# Programming, Probability, and the Modern Mathematics Classroom Exercises — Part 6

Manan Shah
Mathematician-At-Large

June 10, 2013

Please make sure to have read the blog post with the same topic name on the Math Misery website, otherwise this will be out of context.

## Some reminders

- Have students to comment the code, line by line.
- Let students tweak, modify, and otherwise alter the code.
- Don't worry about programming formalities.
- Have fun, relax, and learn.

## Introduction to file I/O

```
>>> ##In this session we will explore file I/O (input/output)
>>> ##We will work with strings, lists, sets, and dictionaries, as necessary
>>>
>>> ##First, create a folder called in "C:/", I called it "myBlog"
>>> ##Next, we will create a file to write to
>>> import os
>>> os.chdir("C:/myBlog")
>>> f.close()
>>> f = open("myfirstfile.txt","w")
>>> ##The "open" command with the argument "w" creates a file for writing
>>> ##"myfirstfile.txt" is the name of the file
>>> f.write("hello there!")
12
>>> ##the 12 that was returned from f.write is the number of characters written
>>> ##HOWEVER, if you notice, myfirstfile.txt has nothing yet written to it!
>>> ##This is because Python likes to write data to a file in chunks.
>>> ##There is not yet enough data to write, so the information is held
>>> ##in a buffer. Once enough information is ready to be written, Python will
>>> ##begin to write to file. So, let's see this happen.
>>> for i in range(1000):
    z = f.write("hello there times " + str(i+1) + "!\n")
```

```
>>> ##Now if we peak at the file (in windows you can do this just by opening it
>>> ##in notepad (don't use other programs because of some strange behavior
>>> ##that could happen via file locks and access permissions)),
>>> ##we will notice that information was written, but not all 1000 lines were
>>> ##written. Again this is because of how much is in the buffer.
>>> ##One way to force the information to be written is to "flush" the buffer.
>>> f.flush()
>>> ##Now peak at the file and you will see that everything is written.
>>> ##You will notice that we used the command "open" to open the file.
>>> ##When we are done with the file we should close it.
>>> f.close()
>>> ##Now, let's try to write to the file again. This means we have to open it
>>> ##again. WARNING: do not use open("myfirstfile.txt","w") as this will erase
>>> ##everything currently written in it!!!
>>> f = open("myfirstfile.txt","a")
>>> ##The "a" option is to append to the file
>>> f.write("new text via append\n")
20
>>> f.close()
>>> ##Now, let's say we want to read from this file.
>>> f = open("myfirstfile.txt","r")
>>> ##The "r" option is for reading
>>> f.read(1)
'h'
>>> ##This reads one byte and moves the cursor
>>> f.read(1)
'e'
>>> f.read(2)
'll'
>>> f.read(20)
'o there!hello there '
>>> f.read(100)
'times 1!\nhello there times 2!\nhello there times 3!\nhello there
 times 4!\nhello there times 5!\nhello t'
>>> ##Notice that while viewing the file in a text editor the new lines are
>>> ##shown as breaks. However, when reading via f.read we see "\n".
>>> ##"\n" is the "escape sequence" for a new line character. (Instructors step in
>>> ##with more examples about escape sequences, like \r,\t, etc.)
>>> ##Notice how we added "\n" to the string when we were writing to the file?
>>>
>>> ##If we wanted to go back to the beginning (or any position) of the file,
>>> ##we use the "seek" command.
>>> f.seek(0)
0
>>> ##This puts us back to the beginning of the file
>>> f.read(124)
'hello there!hello there times 1!\nhello there times 2!\nhello there
 times 3!\nhello there times 4!\nhello there times 5!\nhello t'
>>> f.seek(0)
```

```
0
>>> ##Python is a bit more friendly than this way of reading
>>> f.readline()
'hello there!hello there times 1!\n'
>>> f.readline()
'hello there times 2!\n'
>>> f.readline()
'hello there times 3!\n'
>>> ##Python is still friendlier than this.
>>> ##(We will use the "enumerate" command. this has nothing to do with file I/O
>>> ##but is something new, so take a few minutes to fiddle with enumerate.)
>>> f.seek(0)
0
>>> for linenumber, line in enumerate(f):
    if linenumber < 20:
        print("Line number " +str(linenumber)+ " reads: " +line, end='')


Line number 0 reads: hello there!hello there times 1!
Line number 1 reads: hello there times 2!
Line number 2 reads: hello there times 3!
Line number 3 reads: hello there times 4!
Line number 4 reads: hello there times 5!
Line number 5 reads: hello there times 6!
Line number 6 reads: hello there times 7!
Line number 7 reads: hello there times 8!
Line number 8 reads: hello there times 9!
Line number 9 reads: hello there times 10!
Line number 10 reads: hello there times 11!
Line number 11 reads: hello there times 12!
Line number 12 reads: hello there times 13!
Line number 13 reads: hello there times 14!
Line number 14 reads: hello there times 15!
Line number 15 reads: hello there times 16!
Line number 16 reads: hello there times 17!
Line number 17 reads: hello there times 18!
Line number 18 reads: hello there times 19!
Line number 19 reads: hello there times 20!
>>> ##Notice how the "\n" don't show up --- this is just a matter of display
>>> ##via the print command (that's why I used "end = ''").
>>>
>>> ##Again, once we are done working with the file, we should close it
>>> f.close()
>>>
>>> ##We saw earlier that f was of type _io.TextIOWrapper
>>> ##This allowed us to call .write, .read, .readline, .seek, .close, .flush
>>> ##There are other commands available as well
>>> f.name
'myfirstfile.txt'
>>> f.closed
True
```

```
>>> ##Some methods only work when the file is open
>>> f.fileno()
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    f.fileno()
ValueError: I/O operation on closed file
>>> ##Explore this a bit on your own.
>>>
>>> ##Now, instead of writing f = open(...) to open a file,
>>> ##we can use the "with" command.
>>> ##When we opened a file via f = open(...) we had to make sure to remember to
>>> ##close it. This can be dangerous if we forget to close the file
>>> ##or open it multiple times.
>>> ##The "with" command will automatically close the file once we break out of
>>> ##the "with" block.
>>> ##Here's how it works
>>> with open("mysecondfile.txt","w") as g:
    for i in range(1000):
        z = g.write("hi there, times " + str(i) + " this is my second file\n")


>>> g.closed
True
>>> ##Voila the file was opened for writing in the "with" block,
>>> ##and once we exited the with block, the file was closed.
>>> g.name
'mysecondfile.txt'
>>> ##This is file I/O in a nutshell.
>>> ##In the next lesson we'll get back into probability
```

## Summary

In this part, we discussed file I/O in Python. In the next lesson, we'll get back into probability. In the meanwhile, practice reading and writing from text files. See what works and what doesn't.

If you need help, have questions, or would like to set up a workshop at your school get in touch with me at help@mathmisery.com.