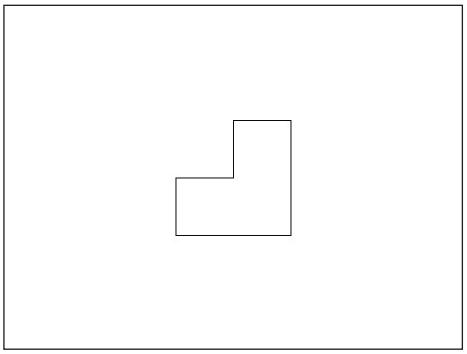
07
도형의 이동과 변형

07.01
도형의 이동

우선 괜찮은 도형을 하나 그려봅시다. 너무 간단하지도 복잡하지도 않은 도형이면 좋겠습니다. 적당히 비대칭이어서 어떤 방향을 향하고 있는지 알 수 있었으면 좋겠습니다. 음... 테트리스 블록 어떤가요? 테트리스 블록을 그려보겠습니다.

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

void draw() {
  background(255);

  beginShape();
  vertex(240, 180);
  vertex(240, 120);
  vertex(300, 120);
  vertex(300, 240);
  vertex(180, 240);
  vertex(180, 180);
  endShape(CLOSE);
}
```

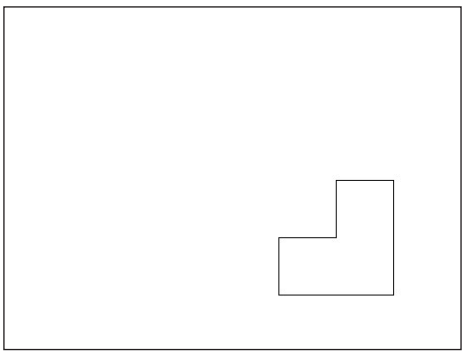


myTetris

도형 좌표를 이동하기

이 도형을 이동시키고 싶습니다. 어떻게 해야 할까요? 도형을 정의할 때 상수값을 사용했으니 이 도형을 옮길 방법이 마땅치 않습니다. 적어도 지금까지 배운 내용에서는 뾰족한 수가 없습니다. 이전에는 변수를 이용해서 아래와 같은 코드를 작성한 적이 있기는 합니다만 지금 원하는 답은 아닙니다.

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

void draw() {
   background(255);
   beginShape();
      float step = 60;

      float posX = mouseX;
      float posY = mouseY;
      vertex(posX, posY);

      posY -= step;
      vertex(posX, posY);

      posX += step;
      vertex(posX, posY);

      posY += 2 * step;
      vertex(posX, posY);

      posX -= 2 * step;
      vertex(posX, posY);

      posY -= step;
      vertex(posX, posY);
      endShape(CLOSE);
}
```



myTetrisRelativePosition

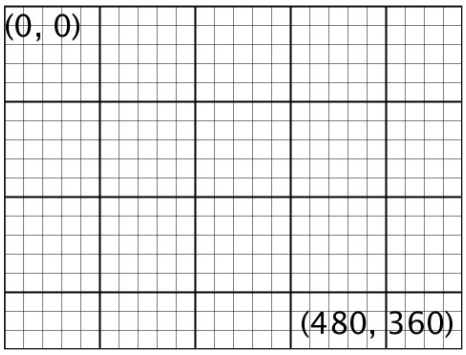
좌표 체계 이동하기

변수를 지정해서 옮기는 것이 유일한 방법일까요? 생각해보면 이 방법은 한계가 많습니다. 예를 들어 화면에 도형이 잔뜩있는 경우를 생각해봅시다. 만약 도형을 이동하고 싶다면 모든 도형의 위치를 변수로 연결해야 하고 이 변수를 이리저리 가공해야 합니다. 도형이 10개 정도 있다면 가능한 방법이지만 도형이 수천 개라면 포기하고 말겠습니다. 무언가 다른 방법이 필요합니다.

첫 시간에 프로세싱의 좌표체계에 대해 언급했습니다. 화면의 왼쪽 위가 (0, 0), 오른쪽 아래는 (width, height) 값을 가지게 됩니다.

복습도 할 겸 좌표 그리드를 한번 그려볼까요?

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

int step = 20;
void draw() {
   background(255);
   for (int y = 0; y <= height; y += step) {
      if(0 == (y % 100)) {
      strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(0, y, width, y);
   }

   for (int x = 0; x <= width; x += step) {
      if(0 == (x % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(x, 0, x, height);
   }

   textSize(32);
   fill(0);
   text("(0, 0)", 0, 32);
   text("(480, 360)", width - 170 , height - 16);
}
```



myCoordinateSystemGrid

좌표를 가만히 보고 있으니 아이디어가 떠오릅니다. 도형을 옮기는 대신 좌표 자체를 옮기는 것은 어떨까요? 종이 위에 그려진 그림을 이용하려고 할 때 새로 그리는 대신 종이 자체를 옆으로 이동하면 보는 사람 입장에서는 도형이 이동한 것과 같은 변화가 일어난 것입니다.

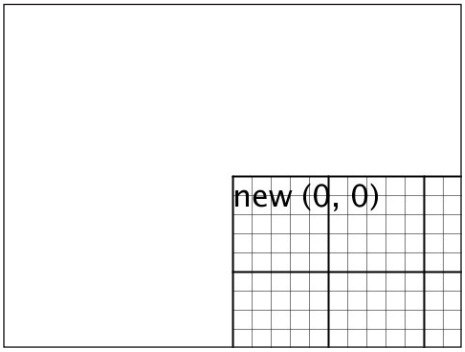
이와 마찬가지로 화면 위의 도형을 옮기는 대신 화면의 좌표 자체를 이동합시다.

좌표체계를 이용할 때는 translate() 함수를 사용합니다. translate() 함수는 원점을 새로 정하는 함수입니다.

translate() 함수는 입력된 값을 새로운 원점으로 삼습니다.

위의 코드에 translate() 함수를 적용해볼까요?

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

int step = 20;
void draw() {
   background(255);
   translate(width / 2.0, height / 2.0);
   for (int y = 0; y < height; y += step) {
      if(0 == (y % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(0, y, width, y);
   }

   for (int x = 0; x < width; x += step) {
      if(0 == (x % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(x, 0, x, height);
   }

   textSize(32);
   fill(0);
   text("new (0, 0)", 0, 32);
   text("(480, 360)", width - 170 , height - 16);
}
```

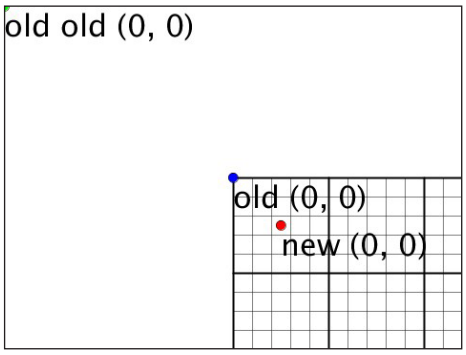


myCoordinateSystemGridTranslate

tranlae(width / 2.0, height / 2.0) 구문을 실행해 좌표 자체를 이동시키니 좌표에 그렸던 그림들이 모두 함께 이동합니다.

translate() 함수의 효과는 누적됩니다. 간단한 예제를 실행해 보겠습니다.

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

int step = 20;
void draw() {
   background(255);

   fill(0);
   textSize(32);
   text("old old (0, 0)", 0, 32);
   fill(0, 255, 0);
   ellipse(0, 0, 10, 10);

   translate(width / 2.0, height / 2.0);
   for (int y = 0; y <= height; y += step) {
      if(0 == (y % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(0, y, width, y);
   }

   for (int x = 0; x <= width; x += step) {
      if(0 == (x % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(x, 0, x, height);
   }

   fill(0);
   textSize(32);
   text("old (0, 0)", 0, 32);
   text("(480, 360)", width - 170 , height - 16);
   fill(0, 0, 255);
   ellipse(0, 0, 10, 10);

   translate(50, 50);
   fill(0);
   text("new (0, 0)", 0, 32);
```



myCoordinateSystemGridTranslateCumulative

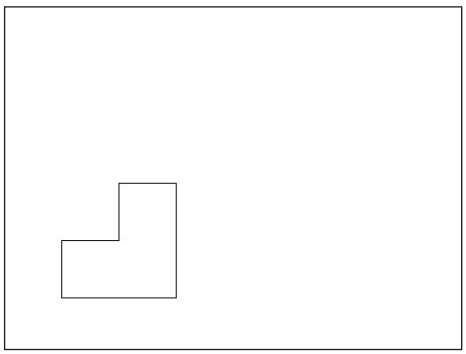
```
   fill(255, 0, 0);
   ellipse(0, 0, 10, 10);
}
```

원점 (0, 0)에 원을 그리라는 명령은 동일하지만 translate() 함수의 효과가 누적되어 다른 위치에 원이 그려집니다.

translate() 함수를 이용하면 도형의 이동을 쉽게 할 수 있습니다.

도형을 정의할 때 상수값을 사용해서 도형을 옮길 방법이 마땅치 않았던 경우를 기억하시나요? translate() 함수를 이용하면 이런 경우도 쉽게 다룰 수 있습니다.

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

void draw() {
   background(255);
   translate(mouseX - 240, mouseY - 180);

   beginShape();
   vertex(240, 180);
   vertex(240, 120);
   vertex(300, 120);
   vertex(300, 240);
   vertex(180, 240);
   vertex(180, 180);
   endShape(CLOSE);
}
```



myTetrisRevisit

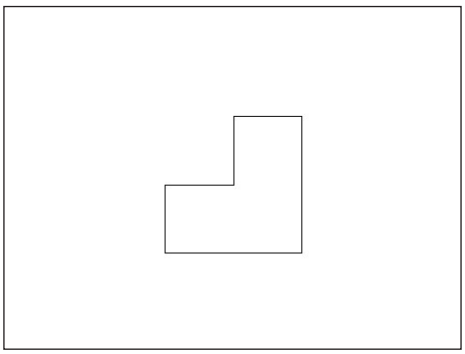
이제 translate() 함수를 적극적으로 사용할 것입니다.

07.02
도형의 확대

도형 크기 바꾸기

도형의 크기를 늘이려면 변수를 이용하면 가능합니다. 길이의 단위를 정해두고 시간에 따라 크기가 커지거나 작아지게 하는 것은 그렇게 어려운 일이 아닙니다.

```
//myVaryingSize
float step;
float offset = 0.0;

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

void draw() {
  background(255);
  step = map(noise(offset), 0, 1, 10, 110);

  beginShape();
  float posX = mouseX;
  float posY = mouseY;
  vertex(posX, posY);

  posY -= step;
  vertex(posX, posY);
```



myVaryingSize

```
  posX += step;
  vertex(posX, posY);

  posY += 2 * step;
  vertex(posX, posY);

  posX -= 2 * step;
  vertex(posX, posY);

  posY -= step;
  vertex(posX, posY);
  endShape(CLOSE);

  offset += 0.01;
}
  clickRotateX = rotateX;
  clickRotateY = rotateY;
}
```

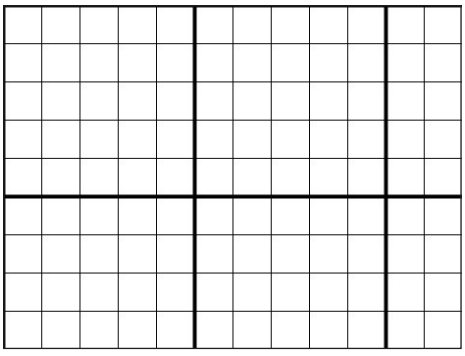
좌표 크기 바꾸기

이전에 설명한 것처럼 마찬가지로 모든 도형의 크기를 바꿔야 한다면 도형 하나하나의 크기를 나타내는 변수를 일일이 지정해줘야 합니다. 이건 프로그래머로서 참을 수 없는 불필요한 반복입니다. 어떻게 이 문제를 해결할 수 있을까요? 다시 좌표체계에서 답을 찾아봅시다.

scale() 함수를 사용하면 문제를 해결할 수 있습니다. scale()함수는 화면을 확대하거나 줄입니다. 실수를 입력으로 받습니다. 2.0은 화면을 두 배 확대하라는 뜻이고 1.5는 화면을 1.5배 확대하라는 명령입니다. 비유하자면 모눈종이의 눈금의 더 커진 종이 위에 도형을 그리는 것과 같습니다.

scale() 함수는 모든 것을 확대하고 줄입니다. scale(2.0)을 적용하면 도형의 획도 덩달아 확대되어 strokeWeight()을 실행한 것과 같은 효과를 가집니다.

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

int step = 20;
void draw() {
  background(255);
  scale(2.0);
  for (int y = 0; y < height; y += step) {
    if(0 == (y % 100)) {
      strokeWeight(2);
```

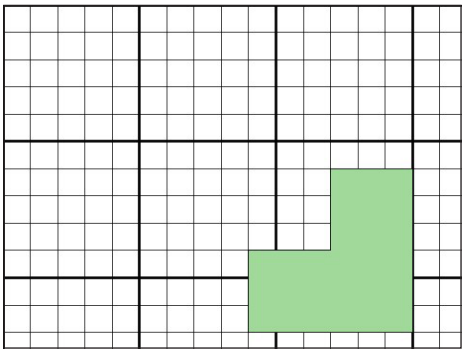


myScale

```
  }
  else {
    strokeWeight(0.5);
  }
  line(0, y, width, y);
  }

  for (int x = 0; x < width; x += step) {
    if(0 == (x % 100)) {
    strokeWeight(2);
  }
  else {
    strokeWeight(0.5);
  }
  line(x, 0, x, height);
  }
}
```

이제 이전 질문에 답할 차례입니다. 도형을 확대하려면 scale()을 이용하면 가능합니다.

```
//myVaryingSizeScale

float step = 60.0;
int grid = 20;
float zoom;
float offset = 0.0;

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

void draw() {
  background(255);
  zoom = map(noise(offset), 0, 1, 0.5, 4.0);
  scale(zoom);

  for (int y = 0; y < height; y += grid) {
    if(0 == (y % 100)) {
      strokeWeight(2);
    }
    else {
      strokeWeight(0.5);
    }
    line(0, y, width, y);
    }

  for (int x = 0; x < width; x += grid) {
    if(0 == (x % 100)) {
      strokeWeight(2);
```



myVaryingSizeScale

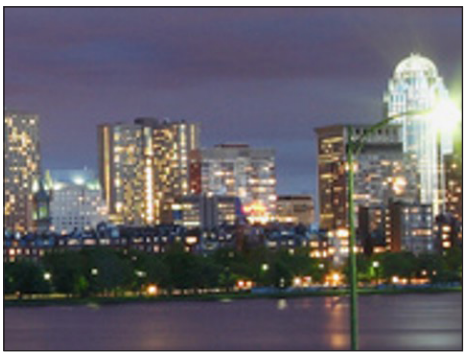
```
      }
      else {
         strokeWeight(0.5);
      }
      line(x, 0, x, height);
   }
   fill(#a1d99b);
   beginShape();

   float posX = width / 2.0;
   float posY = height / 2.0;
   vertex(posX, posY);

   posY -= step;
   vertex(posX, posY);

   posX += step;
   vertex(posX, posY);

   posY += 2 * step;
   vertex(posX, posY);

   posX -= 2 * step;
   vertex(posX, posY);

   posY -= step;
   vertex(posX, posY);
   endShape(CLOSE);

   offset += 0.01;
}
```

큰 이미지 파일 자세히 보기

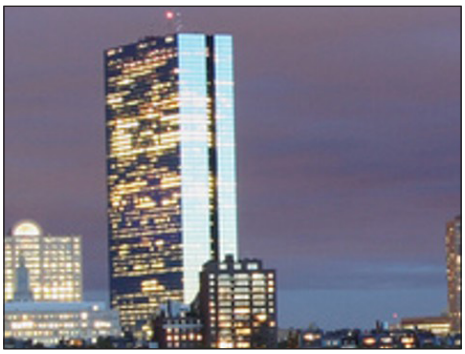
translate()와 scale()은 궁합이 잘 맞습니다. 두 함수를 함께 사용하면 관심있는 부분을 확대하고 사진 전체를 훑어보는 것이 가능합니다.

```
//myImage
PImage img;

float centerX;
float centerY;
float imageWidth;
float imageHeight;
float clickX;
float clickY;
float offsetX;
float offsetY;
float targetX;
```

```
float targetY;

float zoom = 1.0;

void setup() {
   size(480, 360);
   img = loadImage("image.jpg");
   imageWidth = img.width;
   imageHeight = img.height;

   centerX = width / 2.0;
   centerY = height / 2.0;
   imageMode(CENTER);
}

void draw() {
   translate(centerX, centerY);
   scale(zoom);
   image(img, 0, 0);
}

void mousePressed() {
   clickX = mouseX;
   clickY = mouseY;
}

void mouseDragged() {
   offsetX = mouseX - clickX;
   offsetY = mouseY - clickY;
   targetX = clickX + offsetX;
   targetY = clickY + offsetY;
   centerX += (targetX - centerX);
   centerY += (targetY - centerY);
   println("(" + centerX + ", " + centerY + ")");
}

void keyPressed() {
   if (key == 'i' || key == 'I') {
      zoom += 0.2;
   }
   if (key == 'o' || key == 'O') {
      zoom -= 0.2;
   }
   constrain(zoom, 0.1, 4.0);
}
```



myImage



myImage

07.03
도형의 회전

지금까지 도형은 모두 x축과 y축에 평행하게 이동했습니다. 이제 도형을 회전시켜 볼 차례입니다. 도형을 회전할 때는 각도라는 개념이 따라 나옵니다. 간단하게 각도에 대한 내용을 복습해 봅시다.

각도

일상 생활에서는 여러 방법으로 각도를 이야기 합니다. 자동차 운전을 할 때는 11시 방향이나 3시 방향이라고 이야기하는 경우도 있습니다. 또 다른 방법으로 360도를 한바퀴로 보는 방법이 있습니다. 180도 회전은 뒤로 돌아보는 것이고 90도 회전은 오른쪽으로 고개를 돌리는 것을 말합니다.

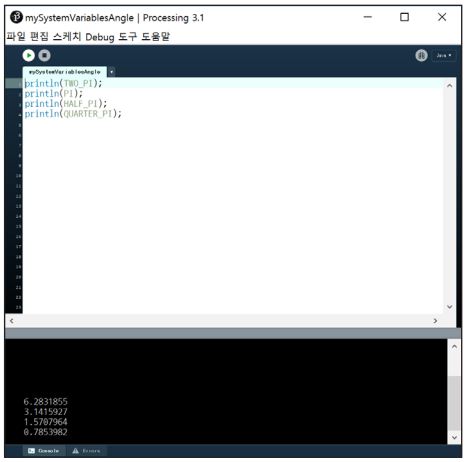
하지만 이런 각도 표기는 불편한 점이 있습니다. 각도를 정할 때만 특별한 단위를 사용해야 할 이유가 사실은 없습니다. 지금까지 우리가 연습했던 숫자를 각도계산에도 사용하고 싶습니다. 어떻게 하면 될까요?

공학과 과학에서 각도는 실수로 표현됩니다. 실수를 사용하는 것까지는 알겠습니다. 그러면 한바퀴를 실수 얼마로 잡아야 할까요? 편의상 한바퀴(360도)를 2π로 정의합니다. 그러면 반바퀴 각도는 π가 되고, 90도 회전은 π/2로 나타낼 수 있습니다.

2π, π, π/2, π/4 값은 자주 사용되기 때문에 프로세싱에서는 따로 시

스템변수로 지원합니다. 프로세싱 편집창에서 TWO_PI, PI, HALF_PI, QUARTER_PI 를 입력하면 초록색으로 하이라이트 되는 것을 확인할 수 있습니다.

```
//mySystemVariablesAngle
println(TWO_PI);
println(PI);
println(HALF_PI);
println(QUARTER_PI);
```



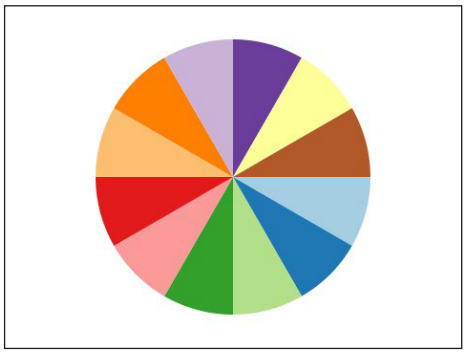
MySystemVariablesAngle

각도를 입력받는 프로세싱 함수는 모두 이 방법을 따릅니다. 그냥 가기 섭섭하니 간단하게 그림을 그려봅시다. arc() 함수의 기본 형식을 다시 한 번 떠올려 볼까요?

```
arc(posX, posY, width, height, begin, end);
```

begin과 end에는 각각 호가 시작하고 마치는 각도값이 입력됩니다. 이때 우리가 입력하는 값은 모두 radian 단위를 사용한 값입니다.

```
//myArcs
color[] colors = new color[]{#a6cee3, #1f78b4,
#b2df8a, #33a02c, #fb9a99, #e31a1c, #fdbf6f,
#ff7f00, #cab2d6, #6a3d9a, #ffff99, #b15928};
int nC = colors.length;
float radius;
float r = TWO_PI / float(nC);
void setup() {
size(480, 360);
   ellipseMode(RADIUS);
   noStroke();
   radius = min(width, height) * 0.4;
}

void draw() {
   background(255);
   translate(width / 2.0, height / 2.0);
   for (int i = 0; i < nC; i++) {
      float begin = r * i;
      float end = r * (i + 1);
      fill(colors[i]);
      arc(0, 0, radius, radius, begin, end);
   }
}
```

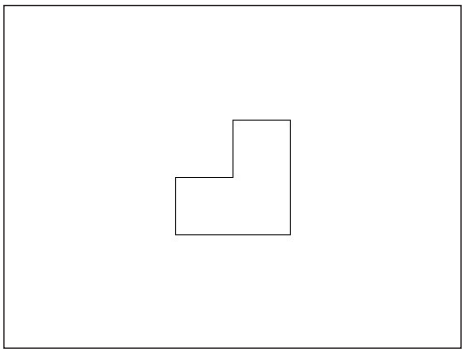

myArcs

도형 회전하기

각도에 대해 감을 잡았으니 이제 정말 도형을 회전시켜 봅시다. 위에서 한 것과 마찬가지로 먼저 도형 자체를 회전해봅시다. 도형 자체를 회전시키려면 약간의 산수가 필요합니다. 모든 설명을 이해할 필요는 없습니다. 도형 자체를 회전하려면 모든 꼭지점을 다시 계산해야 한다는 것만 기억하세요.

테트리스 블록이 다시 등장합니다.

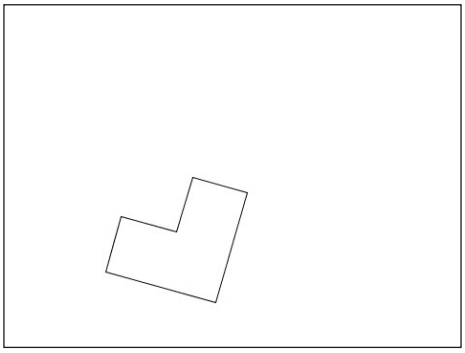
```
//myTetrisOriginal
void setup() {
  size(480, 360);
}

void draw() {
  background(255);
  beginShape();
  vertex(240, 180);
  vertex(240, 120);
  vertex(300, 120);
  vertex(300, 240);
  vertex(180, 240);
  vertex(180, 180);
  endShape(CLOSE);
}
```



myTetrisOriginal

이제 간단한 수학을 이용해 도형의 꼭지점 좌표를 변환합시다.

```
//myTetrisRotationMatrix
float theta = 0.0;
void setup() {
  size(480, 360);
}

void draw() {
  background(255);
  float c = cos(theta);
  float s = sin(theta);
  beginShape();
  vertex(c * 240 - s * 180, s * 240 + c * 180);
  vertex(c * 240 - s * 120, s * 240 + c * 120);
  vertex(c * 300 - s * 120, s * 300 + c * 120);
  vertex(c * 300 - s * 240, s * 300 + c * 240);
  vertex(c * 180 - s * 240, s * 180 + c * 240);
  vertex(c * 180 - s * 180, s * 180 + c * 180);
  endShape(CLOSE);

  theta += 0.01;
}
```



myTetrisRotationMatrix

하지만 이렇게 각 도형의 꼭지점을 모두 수정하는 것은 좋은 방법이 아닙니다. 도형의 이동에서 설명한 것처럼 도형의 갯수가 많아지면 이렇게 문제를 해결하는 것은 거의 불가능에 가깝습니다.

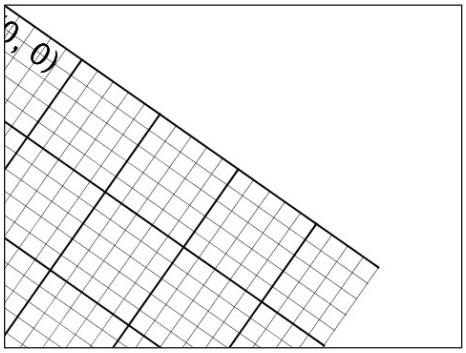
어떻게 해야 할까요? 답은 다시 좌표체계에 있습니다.

좌표 체계 회전하기

지금까지 translate(), scale()과 같이 좌표체계와 관련있는 함수를 배웠습니다. 원점을 이동시키고 원점을 기준으로 확대나 축소를 하는 방법을 배웠습니다. 이제 원점을 중심으로 좌표체계를 회전시키는 방법을 알아봅시다. rotate() 함수에 대해 배워봅시다.

rotate() 함수는 각도를 입력값으로 받습니다. 원점을 중심으로 각도만큼 시계방향으로 좌표체계를 회전시킵니다. 이전에 등장했던 그리드를 다시 한 번 그려보겠습니다.

```
//myCoordinateSystemGridRotate
float theta = 0.0;
int step = 20;
void setup() {
   size(480, 360);
}

void draw() {
   background(255);
   rotate(theta);
   for (int y = 0; y <= height; y += step) {
      if(0 == (y % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(0, y, width, y);
   }

   for (int x = 0; x <= width; x += step) {
      if(0 == (x % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(x, 0, x, height);
   }

   textSize(32);
   fill(0);
   text("(0, 0)", 0, 32);
```



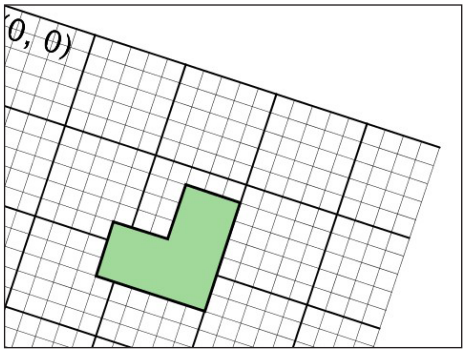
myCoordinateSystemGridRotate

```
   text("(480, 360)", width - 170 , height - 16);
   theta += 0.01;
}
```

좌표체계 자체가 원점을 중심으로 회전했으니 바뀐 좌표체계를 바탕으로 그린 도형 또한 모두 회전합니다.

이제 rotate() 함수를 이용해 테트리스 블록을 회전해 볼까요?

```
//myTetrisRotate
float theta = 0.0;
int step = 20;

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

void draw() {
   background(255);
   rotate(theta);

   for (int y = 0; y <= height; y += step) {
      if(0 == (y % 100)) {
      strokeWeight(2);
      }
      else {
      strokeWeight(0.5);
      }
      line(0, y, width, y);
   }

   for (int x = 0; x <= width; x += step) {
      if(0 == (x % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(x, 0, x, height);
   }

   textSize(32);
   fill(0);
   text("(0, 0)", 0, 32);
   text("(480, 360)", width - 170 , height - 16);

   fill(#a1d99b);
   strokeWeight(3);
   beginShape();
   vertex(240, 180);
```



myTetrisRotate

```
  vertex(240, 120);
  vertex(300, 120);
  vertex(300, 240);
  vertex(180, 240);
  vertex(180, 180);
  endShape(CLOSE);

  theta += 0.01;
}
```

rotate()를 이용하는 것이 훨씬 간편합니다. 가능하면 rotate() 함수를 자주 사용합시다.

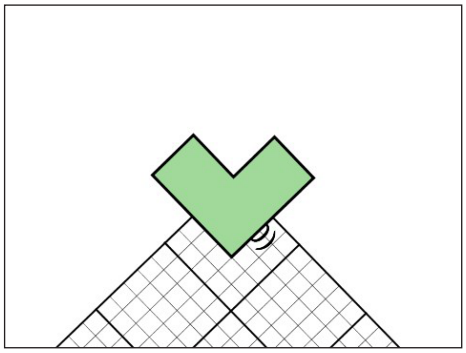
도형을 중심으로 회전하기

rotate()는 원점을 중심으로 좌표체계를 회전합니다. 그렇다면 도형이 스스로를 중심으로 회전하는 모습을 볼 수는 없는 걸까요? rotate() 함수 혼자서는 할 수 있는 방법이 없습니다.

하지만 translate() 함수와 함께 사용하면 어떨까요? translate() 함수로 원점을 도형의 중심으로 이동한 다음 rotate() 함수를 적용하면 도형이 혼자서 회전하지 않을까요?

새로운 코드가 추가되니 draw() 함수 내부가 번잡합니다. 그리드를 그리는 부분을 draw() 바깥으로 빼내서 setGrid() 함수로 묶어서 사용하겠습니다.

```
//myTetrisSpin
float theta = 0.0;
int step = 20;

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

void draw() {
  background(255);
  translate(width / 2.0, height / 2.0);
  rotate(theta);

  setGrid();

  //tetris block
  fill(#a1d99b);
  strokeWeight(3);
  beginShape();
  float step = 60;
  float posX = 0.0;
  float posY = 0.0;
```



myTetrisSpin

```
   vertex(posX, posY);

   posY -= step;
   vertex(posX, posY);

   posX += step;
   vertex(posX, posY);

   posY += 2 * step;
   vertex(posX, posY);

   posX -= 2 * step;
   vertex(posX, posY);

   posY -= step;
   vertex(posX, posY);
   endShape(CLOSE);

   theta += 0.01;
}

void setGrid() {
   //grid
   for (int y = 0; y <= height; y += step) {
      if(0 == (y % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(0, y, width, y);
   }

   for (int x = 0; x <= width; x += step) {
      if(0 == (x % 100)) {
         strokeWeight(2);
      }
      else {
         strokeWeight(0.5);
      }
      line(x, 0, x, height);
   }

   textSize(32);
   fill(0);
   text("(0, 0)", 0, 32);
   text("(480, 360)", width - 170 , height - 16);
}
```

07.04
하나 이상의 좌표체계: 톱니바퀴 따로 돌리기

이제 도형을 회전하는 일이 어렵지 않습니다. translate()를 이용해 회전시킬 도형의 중심으로 원점을 이동시킨 후 rotate() 함수를 이용하면 도형의 회전을 쉽게 할 수 있습니다.

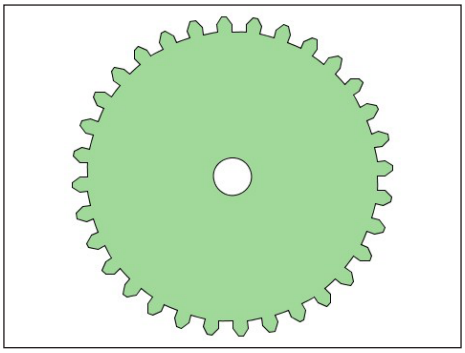
회전을 익히기 위해 하나 더 연습해보겠습니다. 이제 톱니바퀴를 돌려봅시다.

[참고 링크: https://bl.ocks.org/mbostock/1353700]

톱니바퀴 하나 그리기

톱니바퀴 SVG를 만들어서 화면에 띄운 후 회전시킵니다.

```
//myGearPlanet
PShape planet;
float theta = 0.0;

void setup() {
  size(480, 360);
  planet = loadShape("planet.svg");
  planet.disableStyle();
}

void draw() {
```



myGearPlanet

```
  background(255);
  translate(width / 2.0, height / 2.0);
  rotate(theta);
  fill(#a1d99b);
  shape(planet, 0, 0);

  theta += 0.01;
}
```

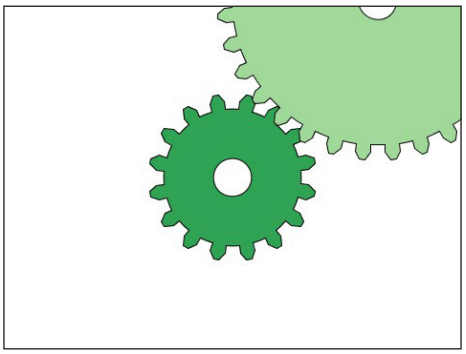
translate()와 rotate()를 결합하니 간단하게 도형을 회전할 수 있습니다.

톱니바퀴 두 개를 회전시키기

하나로는 성에 차지 않습니다. 톱니바퀴는 맞물려서 돌아야야 제대로 된 그림 아니겠습니까?

톱니 수가 적은 톱니바퀴를 하나 더 추가합니다. 작은 톱니바퀴는 16개의 톱니를 가지고 큰 톱니바퀴는 32개의 톱니바퀴를 가지고 있습니다. 따라서 큰 톱니바퀴는 더 천천히, 정확하게는 1/2의 속도로 회전해야 합니다. 이 부분을 추가로 수정했습니다.

```
//myGearPlanetAndSun
PShape sun;
PShape planet;
PShape annulus;
PShape solarSystem;
float theta = 0.0;

void setup() {
  size(480, 360);
  solarSystem = loadShape("solarSystem.svg");
  solarSystem.disableStyle();
  planet = solarSystem.getChild("planet");
  sun = solarSystem.getChild("sun");
}

void draw() {
  background(255);
  translate(width / 2.0, height / 2.0);
  rotate(theta);
  fill(#31a354);
  shape(sun, 0, 0);
  translate(240, 0);
  fill(#a1d99b);
  rotate(-theta * 16 / 32);
  shape(planet, 0, 0);

  theta += 0.01;
```



myGearPlanetAndSun

```
}
```

무언가 이상합니다. 톱니바퀴는 각자 회전해야 하는데 화면 가운데를 중심으로 두 톱니바퀴가 함께 돌고 있습니다. 이 문제를 어떻게 해결해야 할까요?

현재 사용하는 좌표체계를 저장할 수 있으면 좋겠습니다. 좌표체계가 바뀔 때마다 좌표체계를 저장하고 translate(), rotate()를 적용한 다음 더이상 쓸모없어지면 이전 체계로 돌아가면 문제가 해결될 것 같습니다.

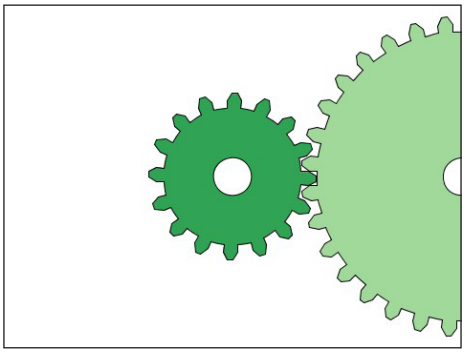
현재 좌표체계를 저장하는 함수가 pushMatrix()입니다. 반대로 현재 좌표체계를 버리고 이전 체계로 돌아가는 함수가 popMatrix()입니다.

좌표체계를 기억하는 것은 빨래통에 빨래를 집어넣는 것과 비슷합니다. 빨래통에 세탁물을 넣었다면(push) 세탁기에 빨래를 넣을 때(pop) 빨래통의 가장 위에 있는 의류부터 빼내야 합니다.

이 시스템은 편리한 점이 있습니다. 지금 어떤 좌표체계를 사용하고 있는지 추적할 필요가 없습니다. 가장 위에 있는 좌표체계만 확인하면 됩니다. 단지 지금 사용하는 좌표체계를 집어넣고(pushMatrix()) 사용이 끝나면 popMatrix()를 실행하면 됩니다.

실제 예제를 봅시다.

```
//myGearPlanetAndSunDualSystem
PShape sun;
PShape planet;
PShape annulus;
PShape solarSystem;
float theta = 0.0;

void setup() {
  size(480, 360);
  solarSystem = loadShape("solarSystem.svg");
  solarSystem.disableStyle();
  planet = solarSystem.getChild("planet");
  sun = solarSystem.getChild("sun");
}

void draw() {
  background(255);
  translate(width / 2.0, height / 2.0);
  pushMatrix();
    rotate(theta);
    fill(#31a354);
    shape(sun, 0, 0);
  popMatrix();
```



myGearPlanetAndSunDualSystem

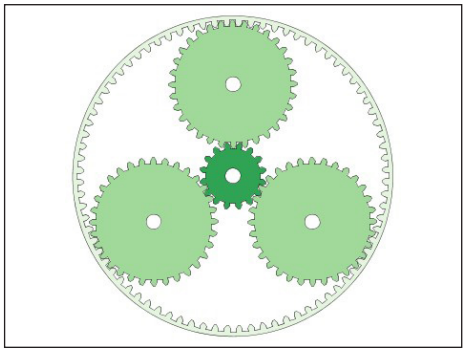
```
  translate(240, 0);
  pushMatrix();
    fill(#a1d99b);
    rotate(-theta * 16 / 32);
    shape(planet, 0, 0);
  popMatrix();

  theta += 0.01;
}
```

톱니바퀴 여러 개를 회전시키기

그냥 지나칠 수 없습니다. 조금 더 추가해 더 그럴듯한 그림을 그려봅시다. 여러분들은 이미 많은 것을 알고 있습니다. 약간의 상상력만 가미해 봅시다.

이제 톱니바퀴 여러 개를 동시에 회전시킵시다.

```
//myGearSolarSystem
PShape sun;
PShape planet;
PShape annulus;
PShape solarSystem;
float theta = 0.0;

void setup() {
  size(480, 360);
  solarSystem = loadShape("solarSystem.svg");
  solarSystem.disableStyle();
  planet = solarSystem.getChild("planet");
  sun = solarSystem.getChild("sun");
  annulus = solarSystem.getChild("annulus");
}

void draw() {
  background(255);
  translate(width / 2.0, height / 2.0);
  scale(0.4);

  //sun
  pushMatrix();
    rotate(theta);
    fill(#31a354);
    shape(sun, 0, 0);
  popMatrix();

  //planet01
  pushMatrix();
    translate(240 * cos(PI / 6), 240 * sin(PI / 6));
```



myGearSolarSystem

```
    fill(#a1d99b);
    rotate(-theta * 16 / 32);
    shape(planet, 0, 0);
  popMatrix();

  //planet02
  pushMatrix();
    translate(240 * cos(PI * 5 / 6), 240 * sin(PI * 5
/ 6));
    fill(#a1d99b);
    rotate(-theta * 16 / 32);
    shape(planet, 0, 0);
  popMatrix();

  //planet03
  pushMatrix();
    translate(240 * cos(PI * 9 / 6), 240 * sin(PI * 9
/ 6));
    fill(#a1d99b);
    rotate(-theta * 16 / 32);
    shape(planet, 0, 0);
  popMatrix();

  //annulus
  pushMatrix();
    fill(#e5f5e0);
    rotate(-theta * 16 / 80);
    shape(annulus, 0, 0);
  popMatrix();

  theta += 0.01;
}
```

popMatrix()와 pushMatrix()는 편리한 도구입니다. 손에 익도록 자주 사용합시다.