```
// Meditative Symbiosis. Jean Laffert . 2015
// example code of the graphic pattern designed to be projected over the surface of the plant.
// note: this example is only for demonstrate the graphic pattern, it does not include the
connection to sensors.
// This code is a variation based on the "Sutcliffe Pentagon" composition, by Matt Pearson
(see: Generative Art-Pearson 2011.p170-183)
// jdlaffert.com
//============================================================

FractalRoot pentagon;
float _strutFactor = 0.1;
float _strutNoise;
int _maxLevels = 3;

// ====================== main program ==================
void setup() {
  background (0);
  frameRate (5);
  size(800, 600);
  smooth();
  _strutNoise=random(10);
}

void draw(){

background (0);

_strutNoise+=0.01;
_strutFactor=(noise(_strutNoise)*3)-1;

pentagon=new FractalRoot(frameCount);
pentagon.drawShape();

}

// ======================== objects ======================

class PointObj {
  float x, y;
  PointObj(float ex, float why) {
    x = ex;
    y = why;
  }
}

class FractalRoot {
  PointObj[] pointArr = new PointObj[6];
  Branch rootBranch;

  FractalRoot(float startAngle) {
    float centX = width/2;
    float centY = height/2;
    int count = 0;
    for (int i = 0; i<360; i+=60) {
      float x = centX + (240 * cos(radians(i)));
      float y = centY + (240 * sin(radians(i)));
      pointArr[count] = new PointObj(x, y);
      count++;
    }
    rootBranch = new Branch(0, 0, pointArr);
  }

  void drawShape() {
    rootBranch.drawMe();
  }
}
```

```
class Branch {
 int level, num;
 PointObj[] outerPoints = {
 };
 PointObj[] midPoints = {
 };
 PointObj[] projPoints = {
 };
 Branch[] myBranches = {
 };

 Branch(int lev, int n, PointObj[]points) {
  level = lev;
  num = n;
  outerPoints = points;

  midPoints = calcMidPoints();
  projPoints = calcStrutPoints();

    if ((level+1)< _maxLevels) {
    Branch childBranch = new Branch(level+1, 0, projPoints);
    myBranches = (Branch[])append(myBranches, childBranch);

    for (int k=0; k<outerPoints.length; k++) {
      int nextk = (k+4)%outerPoints.length;
      PointObj[] newPoints = {
        outerPoints[k], midPoints[k], projPoints[k], projPoints[nextk], midPoints[nextk]
      };
      childBranch = new Branch(level+1, k+1, newPoints);
      myBranches = (Branch[])append(myBranches, childBranch);
    }
   }
  }

 void drawMe() {
  stroke(255,70);
  strokeWeight(3- level);
  for (int i = 0; i < outerPoints.length; i++) {
    int nexti = i+1;
    if (nexti == outerPoints.length) {
      nexti = 0;
    }
    line(outerPoints[i].x, outerPoints[i].y, outerPoints[nexti].x, outerPoints[nexti].y);
  }

  strokeWeight(0.1);
  fill(255,150);
  for(int j=0; j<midPoints.length; j++) {
   ellipse(midPoints[j].x, midPoints[j].y, 5, 5);
   line(midPoints[j].x, midPoints[j].y, projPoints[j].x, projPoints[j].y);
    ellipse(projPoints[j].x, projPoints[j].y, 5, 5);
   }
  for(int k=0; k<myBranches.length; k++) {
    myBranches[k].drawMe();
  }
 }


 PointObj[] calcMidPoints() {
  PointObj[] mpArray = new PointObj[outerPoints.length];
  for (int i = 0; i < outerPoints.length; i++) {
    int nexti = i+1;
    if (nexti == outerPoints.length) {
      nexti = 0;
    }
    PointObj thisMP = calcMidPoint(outerPoints[i], outerPoints[nexti]);
    mpArray[i] = thisMP;
```

```
  }
  return mpArray;
 }

 PointObj calcMidPoint(PointObj end1, PointObj end2) {
  float mx, my;

  if(end1.x>end2.x) {
   mx=end2.x + ((end1.x - end2.x)/2);
  }else{
   mx=end1.x+((end2.x-end1.x)/2);
  }
   if(end1.y>end2.y){
    my=end2.y+((end1.y-end2.y)/2);
   }else{
    my=end1.y+((end2.y-end1.y)/2);
   }
   return new PointObj(mx,my);
 }

 PointObj[] calcStrutPoints() {
  PointObj[] strutArray = new PointObj[midPoints.length];
  for(int i=0; i< midPoints.length; i++) {
   int nexti = (i+3)%midPoints.length ;
   PointObj thisSP = calcProjPoint(midPoints[i], outerPoints[nexti]);
   strutArray[i] = thisSP;
  }
  return strutArray;
 }

 PointObj calcProjPoint(PointObj mp, PointObj op) {
  float px, py;
  float adj,opp;
  if(op.x>mp.x){
   opp=op.x-mp.x;
  }else{
   opp=mp.x-op.x;
  }
  if(op.y>mp.y){
   adj=op.y-mp.y;
  }else{
   adj=mp.y-op.y;
  }
  if(op.x>mp.x){
   px=mp.x+(opp*_strutFactor);
  }else{
   px=mp.x-(opp*_strutFactor);
  }
  if(op.y>mp.y){
   py=mp.y+(adj*_strutFactor);
  }else{
   py=mp.y-(adj*_strutFactor);
  }

  return new PointObj(px, py);
 }
}
```