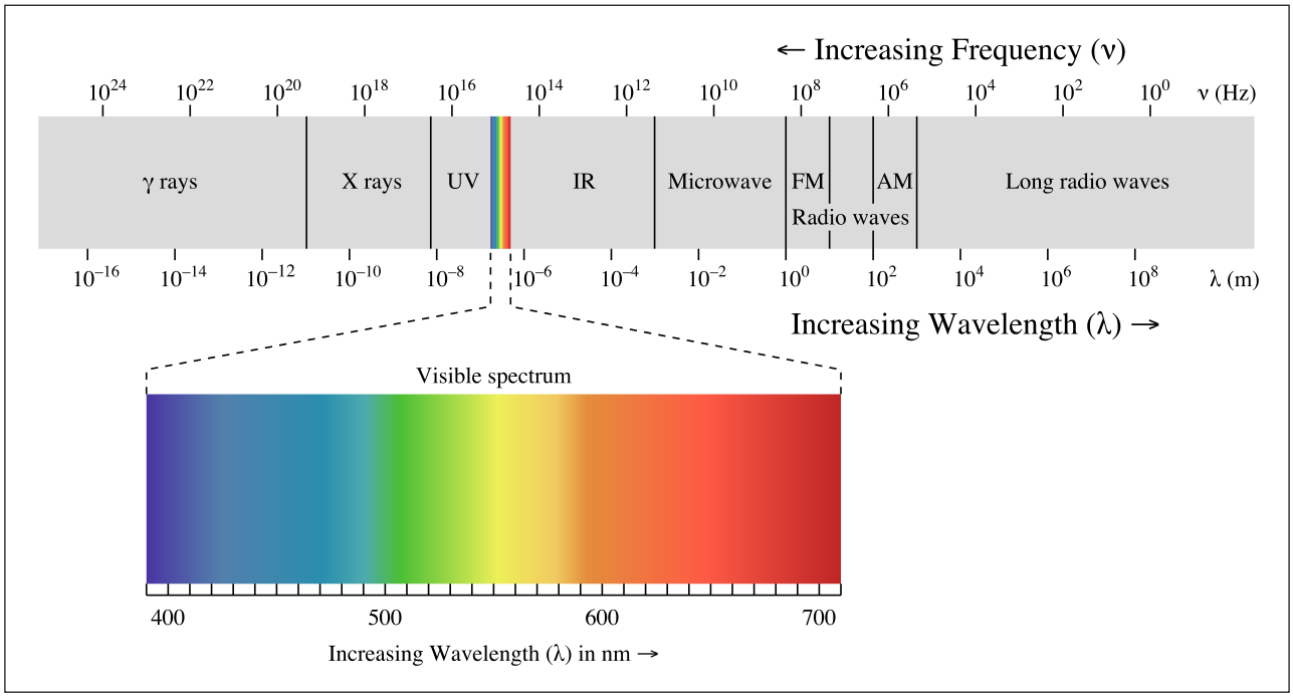
거칠게 이야기하자면 HSB모드는 사람에게 더 친근한 색체계입니다. 우리가 색을 이해하는 방법과 연관이 많은 HSB색상체계를 한 번 배워봅시다.

빛의 파장과 에너지

색에 대해 이야기하며 '빨주노초파남보'가 아직 나오지 않으니 당황스럽습니다. 여기서부터 '빨주노초파남보' 이야기가 시작됩니다.

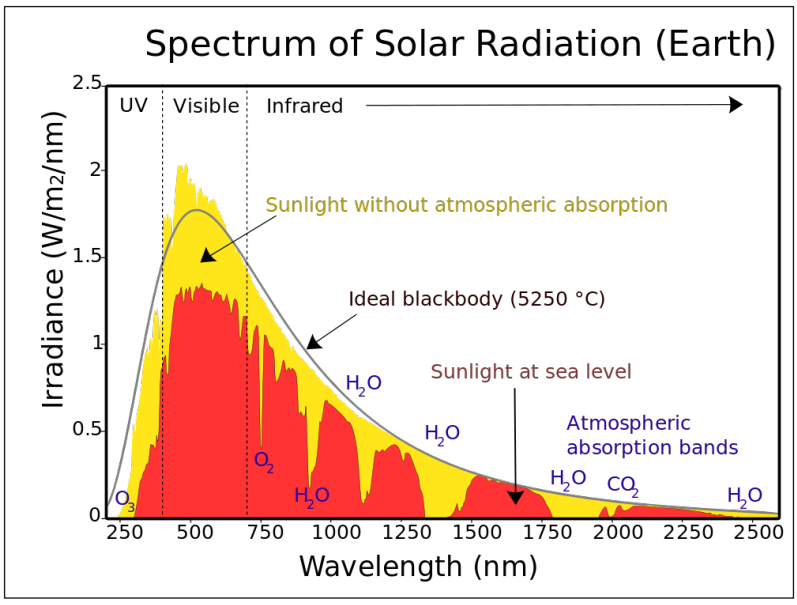
빛은 기본적으로 전자기파입니다. 핸드폰 전파, TV 전파, 자외선, 적외선 그리고 병원에서 찍는 X레이도 모두 전자기파의 일종입니다. 이렇게 많은 전자기파 중 우리가 볼 수 있는 것은 얼마되지 않고, 그런 전자기파

를 따로 가시광선이라고 칭합니다.



우리가 보는 빛에는 여러 파장이 섞여있습니다. 보기에는 하얀빛에 불과하지만 여러 파장의 - 여러 색의 - 빛이 혼합된 결과입니다. 결국 어떤 색이 어떻게 구성되어 있는지를 바탕으로 우리는 색을 구별하게 됩니다.

태양빛은 어떻게 구성되어 있을까요? 아래의 그래프는 태양빛의 파장과 에너지분포를 분석한 그림입니다.

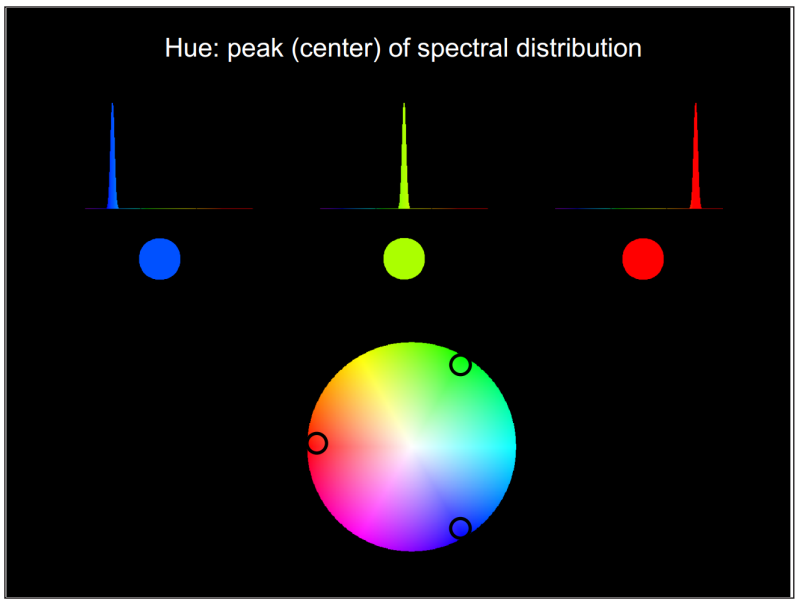


자연계의 빛은 이렇게 여러 파장의 빛이 다른 에너지를 가지고 섞여 있습니다. 다시 말하면 어떤 파장(색 Hue)이 어떤 구성(채도 Saturation)을 가지며 어떤 세기(명도 Brightness)를 띄는지가 색을 결정하는 요소입니다. HSB는 Hue, Saturation, Brightness의 첫 글자를 딴 약어입니다.

이제 색, 채도, 명도에 대해 하나씩 살펴봅시다.

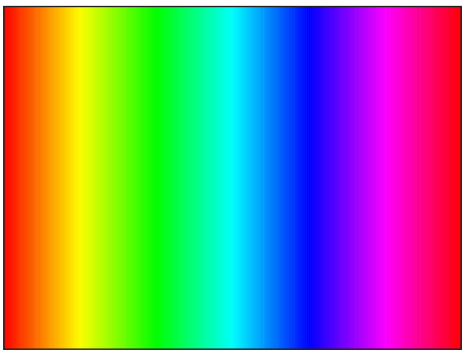
색조(Hue)

색조(Hue)는 가시광선의 파장에 따른 순색의 종류를 말합니다. 여러 파장이 섞여 있는 경우라면 가장 높은 에너지를 띄는, 다시 말해 분포도에서 꼭대기를 찍는 파장의 색을 말합니다.



빨간색(700nm)부터 보라색(400nm)까지 무지개색 순서에 따라 색을 나열해볼까요?

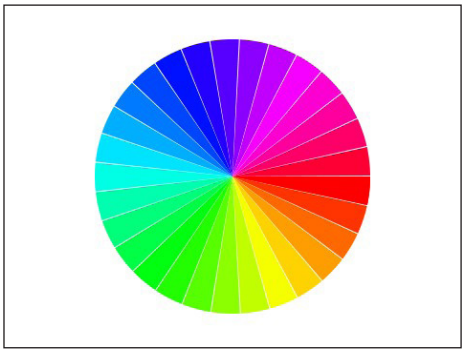
```
//myHue
void setup() {
   size(480, 360);
   colorMode(HSB);
   noStroke();
}

int nH = 30;
void draw() {
   float tileX = width / nH;
   for (int i = 0; i < nH; i++) {
      float posX = map(i, 0, nH - 1, 0, width);
      float h = map(i, 0, nH - 1, 0, 255);
      fill(h, 255, 255);
      fill(h, 255, 255);
      rect(posX, 0, tileX, height);
   }
}
```



myHue

HSB체계에서 Hue는 양쪽 끝을 이어서 원으로 표현하는 경우가 많습니다. 호(Arc) 그리는 법을 복습해보는 시간입니다.

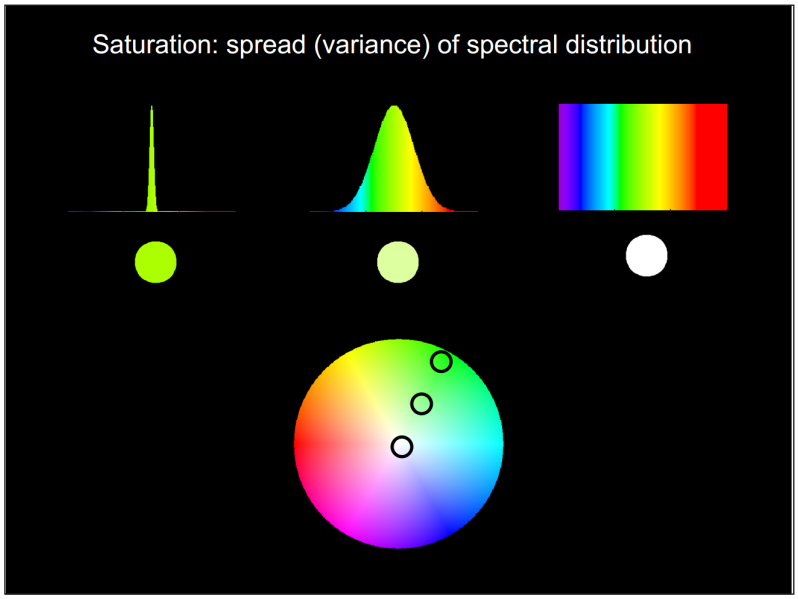
```
//myHueDisk
void setup() {
   size(480, 360);
   colorMode(HSB);
   noStroke();
}

int nH = 30;

void draw() {
   background(255);
   float radius = min(width, height) * 0.8;
   for (int i = 0; i < nH; i++) {
      float h = map(i, 0, nH - 1, 0, 255);
      fill(h, 255, 255);
      float start = map(i, 0, nH - 1, 0, TWO_PI);
      float end = start + TWO_PI / nH;
      arc(width / 2.0, height / 2.0, radius, radius,
start, end);
   }
}
```



myHueDisk

채도(Saturation)

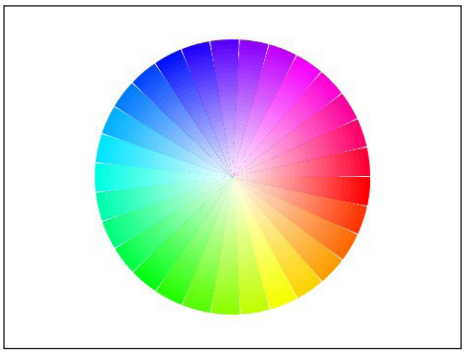
같은 색(Hue)를 가진다는 말은 단지 가장 높은 에너지를 가진 파장이 같다는 것을 말해줄 뿐 다른 파장의 분포에 대한 정보를 제공하지 못합니다. 채도는 파장의 분포에 대한 정보를 제공합니다. 채도가 높으면 색이 순수해집니다. 채도가 낮으면 다른 파장이 많이 섞여 있어 회색을 띄게 됩니다.



보통 Hue 디스크의 바깥쪽에 채도가 높은 색을 배열합니다. 바깥쪽으로 갈수록 색이 더 선명해집니다. 색이 선명해진다는 것은 다른 파장이 덜 섞였다는 뜻입니다. 밝아진다는 것과는 다른 말입니다.

이제 색조과 채도를 함께 디스크에 담아봅시다.

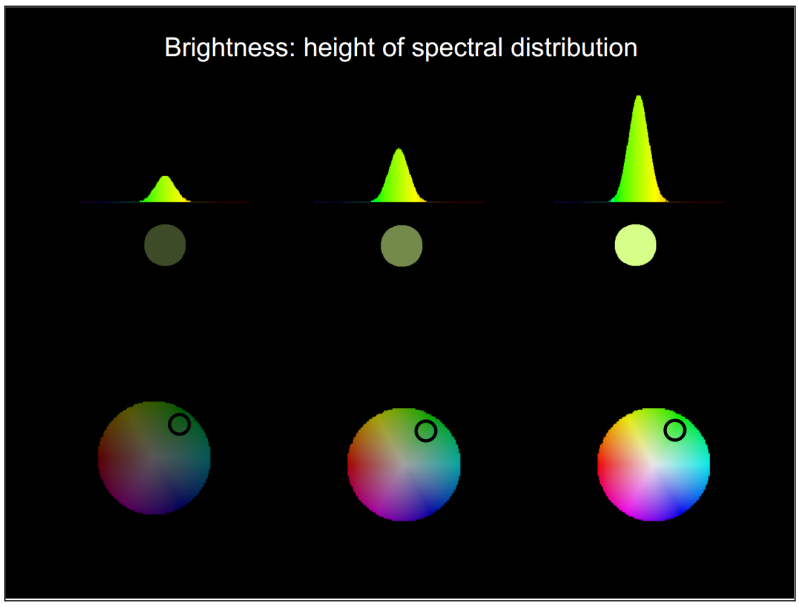
```
//myHueSaturationDisk
void setup() {
   size(480, 360);
   colorMode(HSB);
   noStroke();
   ellipseMode(RADIUS);
}

int nH = 30;
int nS = 30;

void draw() {
   background(255);
   float radius = min(width, height) * 0.4;
   for (int i = 0; i < nH; i++) {
      for (int j = 0; j < nS;j++) {
         float h = map(i, 0, nH - 1, 0, 255);
         float s = map(j, 0, nS - 1, 255, 0);

         fill(h, s, 255);
         float start = map(i, 0, nH - 1, 0, TWO_PI);
         float end = start + TWO_PI / nH;
         float r = map(j, 0, nS - 1, radius, 0);
         arc(width / 2.0, height / 2.0, r, r, start,
end);
      }
   }
}
```



myHueSaturationDisk

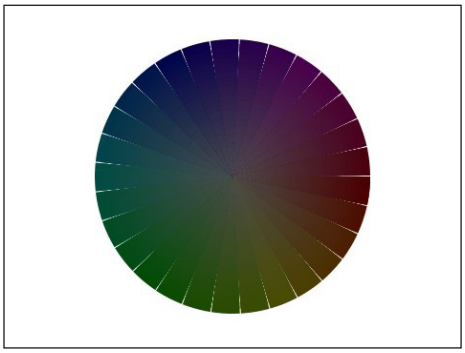
명도(Brightness)

빛이 에너지를 얼마나 가지고 있는지를 알려주는 특성입니다. 위의 그래프 아래의 면적에 해당합니다. 에너지가 낮을수록 어두운 색이 됩니다. 그늘에 있는 물체이 색이 어두워지는 것과 같은 원리입니다.

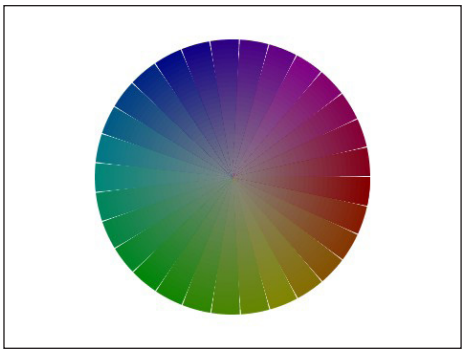


시간에 따라 명도가 달라지는 도형을 만들어봅시다.

```
//myHueSaturationBrightnessDisk
void setup() {
   size(480, 360);
   colorMode(HSB);
   noStroke();
   ellipseMode(RADIUS);
}

int nH = 30;
int nS = 30;
float b = 0.0;
float step = 1.0;

void draw() {
   background(255);
   float radius = min(width, height) * 0.4;
   for (int i = 0; i < nH; i++) {
      for (int j = 0; j < nS;j++) {
         float h = map(i, 0, nH - 1, 0, 255);
         float s = map(j, 0, nS - 1, 255, 0);
         fill(h, s, b);
         float start = map(i, 0, nH - 1, 0, TWO_PI);
         float end = start + TWO_PI / nH;
         float r = map(j, 0, nS - 1, radius, 0);
         arc(width / 2.0, height / 2.0, r, r, start,
end);
      }
```



myHueSaturationBrightnessDisk BeginB



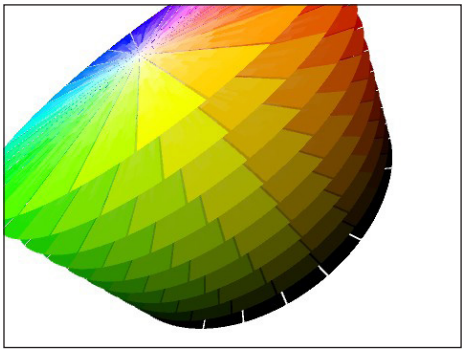
myHueSaturationBrightnessDisk BeginE

```
  }

  if (b > 255) {
    step = -1;
  }
  else if (b < 0) {
    step = 1;
  }
  b += step;
}
```

HSB색상표

RGB색상표를 만든 것처럼 HSB색상표를 만들어봅시다. 큐브모양으로 색상표를 만든 RGB와 다르게 HSB는 원기둥모양으로 색상표를 만들 것입니다. Hue를 원형으로 배치하고 Saturation은 바깥으로 갈수록 높게 배치하고 높이을 따라서는 Brightness를 배열합시다.

```
// myHSB
int clickX;
int clickY;
int offsetX;
int offsetY;

float rotateX;
float rotateY;
float clickRotateX;
float clickRotateY;
float targetRotateX;
float targetRotateY;

int nH = 30;
int nS = 30;
int nB = 10;

color WHITE = color(255, 255, 255);
float radius;

void setup() {
  size(480, 360, P3D);
  colorMode(HSB);
  radius = min(width, height) * 0.4;
  ellipseMode(RADIUS);
  noStroke();
}

void draw() {
  background(WHITE);
  setView();
  for (int b = 0; b < nB; b++) {
```



myHSB

```
      pushMatrix();
      float cylinderHeight = map(b, 0, nB - 1, 0,
radius);
      float brightness = map(b, 0, nB - 1, 0, 255);
      translate(0, 0, cylinderHeight);

      for (int h = 0; h < nH; h++) {
         for (int s = nS; s > 0; s--) {
            float begin = map(h, 0, nH - 1, 0, TWO_PI);
            float end = begin + TWO_PI / float(nH);
            float hue = map(h, 0, nH - 1, 0, 255);
            float r = map(s, 0, nS - 1, 0, radius);
            float saturation = map(s, 0, nS - 1, 0,
255);

            fill(hue, saturation, brightness);
            arc(0, 0, r, r, begin, end);
         }
      }
   popMatrix();
   }
}

void mousePressed() {
   clickX = mouseX;
   clickY = mouseY;
   clickRotateX = rotateX;
   clickRotateY = rotateY;
}

void setView() {
   translate(width / 2.0, height / 2.0);
   if (mousePressed) {
      offsetX = mouseX - clickX;
      offsetY = mouseY - clickY;
      targetRotateX = clickRotateX + offsetX /
float(width) * TWO_PI;
      targetRotateY = clickRotateY + offsetY /
float(height) * TWO_PI;
      rotateX += (targetRotateX - rotateX) * 0.25;
      rotateY += (targetRotateY - rotateY) * 0.25;
   }
   rotateX(-rotateY);
   rotateY(rotateX);
}
```

일상 속의 HSB체계

HSB를 이용하면 생활속의 색에 쉽게 대응할 수 있습니다. 그림자를 표현하고 싶을 때, 색과 채도를 유지한 채 명도를 낮춰봅시다. 물속의 물체

를 표현하고 싶다면 색과 명도를 유지한 채 채도를 낮추어 봅시다. 빛이 물속에서 산란되면 채도가 낮아지게 됩니다.

05.03
그라데이션

어떤 두 색을 섞으려면 어떻게 해야 할까요? 절반씩 섞는 것도 좋지만 점진적으로 두 색이 섞이는 모습을 볼 수 있으면 더 좋겠습니다. 빈번하게 요구되는 일인지라 프로세싱은 이런 작업을 위한 함수를 미리 만들어 두었습니다.

lerpColor() 함수

lerpColor() 함수는 두 색을 섞어 새로운 색을 만듭니다. 어떤 색을 더 진하게 섞느냐를 입력값을 추가해서 조정할 수 있습니다.

lerpColor() 함수는

```
color mixed = lerpColor(color1, color2, amount)
```

형식으로 사용됩니다. amount에는 0에서 1.0사이 값이 옵니다. 수식으로 표현하면 아래와 같은 관례가 성립합니다.

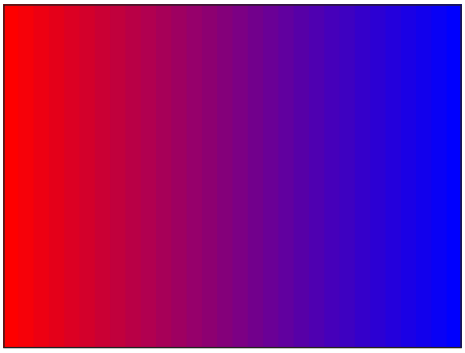
```
mixed = (1 - amount) * color1 + amount * color2
```

RGB색상모드에서 그라데이션

RGB색상공간에서 두 색 사이를 잇는 최단거리는 공간을 뚫는 직선의 모습을 띌 것입니다. 이제 임의의 두 색 사이의 중간색을 구해봅시다.

Red부터 시작히 Blue까지 그라데이션을 만들어 봅시다.

```
//myRGBGradation
void setup() {
   size(480, 360);
   noStroke();
}

int nStep = 30;

void draw() {
   color r = color(255, 0, 0);
   color b = color(0, 0, 255);

   float tileWidth = width / nStep;
   for (int i = 0; i < nStep; i++) {
      color mixed = lerpColor(r, b, map(i, 0, nStep -
1, 0, 1));
      fill(mixed);
      rect(tileWidth * i, 0, tileWidth, height);
   }
}
```



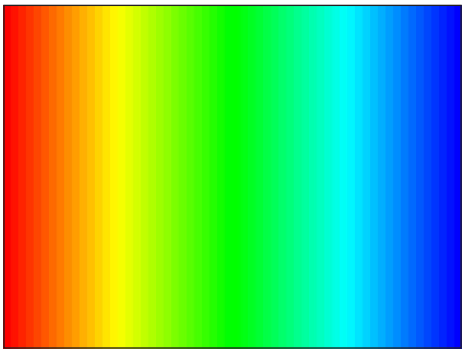
myRGBGradation

우리가 예상한 것과 비슷한가요? 이제 HSB모드에서 같은 작업을 수행해 봅시다.

HSB색상모드에서 그라데이션

아이디어는 동일합니다. lerpColor() 함수를 이용해 중간색을 구해 볼 것입니다. 유일한 차이는 RGB모드 대신 HSB를 이용한다는 것 뿐입니다.

```
//myHSBGradation
void setup() {
   size(480, 360);
   noStroke();
}

int nStep = 30;

void draw() {
   colorMode(RGB);
   color r = color(255, 0, 0);
   color b = color(0, 0, 255);

   colorMode(HSB);
   float tileWidth = width / nStep;
   for (int i = 0; i < nStep; i++) {
      color mixed = lerpColor(r, b, map(i, 0, nStep -
```



myHSBGradation

```
1, 0, 1));
      fill(mixed);
      rect(tileWidth * i, 0, tileWidth, height);
    }
}
```

RGB모드와 HSB모드의 차이점을 발견하셨나요? RGB는 중간색에 노란색이나 연두색이 보이지 않습니다. 반면 HSB는 여러 색조를 중간색으로 가집니다.

이유는 이렇습니다. RGB 색공간에서 두 색 사이의 최단거리는 두 지점을 잇는 직선입니다. 이 직선과 교차하는 색을 중간색으로 표현합니다.

반면 HSB모델은 직선으로 중간값을 찾는 것이 아니라 원을 그리며 최단거리를 찾습니다. 공간자체가 그렇게 휘어져 있기 때문에 바깥에서 보기에는 둘러가는 것 같지만 공간 내에서는 최단거리를 찾아가고 있는 것입니다.

05.04
color

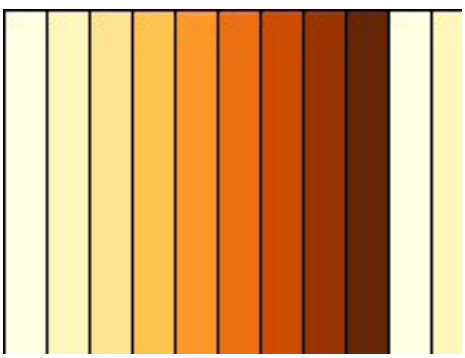
혹시 바로 전 코드에서 이상한 것을 발견하지 않았나요? color라는 키워드가 int나 float과 같은 데이터 타입 형태로 사용되었습니다. 이건 무엇인가요?

프로세싱은 color라는 특별한 형태의 데이터 타입을 가지고 있습니다. color는 정수와 비슷한 데이터 타입이지만 약간의 차이가 있습니다. color는 말 그대로 색상을 나타내는데 특화된 데이터 타입입니다.

color 변수

예를 들어 인터넷을 떠돌다 적당한 색을 발견했다고 합시다. 이 색상을 저장해서 원할 때마다 사용하고 싶으면 color 데이터 타입의 변수로 지정하면 나중에 사용할 수 있습니다.

```
//myColor
color[] myColor = new color[]{#ffffe5, #fff7bc,
#fee391, #fec44f, #fe9929, #ec7014, #cc4c02,
#993404, #662506};

void setup() {
   size(480, 360);
}

int n = 30;
```



myColor

```
  void draw() {
    float tileX = width / n;
    for (int i = 0; i < n; i++) {
      fill(myColor[i % myColor.length]);
      rect(tileX * i, 0, tileX, height);
    }
  }
```

pixels[] 배열과 컬러

결국 우리가 보는 화면은 픽셀로 구성되어 있습니다. 3D로 도형을 그리든 움직이는 도형을 그리든 최종 결과물은 특정 색을 띄는 픽셀의 모임으로 표현됩니다. 그렇다면 중간단계를 거치지 않고 픽셀을 바로 수정하는 것은 어떨까요? 이제 픽셀에 바로 접근하는 법을 배워봅시다.

프로세싱 스케치는 pixels[] 배열에 저장된 color값으로 결정됩니다. 여기까지 이야기를 들으면 자연스럽게 떠오르는 질문이 있습니다. pixels[] 배열은 1차원 배열인데 어떻게 2차원 화면의 정보를 모두 저장할 수 있을까요?

의외로 간단합니다. (x, y) 위치의 픽셀값은 pixels[x + h * width]에 저장되어 있습니다.

random() 함수로 화면 표현하기

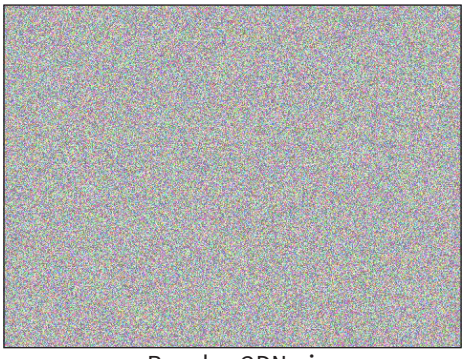
픽셀 별로 임의의 색을 가지도록 random() 함수와 color() 함수를 활용해 봅시다.

pixels[] 함수를 이용할 때는 화면의 픽셀 정보를 불러오도록 명령하는 loadPixels() 함수를 먼저 실행해야 합니다.

pixels[] 배열의 정보를 수정하고 난 후에는 바뀐 정보를 바탕으로 화면을 업데이트하도록 updatePixels() 명령을 실행해야 합니다.

순서를 기억하세요. loadPixels(), pixels[] 그리고 마지막으로 updatePixels()를 실행합니다.

```
//myRandom2DNoiseRGB
size(480, 360);
loadPixels();
for (int x = 0; x < width; x++) {
  for (int y = 0; y < height; y++) {
    color c = color(random(100, 255), random(100,
255), random(100, 255));
    pixels[x + y * width] = c;
  }
}
```



myRandom2DNoise

```
updatePixels();
```

한가지 추가로 주의할 점이 있습니다. pixels[] 배열을 사용할 때는 setup()이나 draw() 함수를 사용하지 않습니다. 바로 픽셀값을 수정하는 것으로 충분합니다.

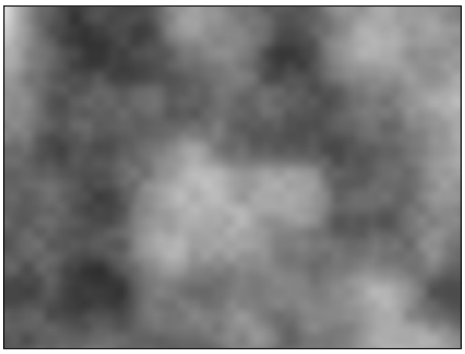
noise() 함수 사용하기

random() 함수로 만든 화면은 어수선합니다. 너무나 랜덤해서 기계가 만든 것처럼 눈에 거슬리기만 합니다. 구름이 떠다니는 모습이나 골짜기의 높낮이처럼 적당히 랜덤한 모습을 그리고 싶으시다면 random() 함수 대신 noise() 함수를 사용하는 것을 고려할 때입니다.

noise() 함수를 이용하면 그럴듯한 화면을 만들 수 있습니다. random() 함수와 비교하면 약간 복잡하게 보일 수도 있지만 익숙해지면 불편함보다는 편리함이 더 부각되게 될 것입니다.

noise() 함수를 이용해 pixels[] 배열의 색상을 조작하겠습니다. 일단 흑백으로 작업을 하겠습니다.

```
//myNoise
size(480, 360);
float xoff = 0.0;
float yoff = 0.0;
float step = 0.01;
loadPixels();

for (int y = 0; y < height; y++) {
   xoff = 0.0;
   for (int x = 0; x < width; x++) {
      color c = color(map(noise(xoff, yoff), 0, 1, 0,
255));
      pixels[x + y * width] = c;
      xoff = xoff + step;
   }
   yoff = yoff + step;
}
updatePixels();
```



myNoise

마치 구름을 보는 것 같습니다. 대리석 무늬라고 해도 좋겠습니다. 자연에서 흔히 볼 수 있을 법한 그림입니다.