Strings

Strings are a container for text. Specifically, a string is a string of characters, numeric codes for letters.These codes used to be a mere 8 bits to encompass Latin letters and a few special items, but now we have Unicode with more than a million possibilities for Arabic through Yijing. We still use the simple set (known as ASCII) in the key variable.

Strings use the same mechanism as array. That has evolved from a simple block of memory to a specialized class of objects with methods particularly useful for manipulating text.

Processing uses a modified version of the java string class, combining features of String for phrases of fixed length, and stringBuffer, which has features for extensions and deletions. Thus we can get away with:

```
String theString = "This is a short string.";
println(theString);
```

followed by

```
theString = "This is a somewhat longer string";
println(theString);
```

with no errors.

The Processing documents and texts don't talk much about strings. Instead they refer us to the rather intimidating official Java documentation. In fact, few authors have much to say, apparently expecting strings to have been covered in some more basic programming class.

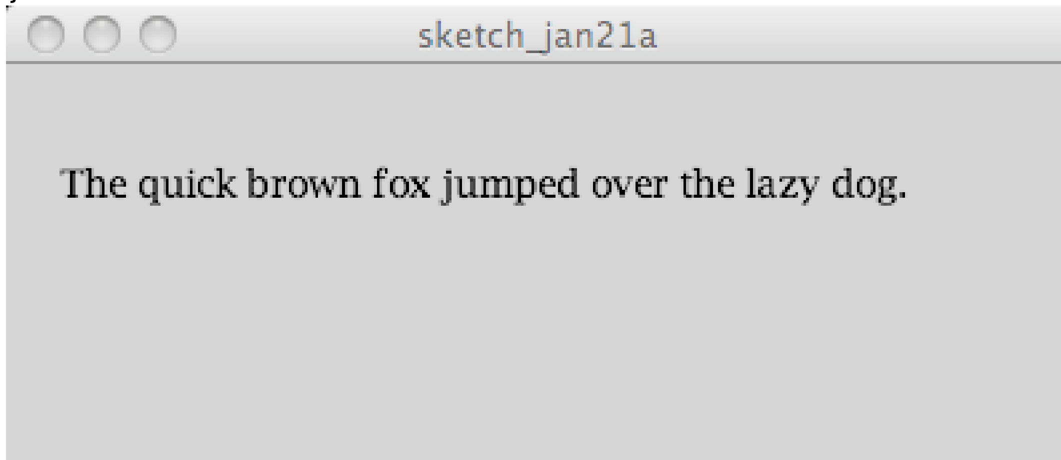
Displaying Strings

The first step in putting type into the processing window is to convert a font into the Pfont format. This is done with the "Create Font.." tool, and leaves a file called something like "CenturyGothic-12.vlw" in the data folder of the sketch. You then need a global PFont variable for each font you intend to use. You connect the font to the variable by calling loadFont in your setup() function. The type will be the current fill color. The actual drawing call is text().

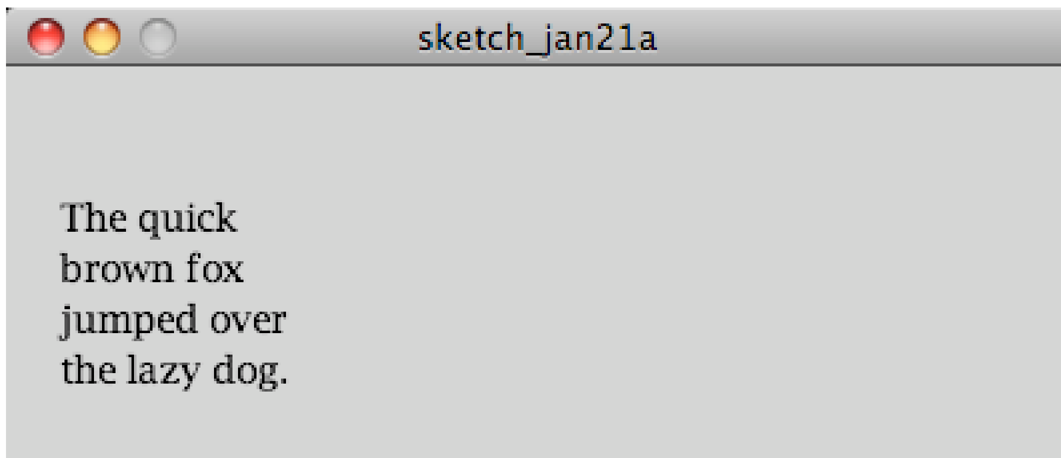
The first argument to text is what to display. Normally this is a string, but you can specify numbers or a char.
Next two arguments are the x,y, coordinates of the text. There is a textAlign() function to determine it the text is LEFT, CENTER, or RIGHT aligned with the x coordinate. The y is always the bottom of the (non-descending) letters. A third coordinate may specify z in

a 3D context, but normally two more numbers specify width and height of a text box to wrap the string into. When you specify width and height, the text is printed below the y coordinate. (You can specify width and height in 3D by tacking z onto the end.)

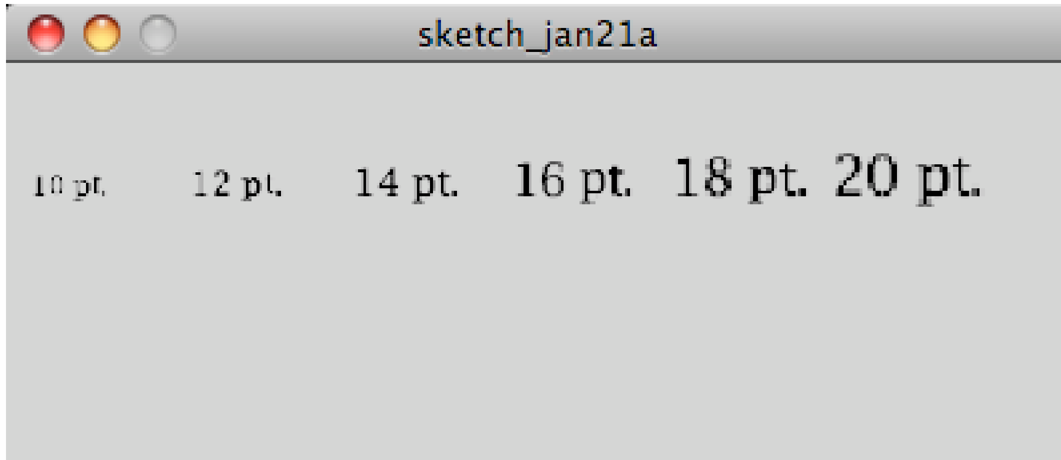
```
//simple type
PFont theFont;
String foxString = "The quick brown fox jumped over the lazy dog.";

void setup(){
 size(400,150);
 theFont = loadFont("LucidaBright-14.vlw");
 textFont(theFont, 14);
 fill(0);
 text(foxString, 20, 150);
}
```




```
text(foxString, 20, 50,100,100);
```



You will notice Create Font asks you to specify size, and textFont offers to change the size for you. It probably depends on the font, but I find that using textFont for changing size does not work well. This is probably because Create Font renders the font into a bitmap form, which is just a little picture of the letters. This includes anti-aliasing, so changing the size can be a bit strange. This is a 14 point font.



Building strings

Most of the strings used in art may be static, but we will need to change them from time to time.

Strings can be built by concatenation using the + or += operator.

```
str3 = str2 + str1;
```

Results in str3 containing str2 followed by str1.

```
str2 += str1
```

Will give the same string but in str2. When you combine strings like this you will have to pay attention to spaces to keep words from running together.

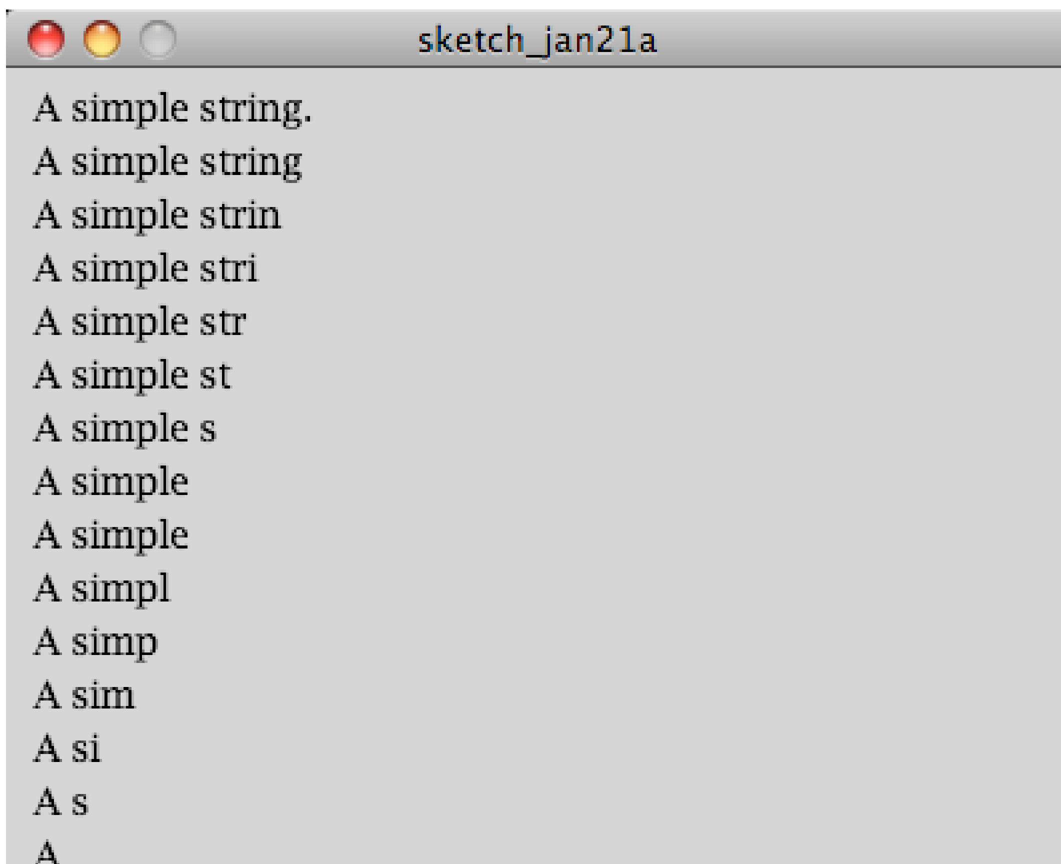
You can always get the length of a string by calling its length() method. Strings are objects, so methods are invoked with the dot:

str1.length()   will return the number of characters in str1.
str1.charAt(n) will return the character at n
str1.indexOf('n') will return the location of the first n in the string. It can also find a string within the string. If not found, a -1 is returned.
str1.indexOf('n',6) will return the location of the first n in the string, but won't start looking until location 6.

Substrings

The function str1.subString(start,end) returns a string containing the characters from start to end.

```
//Substrings
PFont theFont;
String theString = "A simple string.";

void setup(){
  size(400,300);
  theFont = loadFont("LucidaBright-14.vlw");
  textFont(theFont, 14);
  fill(0);
  int k = 0;
  for(int i = theString.length();i>0;--i){
    text(theString.substring(0,i),10,20 + k * 20);
    ++k;
  }
}
```



Notice the way **i** is set to the length of theString and decremented in the for loop. This is the classic method for counting backwards. Just make sure the condition uses > instead

of <. This code also needs a positive counter, **k**. We don't know when the loop ends, so we just increment k each time around.

Comparing Strings

We often need to know when two strings match. You can't use the usual == operation because it would just compare the variable names. The string method equals returns true if all the characters in the string and the comparison match.

```
if(str1.equals("test"))
```

will return true if str1 is "test". Here's a program that uses string comparisons to check for input commands.

```
//equals
PFont theFont;
String inString = "";

void setup(){
 size(400,300);
 theFont = loadFont("LucidaBright-14.vlw");
 textFont(theFont, 14);
 fill(0);
}

void draw(){
 background(255);
 text(inString, 20,20);
 if(inString.equals("square")) {rect(180,130,40,40);return;}
 if(inString.equals("circle")) {ellipse(200,150,40,40);return;}
}

void keyPressed(){
 if(key == BACKSPACE && inString.length()>0)
   inString = inString.substring(0,inString.length()-1);
 else
   if(key != CODED) inString += key;
}
```

The keyPressed routine demonstrates how to use backspace and keep non printing characters out of the input string. The tests are done in the draw() routine.

There are other string comparisons in Java. startsWith and endsWith are useful, and indexOf can detect anything if the result is compared with -1.

Split

Split breaks strings at a selected symbol. (The matching criteria can be quite complicated-- see the Java regular expression syntax). It returns an array of String objects. The most common use is to break a sentence into words as in this example:

```
//split
PFont theFont;
String strings[];
String foxString = "The quick brown fox jumped over the lazy dog.";

void setup(){
  size(400,200);
  theFont = loadFont("LucidaBright-14.vlw");
  textFont(theFont, 14);
  fill(0);
  strings = foxString.split(" ");
  for(int i =0; i< strings.length;++i){
   text(strings[i],20, 20+20*i);
  }
}
```

Note the use of strings.length instead of theString.length() we have been using a lot. Whereas the String object has a length() function, the Array object has a length member variable. Go figure.