```
 1: /*
 2:  * A demonstration of text processing in the form of a
 3:  * speed-reading application, somewhat akin to Spritz.
 4:  *
 5:  * Author: Michael Goldwasser
 6:  */
 7:
 8: // contstants that can be tuned
 9: int WORDS_PER_MINUTE = 300;
10: int FONTSIZE = 70;
11:
12: //String sourcefile = "http://www.gutenberg.org/cache/epub/2591/pg2591.txt";
13: int NUM_SAMPLES = 9;
14: int num = int(1 + random(NUM_SAMPLES));
15: String sourcefile = "example" + num + ".txt";
16:
17: // globals used to store text and progress
18: String[] words;  // will be loaded from file
19: int cur=0;       // index of current word
20: boolean running = false;
21:
22: // global to help with font metric
23: float interwordGap;
24: float tickMarkX;      // x-coordinate of tick mark for placing word
25: float baseline;
26:
27: void setup() {
28:   noLoop();    // don't start drawing until window clicked
29:   frameRate(WORDS_PER_MINUTE/60.0);
30:   textSize(FONTSIZE);
31:
32:   int windowWidth = int(15 * textWidth("M"));
33:
34:   size(windowWidth, 3*FONTSIZE);
35:   background(255);
36:   strokeWeight(2);
37:
38:   interwordGap = (textWidth("OOOOOOOOO") - 10 * textWidth("O"))/9.0;
39:   tickMarkX = 0.33*width;
40:   baseline = 2*FONTSIZE - textDescent();
41:
42:   line(0, 0.1*FONTSIZE, width, 0.1*FONTSIZE);
43:   line(0, 2.9*FONTSIZE, width, 2.9*FONTSIZE);
44:   line(tickMarkX, 0.1*FONTSIZE, tickMarkX, 0.5*FONTSIZE);
45:   line(tickMarkX, 2.5*FONTSIZE, tickMarkX, 2.9*FONTSIZE);
46:
47:   loadTextFile(sourcefile);
48:   renderWord("Click to Begin");
49: }
50:
51: // used to (re)start the rendering of the prose
52: void mouseClicked() {
53:   if (words != null) {
54:     running = !running;
55:     loop();
56:     if (cur >= words.length) {
57:       cur = 0;  // start over
58:     }
59:   }
60: }
61:
62: void draw() {
63:   if (running) {
64:     renderWord(words[cur]);
65:     cur++;
66:     if (cur == words.length) {
67:       running = false;
68:       noLoop();
69:     }
70:   }
71: }
```

```
  72:
  73:
  74: // Display a single word to be read
  75: void renderWord(String word) {
  76:   noStroke();
  77:   fill(255);
  78:   rect(0, 0.5*FONTSIZE, width, 2*FONTSIZE);
  79:
  80:   // divide word into three portions (left, middle, right).
  81:   int divide = getRedIndex(word.length());
  82:   String left = word.substring(0,divide);
  83:   String middle = word.substring(divide, 1+divide);
  84:   String right = word.substring(1+divide);  // until the end
  85:
  86:   textAlign(CENTER);
  87:   fill(255,0,0);
  88:   text(middle, tickMarkX, baseline);
  89:
  90:   float pad = interwordGap + 0.5 * textWidth(middle);
  91:   fill(0);
  92:   textAlign(RIGHT);
  93:   text(left, tickMarkX - pad, baseline);
  94:   textAlign(LEFT);
  95:   text(right, tickMarkX + pad, baseline);
  96:
  97: }
  98:
  99:
 100: // Open a text file and break its text into individual words,
 101: // as separated by spaces. Those words are loaded into the
 102: // global 'words' array.
 103: void loadTextFile(String filename) {
 104:   String[] lines = loadStrings(filename);
 105:
 106:   // first, let's count the total number of words in document
 107:   int total = 0;
 108:   for (int j=0; j < lines.length; j++) {
 109:     total += splitTokens(lines[j]).length;
 110:   }
 111:
 112:   // Now let's resize global array of words and fill it
 113:   words = new String[total];
 114:   int w=0;  // index into words
 115:   for (int j=0; j < lines.length; j++) {
 116:     String[] pieces = splitTokens(lines[j]);
 117:     for (int k=0; k < pieces.length; k++) {
 118:       words[w] = pieces[k];
 119:       w++;
 120:     }
 121:   }
 122: }
 123:
 124: // Given a particular word length, return index of character that becomes red
 125: int getRedIndex(int length) {
 126:   return (length + 2) / 4;  // integer division
 127: }
 128:
```