```
//Simbiosis Meditativa. Jean Laffert. 2015
//Código generador de patrón gráfico a proyectarse sobre la planta
//Este código es una variación de "Sutcliffe Pentagon" de Matt Pearson,
//extraído de: Generative Art, Pearson, Manning 2011, Pág. 170-183
//Para cargar el gráfico de datos de Co2, es necesario implementar la librería
//gwoptics.jar – descargarla en:
// http://jdlaffert.com/simbiosis-esp-informatica.html


import processing.serial.*;
//////gráfico valores de Co2:
  import org.gwoptics.graphics.graph2D.Graph2D;
  import org.gwoptics.graphics.graph2D.traces.ILine2DEquation;
  import org.gwoptics.graphics.graph2D.traces.RollingLine2DTrace;
  class eq implements ILine2DEquation{
  public double computePoint(double x,int pos) {
       return b;
  }
  }
  RollingLine2DTrace r;
  Graph2D g;
  Serial myPort;
        int[] serialInArray = new int[0];
        int serialCount = 0;
        boolean firstContact = false;
//////variables patrón hexagonal:
FractalRoot pentagon;
float _strutFactor =0.00001;//punto c del apartado
float _strutNoise;
float a;
float b;
int _maxLevels = 2;
//////programa/////////
void setup() {
  frameRate (10);
  size(1250, 775);
  smooth();
  _strutNoise=random(10);
   println(Serial.list());
   myPort = new Serial(this, Serial.list()[5], 9600);
   myPort.bufferUntil('\n'); //recibe dato desde arduino (sensor CO2)
//////////////// gráfico de valores de Co2:
     r  = new RollingLine2DTrace(new eq() ,1000,0.1f);
     r.setTraceColour(255,0,0);
     g = new Graph2D(this, 410,650, false);
     g.setYAxisMax(3000);
     g.addTrace(r);
     g.position.y = 50;
     g.position.x = 100;
     g.setYAxisTickSpacing(50);
}

void draw(){
background (0);
// SerialCallResponse:
String inString = myPort.readStringUntil('\n');
 if (inString != null) {
 inString = trim(inString);
  float inByte = float(inString);
 b = inByte; //punto c del apartado (dato desde el sensor de CO2).
 }
a = b /800;

//funciones de la figura:
_strutNoise+= 0.01;
```

```
_strutFactor=(noise(_strutNoise) /a);//punto c del apartado
pentagon=new FractalRoot(frameCount);// punto b del apartado
pentagon.drawShape();
noStroke();
fill(0);
//escribir en la cosnola, para identificar el valor de a = CO2:
   println(inString +" inST      "+
            a +" a    \t" +
            _strutNoise + " SN   " +
            _strutFactor + " SF   " + b );
//bloque fondo blanco para gráfico Co2:
   fill(255);
   rect(0,0,590,770);
   g.draw();
}
void serialEvent (Serial myPort) {
  } //arduino

//////objetos///////
class PointObj {
  float x, y;
  PointObj(float ex, float why) {
    x = ex;
    y = why;
  }
}
class FractalRoot {
  PointObj[] pointArr = new PointObj[6]; // punto a del enuciado (6 vértices para
formar un hexágono)
  Branch rootBranch;
  FractalRoot(float startAngle) {
    float centX = width-300;
    float centY = height-500;
    int count = 0;
    for (int i = 0; i<360; i+=60) { // punto a del enuciado (60º para formar un
hexágono)
      float x = centX + (240 * cos(radians(i)));// para modificar el tamaño de la
imagen
      float y = centY + (240 * sin(radians(i)));// idem
      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(255,250,200);
    strokeWeight(6);
    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);
    }
// las líneas desde los puntos medios:
    strokeWeight(2);
    fill(255,0,0);
    for(int j=0; j<midPoints.length; j++) {
     ellipse(midPoints[j].x, midPoints[j].y, 5, 5);//la posición de los puntos
medios en la línea base del hexagono
     line(midPoints[j].x, midPoints[j].y, projPoints[j].x, projPoints[j].y); //
las líneas a desprender desde el punto medio
      ellipse(projPoints[j].x, projPoints[j].y, 5, 5);//el punto final de la
línea
     }
    // dibuja la sub-estructura ramificada:
    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;
    /// inicio de los cálculos trigonométricos para la proyección de los puntos
medios:
    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);
    }
    /// fin de los cálculos trigonométricos para la proyección de los puntos
medios:
    return new PointObj(px, py);
  }
}
```