05
반복문 연습

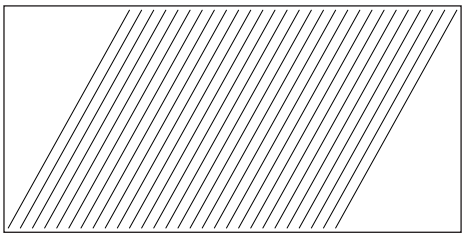
05.01
복잡한 패턴과 반복문

그럴듯한 결과를 얻기위해서 많은 것을 알아야 할 필요는 없습니다. 반복문을 활용해 여러가지 '그럴듯한' 작품을 그려봅시다. 여러분은 이미 충분히 많은 것을 알고 있습니다.

더 촘촘한 선

선을 더 촘촘하게 그어봅시다. 어려울 것은 없습니다. for 반복문의 조건식을 살짝 수정하면 for 반복문 안의 구문이 더 많이 반복되니까요.

```
void setup() {
  size(480, 360);
  smooth();
  background(255);
}

void draw() {
  for (int i = 0; i < 28; ++i) {
    strokeWeight(0.5);
    line(160 + i * 10, 90, 60 + i * 10, 270);
  }
}
```

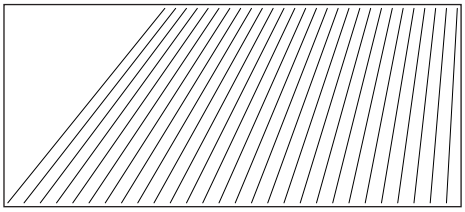


myFineLines

선 펼치기

나란히 놓인 선을 보니 훼방을 놓고 흐트리고 싶은 기분입니다. line() 함수의 입력값을 수정해 직선을 다시 위치해봅시다.

```
void setup() {
   size(480, 360);
   smooth();
   background(255);
}

void draw() {
   for (int i = 0; i < 28; ++i) {
      strokeWeight(0.5);
      line(160 + i * 10, 90, 15 + i * 15, 270);
   }
}
```

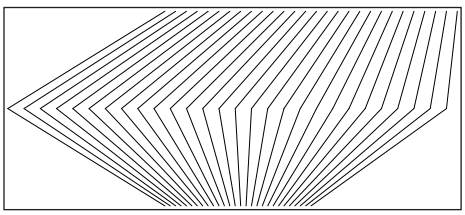


mySpreadLines

꺾은 선

for 반복문 안에 여러 구문을 실행하는 것은 이미 경험한 적이 있습니다. 복습하는 의미로 직선이 중간에 꺾이도록 반복문을 수정합시다.

```
void setup() {
   size(480, 360);
   smooth();
   background(255);
}

void draw() {
   for (int i = 0; i < 28; ++i) {
      strokeWeight(0.5);
      line(160 + i * 10, 90, 15 + i * 15, 180);
      line(15 + i * 15, 180, 160 + i * 5, 270);
   }
}
```



myBendingLines

05.02
이중 반복문

반복문이 이제 익숙해지려고 합니다. 지금까지 반복문을 이용해 도형을 그릴 때는 왼쪽에서 오른쪽으로 혹은 오른쪽에서 왼쪽으로 한 방향으로만 이동했습니다.

이제 좌우로 움직이는 동시에 위/아래로도 이동하며 도형을 그릴 때가 왔습니다. 이중 반복문을 이용하면 가능합니다.

이중 반복문은 반복문 안에 다시 반복문이 있는 것을 말합니다. 사실 반복문 안에 몇 개의 반복문이 들어있는 지에 대한 제한은 없습니다. 하지만 보통 2개를 상한선으로 하고 이보다 더 단계가 많아질 경우 예를 들어 재귀호출과 같은 방법을 사용하는 것이 일반적입니다. 2~3개 정도가 일반적으로 사람이 이해할 수 있는 한계로 보입니다.

이중 반복문의 기본 구조

이중 반복문은 아래와 같은 구조를 가집니다. 항상 반복되는 부분이니 사용할 때마다 같은 구조를 유지해 실수를 줄입시다.

```
for (int y = 0; y < yLimit; y++) {
   for (int x = 0 x < xLimit; x++) {
      ...
      function(x, y);
      ...
```

```
  }
}
```

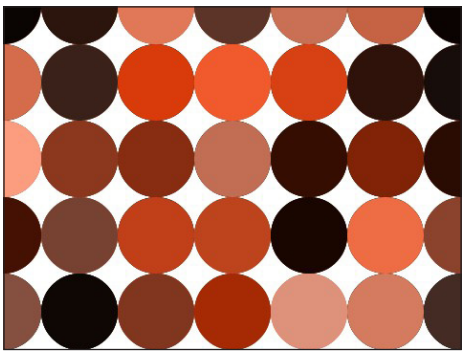
x와 y를 이런 순서로 사용하면 실수를 줄일 수 있습니다.

이제 실제 사례를 두고 연습해 봅시다.

원 그리기

화면을 원으로 채웁시다.

```
void setup() {
  size(480, 360);
  smooth();
  noStroke();
  background(255);
  ellipseMode(RADIUS);
  colorMode(HSB);
  frameRate(1);
}

float radius = 40;

void draw() {
  int nX = int(width / radius / 2.0);
  int nY = int(height / radius / 2.0);

  for (int y = 0; y <= nY; y++) {
    for (int x = 0; x <= nX; x++) {
      fill(10, random(100, 255), random(10,
255));
      ellipse(x * 2 * radius, y * 2 * radius, radius,
radius);
    }
  }
}
```



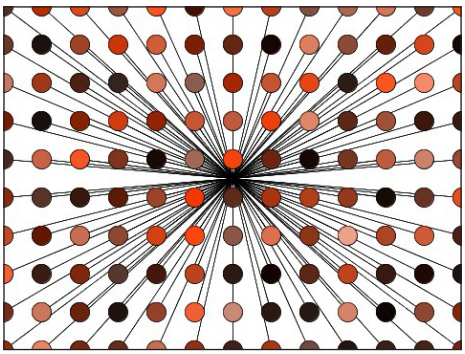
myCircles

중심과 선

화면의 중심으로 선이 모입니다.

```
void setup() {
  size(480, 360);
  smooth();
  background(255);
  ellipseMode(RADIUS);
  colorMode(HSB);
  frameRate(1);
}
```

```
float size = 20;

void draw() {
   int nX = int(width / size / 2.0);
   int nY = int(height / size / 2.0);

   for (int y = 0; y <= nY; y++) {
      for (int x = 0; x <= nX; x++) {
         strokeWeight(0.5);
         line(x * 2 * size, y * 2 * size, width / 2.0,
height / 2.0);
      }
   }

  for (int y = 0; y <= nY; y++) {
      for (int x = 0; x <= nX; x++) {
         fill(10, random(100, 255), random(10,
255));
         ellipse(x * 2 * size, y * 2 * size, 10, 10);
       }
   }
}
```

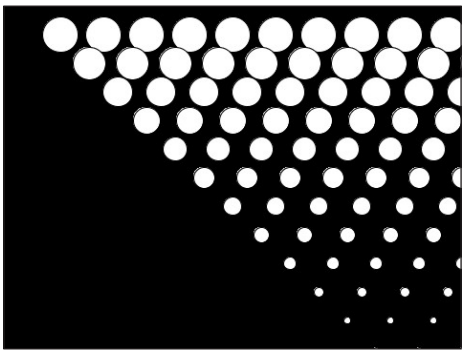


myConvergingLinesAndCircles

작아지는 점

for 반복문의 업데이트 부분에 주의를 기울여 주세요.

```
void setup() {
   size(480,360);
   smooth();
   background(0);
}

void draw() {
   for (int y = 30; y <= height; y = y + 30) {
      for (int x = 30; x <= width; x = x + 45) {
         ellipse(x + y, y, 40 - y / 10.0, 40 - y /
10.0);
      }
    }
}
```



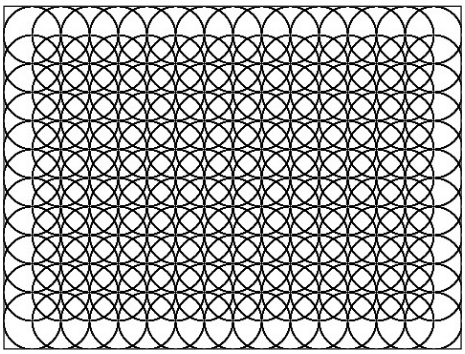
myDiminishingCircles

겹치는 원

for 반복문의 조건식에 주의를 기울여 주세요.

```
void setup() {
   size(480, 360);
   smooth();
```

```
  noFill();
  background(255);
  ellipseMode(RADIUS);
}

int radius = 30;

void draw() {
  for (int y = radius; y + radius <= height; y = y +
radius) {
    for (int x = radius; x + radius <= width; x = x +
radius) {
      ellipse(x, y, radius, radius);
    }
  }
}
```

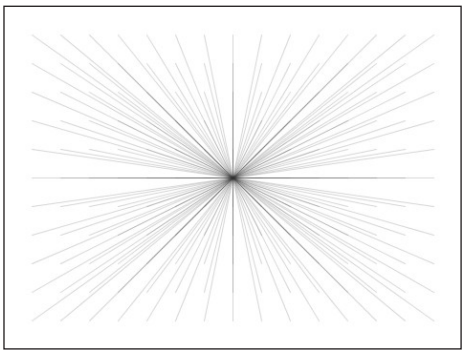


myOverlappingCircles

소실점

알파값을 사용할 때는 noLoop()를 함께 사용해야 원하는 결과를 얻을 수 있습니다.

```
void setup() {
  size(480, 360);
  smooth();
  background(255);
  noLoop();
}

int margin = 30;
void draw() {
  for (int y = margin; y + margin <= height; y = y +
margin) {
    for (int x = margin; x + margin <= width; x = x +
margin) {
      stroke(50, 50);
      line(x, y, width / 2.0, height / 2.0);
    }
  }
}
```

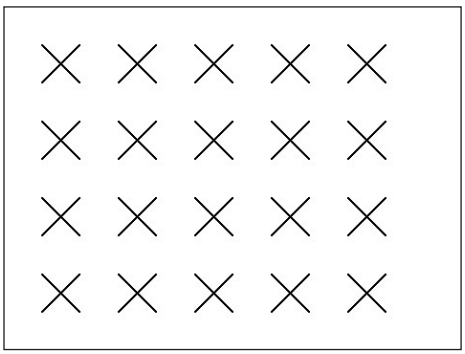


myPerspective

x표

```
void setup() {
  size(480, 360);
  smooth();
  background(255);
```

```
}

int grid = 40;

void draw() {
   for (int y = grid; y + 2 * grid <= height; y = y + 2 *
grid) {
      for (int x = grid; x + 2 * grid <= width; x = x + 2
* grid) {
         line(x, y, x + grid, y + grid);
         line(x + grid, y, x, y + grid);
      }
   }
}
```



myXMarks

미국식 횟수 세기

우리나라는 바를 정(正)을 이용해 횟수를 세지만 미국사람들은 다른 도형을 사용합니다. 화면을 미국식 마크로 채워봅시다.

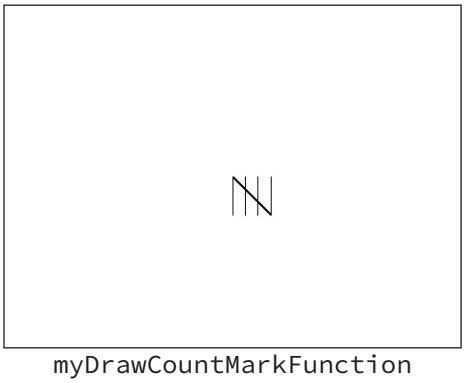
마크를 그리는 작은 함수를 정의해서 사용합시다.

원하는 임무를 수행하는 함수를 만드는 연습을 해보는 시간입니다. 함수는 되도록 작은 역할을 하도록 설계합시다. 더 크고 복잡한 일을 하는 함수는 작은 함수를 다시 조립하는 방법으로 프로그램의 살을 붙여나가면 실수를 줄일 수 있습니다. 만약 크고 복잡한 프로그램을 한 번에 만들여 보려고 한다면 반드시라고 해도 좋을 만큼 실패할 확률이 높습니다.

drawCountMark()

우선 미국식 마크를 그리는 함수부터 작성합시다.

```
void drawCountMark(float posX, float posY, float
width, float height) {
   float margin = width / 3.0;
   for (int i = 0; i < 4; i++) {
      line(posX + margin * i, posY, posX + margin * i,
posY + height);
   }
   line(posX, posY, posX + width, posY + height);
}
```



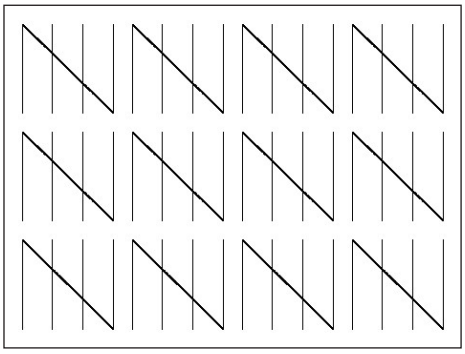
myDrawCountMarkFunction

```
void setup() {
   size(480, 360);
   smooth();
   background(255);
}
```

```
void draw() {
   drawCountMark(width / 2, height / 2, 40, 40);
}
```

이제 마크를 화면 가득 그려봅시다.

```
int nMarkX = 4;
int nMarkY = 3;

float margin = 20;

void setup() {
   size(480, 360);
   smooth();
   background(255);
}

void draw() {
   float markX = (width - margin * (nMarkX + 1)) /
nMarkX;
   float markY = (height - margin * (nMarkY + 1)) /
nMarkY;

   for (int y = 0; y < nMarkY; y++) {
      for (int x = 0; x < nMarkX; x++) {
         drawCountMark(margin + x * (markX + margin),
margin + y * (markY + margin), markX, markY);
      }
   }
}

void drawCountMark(float posX, float posY, float
width, float height) {
   float margin = width / 3.0;
   for (int i = 0; i < 4; i++) {
      line(posX + margin * i, posY, posX + margin * i,
posY + height);
   }
   line(posX, posY, posX + width, posY + height);
}
```



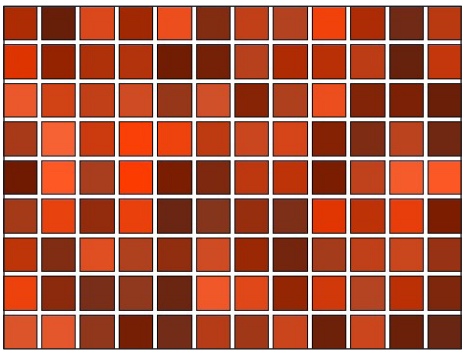
myDrawCountMarks

모로코 타일

모로코는 이슬람의 전통을 이어받아 아름다운 타일을 만드는 것으로 유명합니다. 반복문을 이용해 모로코식 타일을 그려봅시다.

타일의 색은 random()을 이용해 임의로 정합니다. 색을 정할 때 지금까지 사용했던 RGB모드를 사용하지 않고 HSB모드를 사용합니다. HSB

모드를 사용하면 비슷한 계열의 색을 선택하는데 편리합니다.


myDrawCountMarks

```
int nTileX = 12;
int nTileY = 9;

float margin = 5;

void setup() {
   size(480, 360);
   smooth();
   background(255);
   colorMode(HSB);
   noLoop();
}

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

   for (int y = 0; y < nTileY; y++) {
      for (int x = 0; x < nTileX; x++) {
         fill(10, random(200, 255), random(100,
255));
         rect(x * (tileX + margin), y * (tileY +
margin), tileX, tileY);
      }
   }
}
```

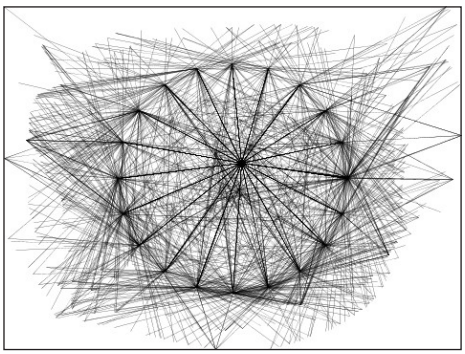
webby image

반복문을 정적인 이미지에만 사용할 이유는 없습니다. 이번에는 거미줄을 그려봅시다. 거미줄은 고정된 점에서 마우스의 현재 위치를 연결합니다. 일정거리 안에 들어올 때만 거미줄을 연결하고 항상 그리는 것이 아니라 확률을 가지고 연결합니다.

고정된 점의 위치를 저장하기 위해 배열을 이용합니다.

```
float[] posXs = new float[]{
360, 354, 337, 310, 277, 240, 202, 169, 142, 125,
120, 125, 142, 169, 202, 239, 277, 310, 337, 354,
360};

float[] posYs = new float[]{
180, 217, 250, 277, 294, 300, 294, 277, 250, 217,
180, 142, 109, 82,  65,  60,  65,  82, 109, 142, 179};
```

```
void setup() {
   size(480, 360);
   background(255);
   smooth();
}

void draw() {
   strokeWeight(0.1);
   for (int i = 0; i < posXs.length; i++) {
   if((0.9 < random(1)) && (dist(posXs[i], posYs[i],
mouseX, mouseY) < 180)) {
      line(posXs[i], posYs[i], mouseX, mouseY);
      }
   }
}
```



myWebbyImage