```
/*-------------------------------------------------------------

 Code for generating a Word Cloud Visualization
 Version 1.2 (30 March 2015)

 Author: Michael Goldwasser

 This implementation is based heavily on one originally provided by
 Ira Greenberg, Dianna Xu, and Deepak Kumar in
 Processing: Creative Coding and Generative Art in Processing 2

 However, my implementation is done in a purely procedural style.
 -------------------------------------------------------------*/

// configuration globals
int canvasSize = 800;
String inputFileName = "peterpan.txt";
int N = 150;                              // maximum number of words to display
float SCALE = 1.50;                       // used to alter overall render scale

// For convenience, we define our own composite data type
class Entry {
  String word;
  int freq;      // number of occurrences
  float size;    // font size
  float x;       // x-coordinate of center
  float y;       // y-coordinate of baseline
  float w;       // width of rendered word
  float ascent;  // height of rendered word above baseline
  float descent; // height of rendered word below baseline
  color c;       // color for rendered word
};

// list of unique words in the input text
Entry[] lexicon = new Entry[100000];
int numEntries = 0;

void setup() {
  // pick better looking font
  size(canvasSize, canvasSize);
  background(255);
  textFont(createFont("Times New Roman", 120));
  textAlign(CENTER, BASELINE);
  loadWords();
  orderWords();
  renderWords();
}

// load the words from the input file, removing stopwords
// and computing frequencies
void loadWords() {
  // let's begin by loading stop words to ignore
  String[] stopWords = loadStrings("stopwords.txt");
  String delimiters = " ,./?<>;:\"[{]}\\|=+-_()*&^%$#@!~";
  String[] lines = loadStrings(inputFileName);
  String rawText = join(lines, " ").toLowerCase();
  String[] tokens = splitTokens(rawText, delimiters);
  for (int j=0; j < tokens.length; j++) {
    if (!contains(stopWords, tokens[j])) {
      int k = find(tokens[j]);
      if (k == -1) {
        lexicon[numEntries] = new Entry();
        lexicon[numEntries].word = tokens[j];
        lexicon[numEntries].freq = 1;
        numEntries++;
      } else {
        lexicon[k].freq++;
      }
    }
  }
}
```

```
 72:
 73: // return the index at which given word can be found
 74: // within entry list, or -1 if its not found.
 75: int find(String word) {
 76:   for (int j=0; j < numEntries; j++) {
 77:     if (word.equals(lexicon[j].word)) {
 78:       return j;
 79:     }
 80:   }
 81:   return -1;
 82: }
 83:
 84: // determine whether given word is in list of strings.
 85: boolean contains(String[] list, String word) {
 86:   for (int j=0; j < list.length; j++) {
 87:     if (word.equals(list[j])) {
 88:       return true;
 89:     }
 90:   }
 91:   return false;
 92: }
 93:
 94: // select the N most frequently used words and place them
 95: // in the first N spots of the words array (in descending
 96: // order of frequency)
 97: void orderWords() {
 98:   int limit = min(N, numEntries);
 99:   for (int j=0; j < limit; j++) {
100:     // determine most frequent word in lexicon[j...limit-1]
101:     // and swap it to lexicon[j]
102:     int big = j;
103:     for (int k=j+1; k < numEntries; k++) {
104:       if (lexicon[k].freq > lexicon[big].freq) {
105:         big = k;
106:       }
107:     }
108:     if (big != j) {
109:       Entry temp = lexicon[j];
110:       lexicon[j] = lexicon[big];
111:       lexicon[big] = temp;
112:     }
113:   }
114: }
115:
116: // render the most frequent N words
117: void renderWords() {
118:   background(255);
119:   int limit = min(N, numEntries);
120:   float smallFreq = lexicon[limit-1].freq;
121:   float MIN_SIZE = SCALE * canvasSize / limit;   // minimum font size to use
122:
123:   for (int j=0; j < limit; j++) {
124:     setWordProperties(j, MIN_SIZE * lexicon[j].freq / smallFreq);
125:     placeWordSpiral(j);
126:     renderWord(j);
127:   }
128: }
129:
130: void setWordProperties(int j, float size) {
131:   textSize(size);
132:   lexicon[j].size = size;
133:   lexicon[j].w = textWidth(lexicon[j].word);
134:   lexicon[j].ascent = textAscent();
135:   lexicon[j].descent = textDescent();
136:   lexicon[j].c = color(random(127),random(127),random(127));
137: }
138:
139: void placeWordRandom(int j) {
140:   lexicon[j].x = random(0, width);
141:   lexicon[j].y = random(0, height);
142: }
```

```
143:
144: void placeWordRandomNoIntersect(int j) {
145:   do {
146:     placeWordRandom(j);
147:   } while (!clear(j));
148: }
149:
150: void placeWordSpiral(int j) {
151:   float cx = width/2, cy = height/2;
152:   float R = 0.0, dR = 0.2, theta = 0.0, dTheta = 0.5;
153:   do {    // find the next x, y for tile, i in spiral
154:     lexicon[j].x = cx + R*cos(theta);
155:     lexicon[j].y = cy + R*sin(theta);
156:     theta+=dTheta;
157:     R += dR;
158:   } // until the tile is clear of all other tiles
159:   while (!clear (j));
160: }
161:
162: // return true if word j placement is clear of word 0..j-1
163: boolean clear(int j) {
164:   for (int k=0; k < j; k++) {
165:     if (intersects(j, k)) {
166:       return false;
167:     }
168:   }
169:   return true;
170: }
171:
172: // determine if rendering for lexicon[j] and lexicon[k] intersect
173: // (that is, if bounding boxes intersect)
174: boolean intersects(int j, int k) {
175:   // the first word's bounding box
176:   float left1 = lexicon[j].x - 0.5 * lexicon[j].w;
177:   float right1 = lexicon[j].x + 0.5* lexicon[j].w;
178:   float top1 = lexicon[j].y - lexicon[j].ascent;
179:   float bot1 = lexicon[j].y + lexicon[j].descent;
180:   // the second word's bounding box
181:   float left2 = lexicon[k].x - 0.5 * lexicon[k].w;
182:   float right2 = lexicon[k].x + 0.5 * lexicon[k].w;
183:   float top2 = lexicon[k].y - lexicon[k].ascent;
184:   float bot2 = lexicon[k].y + lexicon[k].descent;
185:   // boxes intersect unless we find separating boundary
186:   return !(right1 < left2 || left1 > right2 || bot1 < top2 || top1 > bot2);
187: }
188:
189: // draw the word to the screen with given properties
190: void renderWord(int j) {
191:   textSize(lexicon[j].size);
192:   fill(lexicon[j].c);
193:   text(lexicon[j].word, lexicon[j].x, lexicon[j].y);
194:   // debugging to display bounding box
195:   //noFill();
196:   //rect(lexicon[j].x-0.5*lexicon[j].w, lexicon[j].y - lexicon[j].ascent,
197:   //     lexicon[j].w, lexicon[j].ascent + lexicon[j].descent);
198: }
199:
200: // add (very) basic user controls
201: void draw() { }
202:
203: void mouseClicked() {
204:   renderWords();   // re-render with new colors
205: }
206:
207: void keyPressed() {
208:   if (key == 's') {
209:     save("cloud.jpg");
210:   }
211: }
```