# CS100: Project 3

Revision Date: October 2, 2015

## PREAMBLE

*The world today doesn't make sense, so why should I paint pictures that do?* – Pablo Picasso

Much of life is straightening out messes. Some messes others create; most messes are of your own creation. Your task today is to straighten out somebody else's mess, namely, a bunch of image data has become scrambled across two files. Your task is to merge the two files and reconstruct the image. To do this task, you will need to understand:

- reading records
- two-dimensional arrays
- ASCII graphics

## AN EXAMPLE

Suppose you are to reconstruct a 3x3 image and the two files you are to merge are composed of the following records:

```
23 A
42 7
199 .
214 x
```

and

```
9  s
53 $
77  O
201 z
211 .
```

Note that each record is composed of a numeric key and an ASCII character. Note further that the records in any given data file are sorted in ascending order by their key values.

The first thing to do is to merge the ASCII characters into an array, ordered by their keys. This would yield the following array:

```
[[9,'s'],[23,'A'],[42,'7'],[53,'$'],[77,'O'],[199,'.'],[201,'z'],[211,'.'], [214,'x']]
```

The next step is to place the characters in a 3x3 matrix (in this example), stripping out the keys. This should yield the following matrix:

```
[['s','A','7'], ['$','O','.'], ['z','.','x']]
```

Finally, the matrix is displayed, yielding the following output:

```
sA7
$O.
z.x
```

The actual names of the data files and the number of columns in the resulting matrix will be given as command-line arguments to your program.

## HANDLING SPACES

Space characters will be represented in the data files as the two character string `".."`. So, the record:

```
234 ..
```

means the second value in the record is the space character, while:

```
385 .
```

means the second value in the record is a period. You will need to translate the `".."` strings to spaces when displaying the resulting matrix.

## INPUT AND OUTPUT

The name of your program to assemble the image specified in the data files is to be named *assemble.py*. The program will take three command-line arguments, the names of the two data files, followed by the number of columns in the image.

Suppose the records in the previous section are stored in files *data1.rec* and *data2.rec*. Then, you would assemble and print the image with the command:

```
$ python3 assemble.py data1.rec data2.rec 3
sA7
$O.
z.x
```

## STEPWISE-REFINEMENT

The basic idea is you will have a number of python files, one containing a library of functions and the others using those functions. Note: it is *required* that you follow the steps outlined in this section.

**Task:** *reading* Write a function, that when given a file name, returns an array containing the records stored in that file. Place this function in a file named *support.py*. You will use this function to read the data in from the two data files. Write a program named *getdata.py* that tests this function. Note: your *getdata.py* program will import *support.py*. The name of the data file to be read will be given as a command-line argument to *getdata*.py. Here is an example use:

```
$ python3 getdata.py data1.dat
The file 'data1.dat' contains these records:
    [9, 's']
    [53, '$']
    [77, 'O']
    [201, 'z']
    [211, '.']
```

**Task:** *merging* Write a function that when given two ordered arrays of records, returns an ordered array obtained from merging the two arrays. Also place this function into *support.py*. Write a program named *merge-data.py* that tests this function. The two data files containing the records to be merged will be given as command-line arguments. Here is an example use:

```
$ python3 mergedata.py data1.dat data2.dat
The merged array of records in 'data1.dat' and 'data2.dat' are:
    [9, 's']
    [23, 'A']
    [42, '7']
    [53, '$']
    [77, 'O']
    [199, '.']
    [201, 'z']
    [211, '.']
    [214, 'x']
```

**Task:** *assembling* Write a function that takes a merged array plus the dimensions of a matrix and returns a matrix of the given dimensions filled with the characters in the merged array. Write another function that displays a matrix filled with characters. Place these functions in *support.py*. Test these function with your final program, *assemble.py*.

## TESTING YOUR FINAL IMPLEMENTATION

Download the following test files:

```
wget troll.cs.ua.edu/cs100/python/projects/part1.rec
wget troll.cs.ua.edu/cs100/python/projects/part2.rec
```

and run your program in a terminal window that you have maximized:

```
python3 assemble.py part1.rec part2.rec 200
```

Now make the font as small as possible so that there are no line wraps in your output. Change the font by modifying the terminal characteristics. On Ubuntu, it's *Edit -> Preferences -> Appearance -> Font*. If the image is still too big, here are some smaller versions:

```
wget troll.cs.ua.edu/cs100/python/projects/part1-small.rec
wget troll.cs.ua.edu/cs100/python/projects/part2-small.rec
```

The width for these smaller files is 100.

You can also redirect the output of your program into a file:

```
python3 assemble.py part.rec part2.rec 300 > output.txt
```

and view it with *vim*. Disable line wrapping in *vim* with the following command:

```
:set nowrap
```

Place the identity of the person in the image as a comment at the top of *assemble.py*.

## COMPLIANCE INSTRUCTIONS

Create two files named *test1.dat* and *test2.dat* that together contain 24 records. Your programs should run without error with the following calls:

```
python3 getdata.py test1.dat
python3 getdata.py test2.dat
python3 mergedata.py test1.dat test2.dat
python3 mergedata.py test2.dat test1.dat
python3 assemble.py test1.dat test2.dat 4
python3 assemble.py test1.dat test2.dat 3
```

Your program should print out clearly the results consistent with the call. If your *getdata* tests fail with a runtime error, then you will receive a zero for this assignment.

Note that your answers do not have to be correct for your program to be graded, only that the program does not crash. Of course, correct answers will yield a much higher grade.

## SUBMISSION INSTRUCTIONS

Change to the *project3* directory containing your assignment. Do an *ls* command. You should see something like this:

```
assemble.py  getdata.py  mergedata.py  support.py  test1.dat  test2.dat
```

Extra files are OK, as long as you are in the correct directory. (Submissions from the wrong directory that include **many** extra files will be penalized.) You will not be penalized if you submit files unrelated to this project.

Submit your program like this:

```
submit cs100 YYY project3
```

Remember to replace `YYY` with your section number.