**Text and Fonts in Processing**

**Carson Foster**

We’ve spent a lot of time discussing functions to display shapes and images on screen, and we’ve talked about printing to the console, but how do we put *text* on the screen? Do we have to use individual shapes to simulate letters? Thankfully, no, we do not. In this lesson, we’ll examine the functions for drawing text and using fonts in Processing.

Displaying Basic Text

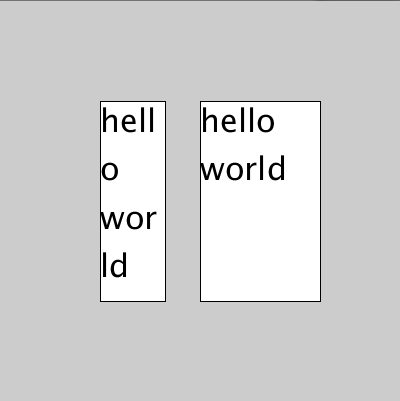
Let’s hop right in: to display plain Jane text, with the default sans-serif font and size, we can use the following function.



The first version of this method simply displays the text specified at the specific location. The coordinates can describe the text in different ways; using another method, we can change whether the coordinates tell Processing where to put the top-left corner, where to put the center of the textbox, or etc. By default, the coordinates describe the bottom-left corner.

The second version of the text method is different: it describes the actual bounding rectangle of the text to be displayed. The text is wrapped within the box: any text that would extend past the right edge is automatically put on the next ‘line’ in the bounding rectangle. This is word wrapping (wraps with regards to words instead of just characters), but Processing will wrap characters if the whole word can’t fit into one ‘line’. However, if text would pass the right and bottom edges of the bounding rectangle, it is simply not displayed.

Here’s an example that shows the basic behavior of text wrapping in Processing:



Here, I have printed “hello world” to the screen twice, and I’ve also drawn the bounding boxes for each. In the first example, you can see that Processing wraps the ‘o’ of ‘hello’ to the next line (character wrapping), since the whole word won’t fit onto the same line. However, in the next example, even though there is room in the first line for at least the ‘w’ of ‘world’, Processing refuses to character wrap, and instead does word wrapping.

As you saw in the example above, I changed the size of the text using another function. It’s pretty simple:



The one parameter is a float that represents the maximum height of each character, in pixels. This is different from the traditional way of measuring text size with units of ‘point’.

This next function is the aforementioned function that changes how the coordinates in text describe the text’s location.



The first parameter determines whether the x coordinate describes the left edge of the text, the right edge of the text, or the middle of the text. Thus, the possible values are LEFT, CENTER, and RIGHT. The second parameter determines whether the y coordinate describes the top edge of the text, the bottom edge of the text, or the middle of the text. The possible values are TOP, CENTER, and BOTTOM.

The final function we’ll look at in this section is a utility function that returns the width of a string (as a float), if it were to be drawn to the screen:



Fonts

Fonts are pretty simple in Processing. First, you have to load a font file into a Processing object, and then you set the font before you call text. The font must either be installed on the machine, or a .otf or .ttf file on the disk (if it’s in the data directory, you don’t have to use an absolute path). To find out which fonts are installed and can be used without a font file, use the following.



To create a font object and load a font, use a statement in the form:



Notice that this is very similar to how we constructed PImages earlier. After a font has been loaded, all that is necessary to use it is to call the following function (with the font object as the parameter).



Now you can use fonts! If you also want to change the color of the text, you can use fill. However, note that stroke does not work on text.



