Processing Notes 7

Chapter 15: Strings and Input/Output

So far we have been using Strings implicitly without much discussion. Strings variable hold a sequence of zero or more characters. For example:

```
String s1 = “This is simple ” ;
String s2 = “This is silly” ;
println(s1) ;
String s3 = s1 + s2 ;
println(s3) ;
```

would create two String variables, s1 and s2, assign “This is simple” and “This is silly” to s1 and s2 respectively, and then print out the contents of s1. The “+” operator when applied to strings concatenates the two operands together. Thus, variable s3 is assigned “This is simple This is silly” and hence that gets printed out next.

Sometimes it is desirable to convert integer and float values to strings. Processing provides the str( ) function to make this easy:

```
int num1 = 55 ;
float num2 = 44.44 ;
String s1 = str(num1) ;   // convert from int to string
String s2 = str(num2) ;   // convert from float to string
String s3 = "string representation of " + num1 + " and " + num2 ;
String s3b = s1 + s2 ;   // concatenate without a space!
println(s3) ;
println(s3b) ;
```

The above will print out:

```
string representation of 55 and 44.44
5544.44
```

Note, because there is no space in the assignment to variable s3b the string representation of the numbers is blurred together. You can also easily go the other way, i.e. from strings to numbers as in the following example:

```
String s4 = "44" ;
int num4 = int(s4) ;
if (num4 == 44)
  println("Yep, an int equal to 44") ;
```

```
String s5 = "44.54" ;
float num5 = float(s5) ;
if (num5 == 44.54)
  println("Yep, a float equal to 44.54") ;
```

Which prints out:

```
Yep, an int equal to 44
Yep, a float equal to 44.54
```

We can also access one character at a time using the charAt( ) method and the length of a string using the length( ) method as below. The code on the left produces the output on the right. Notice that the variables c1, c2, and c3 are of type *char* to hold single letter characters.

```
String s1, s2 ;

s1 = "Sue" ;
s2 = "I want chocolate." ;
println(s1.length() ) ;

char c1, c2, c3 ;

c1 = s1.charAt(0) ;
c2 = s1.charAt(1) ;
c3 = s1.charAt(2) ;

println(c1) ;
println(c2) ;
println(c3) ;

println("================") ;
for (int i = 0 ; i < s2.length() ; i++)
{
  c1 = s2.charAt(i) ;
  println(c1) ;
}
```

```
3
S
u
e
============
I

w
a
n
t

c
h
o
c
o
l
a
t
e
.
```

Sometimes, for example when doing input from a file, we may have a string of characters that we want to split apart into separate words. Consider the following code:

```
s1 = "31 44 56 77 99 121" ;
String[] numsAsStrings = split(s1," " ) ;   // the split token is space
println("Number elements = " + numsAsStrings.length) ;
```

```
for (int i = 0 ; i < numsAsStrings.length ; i++)
  println(numsAsStrings[i]) ;
```

Running this code would produce the following output:

```
Number elements = 6
31
44
56
77
99
121
```

The split function has two parameters, the first the string to be split apart, and the second the character to use as the split delimiter. A common delimiter is a space, another one is the ",". For example, the string:

"This,pig,big,laugh,wise"

Would make 5 elements if the split delimiter was the ",".

All this string manipulation becomes really handy when we are reading data from the user or from a file. First lets consider reading/writing data from/to files. The loadStrings ( ) and saveStrings( ) functions allow us to read/write from/to files. Consider the following example:

```
String[] input = loadStrings("inputData.txt") ;   // the split token is space
println("Number of lines of input data = " + input.length) ;
for (int i = 0 ; i < input.length ; i++)
  println(input[i]) ;
```

This will read all of the text in the file inputData.txt and put each line of data into a separate element of the array named "input". If the sketch has a data folder, the loadStrings() command looks in the data folder for a file named "inputData.txt". If found, it loads the data from that file. If not found, or there is no data folder, then Processing looks for a file named "inputData.txt" in the same directory as the .pde file.

The above code will echo the data from the file to the Processing output window. For example, if the file contains:

```
If I were a rich man
Ya ha deedle deedle, bubba bubba
```

Then those same two lines would be printed in the output window. The lines of text in the file "inputData.txt" are not broken apart into individual words. To do that we would need to use the split command:

```
String[] input ;
String[] temp ;
input = loadStrings("inputData.txt") ;
for (int i = 0 ; i < input.length ; i++)
{
  temp = split(input[i]," ") ;
  for (int j = 0 ; j < temp.length ; j++)
    println(temp[j]) ;
}
```

For the same data file it would print out:

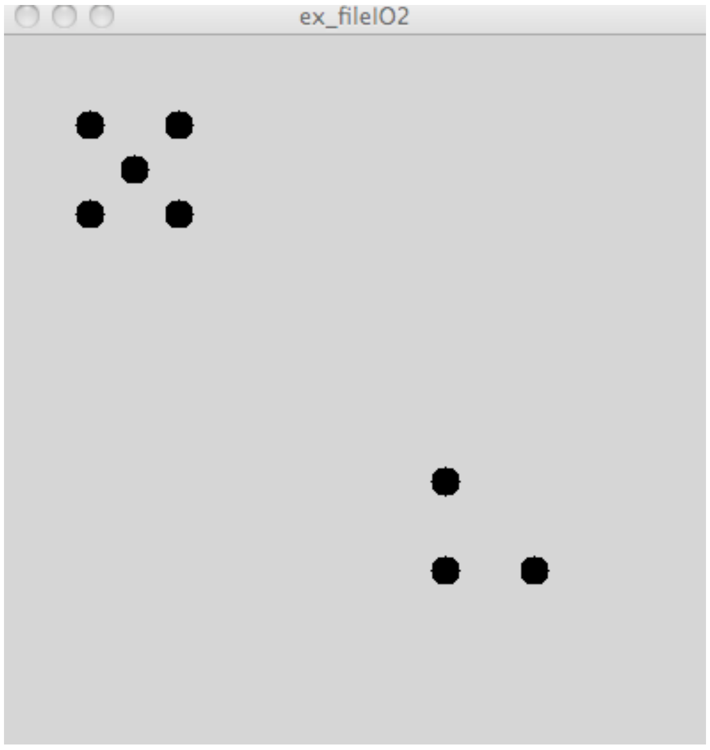
```
 If
 I
were
a
rich
man
Ya
ha
deedle
deedle,
bubba
bubba
```

Now that we can read in data, we can use that data in different ways. One simple way is to create a scatter-plot of data, i.e. a dot at every (x,y) pair specified in the input data file. For example, consider the following code:

```
void setup()
{
  size(400,400) ;

  String[] input;
  String[] temp ;
  int x, y ;

  fill(0) ;
  input = loadStrings("inputData.txt") ;
  for (int i = 0 ; i < input.length ; i++)
  {
    temp = split(input[i]," ") ;
    x = int(temp[0]) ;
    y = int(temp[1]) ;
```

```
        ellipse(x,y,15,15) ;
      }

    }
```

If the data file contains the data is in the below left, the output would be the image in the below right:

50 50
100 100
50 100
100 50
75 75
300 300
250 250
250 300
	

Not only can we load data from a file, we can also save data to a file. We do this using the saveStrings(filename, StringArray) command, where the first argument is the name of the file you want to write, and StringArray is the name of a variable that holds an array of strings. Consider the following code:

```
String[] outStrings ;
outStrings = new String[5] ;
outStrings[0] = "My momma talkn to me" ;
outStrings[1] = "tried to tell me how to live." ;
outStrings[2] = "Da da da, da da da, DA DA!" ;
outStrings[3] = "But I don't listen to her" ;
outStrings[4] = "cuz my head is like a SIEVE" ;
saveStrings("outData.txt",outStrings) ;
```

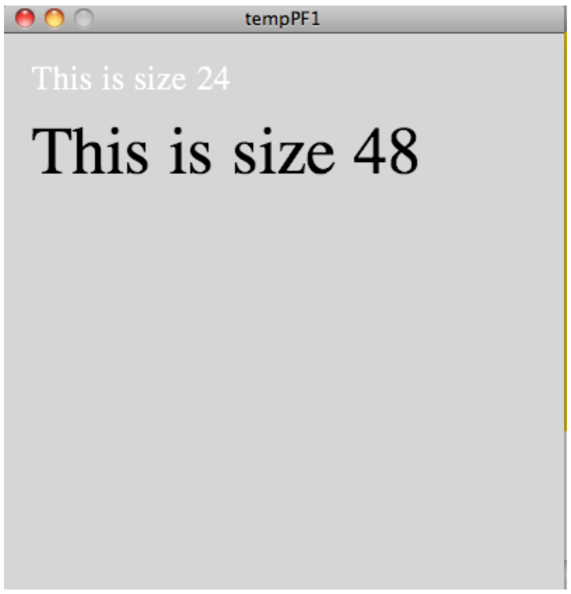
The effect of this code is to create a file name “outData.txt” and put the file lines of text in the array outStrings into the file. Note, if the file already exists the existing file is over-written with this data.

In addition to writing data to a file, we can also put data on the screen using the text command. The text(String, float, float) command puts the text contained in the the argument String at the x,y location specified by the two float arguments. But, before the text command can be used a font must be created and loaded. First we create the font. This is done using the Tools command,

Up at the top tool bar select “Tools” and then “Create Font”. Choose a font family and size, for example Serif-24. The font will be saved in your Sketch folder in the “data” subfolder. For this example it will create a file in data named “Serif-24.vlw”.

Now that the font is created you can load it in your Processing sketch. Once loaded you can use the font. Consider the following code (which assumes you have first created two fonts, Serif-24 and Serif-48) on the left below and its output on the right:

```
PFont font1, font2 ;

void setup()
{
  size(400,400) ;
  font1 = loadFont("Serif-24.vlw") ;
  font2 = loadFont("Serif-48.vlw") ;

  textFont(font1) ;
  text("This is size 24",20,40) ;

  fill(0) ;
  textFont(font2) ;
  text("This is size 48", 20, 100) ;
}
```



The command PFont font1, font2 before setup( ) declares two variables to hold fonts. Inside setup we first load fonts into the variables. If the fonts have not been previously created this command will fail. We can not use the command *textFont( )* to set a font, just as we would use *fill( )* to set the fill. Once textFont is set it is used for all subsequent text( ) commands until the textFont is changed. The *text(String)* command presents the text specified in the string at the x,y location specified in the command using the current textFont. Note, the x,y location will be the bottom left corner of the main body of the text, i.e. ignoring the letter's descenders.

Now that we understand how to put text on the screen, we can use the following code to put our beautiful song on the screen as follows:

```
PFont font1, font2 ;

void setup()
{
  size(400,400) ;

  String[] outStrings ;
  outStrings = new String[5] ;
  outStrings[0] = "My momma talkn to me" ;
  outStrings[1] = "tried to tell me how to live." ;
  outStrings[2] = "Da da da, da da da, DA DA!" ;
  outStrings[3] = "But I don't listen to her" ;
  outStrings[4] = "cuz my head is like a SIEVE" ;

  font1 = loadFont("Serif-24.vlw") ;
  textFont(font1) ;
  fill(0) ;

  for (int i = 0 ; i < outStrings.length ; i++)
    text( outStrings[i], 20, (i+1)*30) ;
}
```

In addition to reading/writing data from/to files, we can get data from the user. Consider the following code:

```
PFont font1 ;
String inputFromKeyboard ;

void setup()
{
  inputFromKeyboard = "" ;
  size(400,400) ;
  font1 = loadFont("Serif-24.vlw") ;
  textFont(font1) ;

  fill(0) ;
  text("Type something please.",20,30)  ;
}

void draw()
{
}

void keyReleased()
{
  background(150) ;
  char nextKey = key ;
  inputFromKeyboard += key ;
  text(inputFromKeyboard,20,30) ;
}
```

The keyReleased( ) function is a processing function that is called when the user depresses and then releases a key on the keyboard. The above code will first put the sentence “Type something please.” on the screen. When the user types a character that character is returned to the program by the command:

```
  char nextKey = key ;
```

where nextKey is declared to be a variable of type *char* , which is a variable that can hold one character. The word “key” is a reserved word in Processing that returns the value of the key just typed on the keyboard. The char is then appended to the string inputFromKeyboard which is then written to the screen.

If you hit return you will see that inputFromKeyboard includes this return.

Exercise 15A

Write a program that interacts with the user to read in keyboard input and echo it to the screen. When the user types the exclamation key, i.e. key == '!', the history is written out to a file named history1 and the history string is reset to the empty string, i.e. it clears the output. Then next time the user hits '!' the history is written out to a file named history2 and everything is reset, and so on.

For example, if I type in:

```
Little Bo Peep
Had lost her sheep
and ! doesn't know
where to find them.!
```

There will be two files created, history1 and history2. File history1 will have three lines of text with the last line have the word "and ". File history 2 will have two
lines of text with the first line being "doesn't know" and the second being "where to find them."

EXERCISE 15B

Write a program that reads a data file that represents rectangles, circles, and lines and draws the picture for that file. Here is an example data file:

```
r 10 50 20 20
c 100 100 20 20
c 150 150 30 30
l 10 50 100 100
l 100 100 150 150
```

For this datafile one rectangle will be drawn: rect(10,50,20,20) ; two circles will be drawn: ellipse(100,100,20,20), ellipse(150,150,30,30) ; and two lines will be drawn: line(10,50,100,100) , line(100,100,150,150)

Create your own data file to test your code. Start with just one line in your data file with no “r” “c” or “l” and get that to work first. Then get it to work for multiple lines. Then get it to work including the “r” “c” and “l”. Hint: use the charAt( ) method to get the single character “r” “c” or “l” and store in a char variable.

EXERCISE 15C

Write a program that collects the location of mouse-clicks, draws a line between each point of mouse click, and stores the (x,y) coordinates in an output data file one line per x,y pair. You can store the points as an array of x coordinates and a second array of y coordinates, or, as a single array where every even cell is the x and every odd cell is the y, or create a Point class and create an array of point objects.