CSC104, Assignment 2, Winter 2006
Due: Thursday March 2nd, 11:59 pm

Danny Heap

Finger exercises

For this assignment we will be (mainly) working with the programming language Python. The exercises in this section are meant to guide you through learning how to use enough Python to work on the *lazyLit.py* program at the end of this assignment. As always, keep track of things you try (whether they work or not), in a journal.

1. Create a directory called *A2* (you might want to look back at *journalA1* to remember how to do this). Make *A2* your current working directory, using the *cd* command. Start the editor *Scite*, and begin a journal entry of your work on assignment 2 with the current date. Save this file as *journalA2*. Summarize all your work on assignment 2 in this journal.

2. You will be using the programming language Python. I will introduce some basic concepts in these exercises, and you may certainly post questions to the course wiki pages. There is also a tutorial on using Python at: http://docs.python.org/tut/tut.html, but you may well find that it is aimed at people with some previous programming experience. The same is true of the help available by typing (of all things) *help* once you have started up python.

   Start out by opening a terminal where you can type commands (as you did for *ls* and *df* in assignment 1). Now type *python*, and then press *enter*. You should see something like:

   ```
   Python 2.4.1 (#2, Aug 11 2005, 16:44:28)
   [GCC 3.3.3] on linux2
   Type "help", "copyright", "credits" or "license" for more information.
   >>>
   ```

   The >>> is indicating where you can type some expression that python will try to understand. Try typing something arithmetic, for example *1+2*, and then press *enter*. Addition and subtraction expressions should take a familiar form. Multiplication uses the * character, and division uses /. Try dividing 10 by 5. Now try dividing 9 by 5. Finally, try dividing 9 by 5.0 — that's 5, followed by a decimal point, followed by a zero. Explain your results in your journal. When you have had enough, hold down the "Ctrl" key with one finger, and then press the "D" key.

3. In assignment 1 you saw that you could store numbers, text, and even formulas in cells, and then refer to those cells by name: *D2* or *B3*. In Python you are given the freedom to decide on how to name "cells," provided you choose a name that begins with an alphabetic character and contains only alphabetic characters, numerals, and the underscore character (uppercase/lowercase is important).

Start up Python (see the previous exercise), and after the >>> type *favouriteNumber*, and then press the *enter* key. You should see an error message. Now type *favouriteNumber = 7*, press the *enter* key, then type *favouriteNumber* and press the *enter* key again. Now store a different number (not 7 again) in a location named *secondFavouriteNumber*. Type *favouriteNumber + secondFavouriteNumber*, then press *enter*.

Here's something a little tricky to puzzle over. Type *secondFavouriteNumber = favouriteNumber* and press *enter*. Now check what is stored in *favouriteNumber* and *secondFavouriteNumber* (you found out how to check what is in a location in previous exercises). Now type *favouriteNumber = 19* and press enter. Re-check what values are in the locations named *favouriteNumber* and *secondFavouriteNumber*. Remember to record all your observations in journalA2.

4. You can also store text, or strings of characters, in Python. After the >>> type *myWord = 'me'*, then press *enter*. The single quotes (or apostrophes) around *'me'* are important, to tip Python off that this is a string of characters, and not the name of some location, such as *favouriteNumber*. Now store the text *'you'* in a location named *yourWord*. Check what each location contains. Try typing *myWord + yourWord*, and then pressing *enter*. Sorry, that's all the arithmetic you can do with text!

5. After the >>> type *myList = [1, 3, 5, 7]*, then press *enter*. Check what is contained in the location named *myList*. Now type *myList[0]*, then press *enter*. What about *myList[1]* or *myList[3]*? Type *myList[0:3]*, then press *enter*. Explain your results as fully as you can.

   You can also makes list of words. After the >>> type *myWords = ['Double', 'double', 'toil', 'and', 'trouble.']*, then press enter. Check what the location named *myWords* contains. Type *fewerWords = myWords[1:3]* and see what the location named *fewerWords* contains. Also check what the location named *myWords* contains. Experiment until you can explain what's going on.

6. Try typing *oneWord = 'hand'*, then press *enter*, followed by *fewerWords + oneWord*, then press *enter*. Type *fewerWords.append('finger')*, then press *enter*, and check the value of stored at the location named *fewerWords*. Type *fewerWords = fewerWords * 2*, then press *enter*. Check what is stored at the location called *fewerWords*. At this point, you might want to chat with a TA or your instructor about what's going on.

7. Similar to a list is a tuple. After the >>> type *myTuple = ('one', 'two', 'three')*, then press *enter*. Check what the location named *myTuple* contains. Try out *myTuple[0]* and *myTuple[0:2]*. You might be tempted to think that tuples and lists are the same until you try *myTuple.append('four')*. Compare the effect of *fewerWords[0] = 'elbow'*, then press *enter*, to *myTuple[0] = 'elbow'*, then press *enter*.

   After the >>> type *tuple1 = ('one', 'two', 'three')* and press *enter*. Now type *tuple2 = ('four',)* and press *enter*. Finally type *tuple1[1:] + tuple2*. Experiment with their weird tuple arithmetic.

8. With both lists and tuples you could get an element if you knew its position, for example, in the last exercise *myTuple[0]* held the element 'one'. Sometimes it is more convenient to access elements without knowing their position, but some other key. Try typing *province = {'toronto' : 'ontario', 'vancouver' : 'british columbia', 'montreal' : 'quebec'}*, then press *enter*. Now type *province['toronto']*, then press *enter*. Try some other cities.

   In the example above, *province* is a look-up table: we look up the character string for a province by providing the character string for a city. Look-up tables are even more flexible than that. After the >>> type *bookTable = {('Alice', 'was', 'becoming') : ['very', 'to']}* and then press *enter*. In the look-up table *bookTable* you can retrieve the list of words ['very', 'to'] using the key ('Alice', 'was', 'becoming'). Try typing *bookTable[('Alice', 'was', 'becoming')]* and then pressing *enter*.

9. Time for some lazy repetition. After the >>> type: *numberList = [1, 3, 5, 7, 9]* and then press *enter*. Type *sum = 0* and then press enter. Type *nextNumber* and then press *enter* (what do you see?). Now type the following, being sure to leave the same spaces on the left-hand side as shown below (press *enter* twice to get back to the left-hand margin after the ...):

```
>>> for nextNumber in numberList:
...   print 'nextNumber = ', nextNumber
...   sum = sum + nextNumber
...   print 'sum = ', sum
...
```

You may want to chat with a TA or instructor about this. A key observation is that all three lines that are indented under *for nextNumber in numberList* are repeated.

If you're comfortable with the previous example, try typing *wordList = ['my', 'dog', 'has', 'fleas']* and then press *enter*. Just to convince yourself that there is no location (yet) named *nextWord*, type *nextWord* and then press *enter*. Now type the following, being sure to leave the same spaces on the left-hand side as shown below (press *enter* twice to get back to the left-hand margin after the ...):

```
>>> for nextWord in wordList:
...   print 'nextWord is', nextWord
...
```

10. Exit Python (by holding down the *Ctrl* key and pressing *D*). Make sure that *A2* is the current working directory (type *pwd* and then press *enter* to verify this). Now type *cp ~heap/pub/alice30.txt .* and press *enter*. This copies the text of "Alice in Wonderland" to your directory *A2*. You can examine it with the text editor *Scite*. Now we'll do a function definition. Type the following, being sure to leave the same spaces on the left-hand side as shown below. Press *enter* twice to get back to the left-hand margin after the ... (the ... are generated automatically by Python, just as the >>> are, so you don't type them).

```
>>> def readSomeLines(numLines):
...   bookFile = file('alice30.txt', 'r')
...   for nextLine in bookFile:
...     numLines = numLines - 1
...     print nextLine
...     print nextLine.split(), '\n' * 2
...     if numLines <= 0: break
...   bookFile.close()
```

After the >>> type *readSomeLines(3)*. Of course, you don't have to stop after 3 lines. This creates a function (program) called *readSomeLines* and the number you type in the parenthesis is stored in the location named *numLines* for the indented lines of Python. What is the difference between what's stored in *nextLine* and *nextLine.split()* at each step?

That's probably plenty of exploring for now. The next section explains the motivation for the *lazyLit* module, and the section following that explains how to complete the file *lazyLit.py* so that you can run the module.

Lazy literature

Throughout your life you are occasionally asked to write documents. Sometimes this task provides an outlet for creative expression, but sometimes it is simply a chore that must be accomplished. This assignment provides a modest solution to the latter situation.

We'd all love to write beautifully, in the style of Shakespeare, Brecht, Brönte, (fill in your favourite author here), but beautiful prose takes hard work. On the other hand, there are severe penalties and social sanctions for trying to pass off other authors' work as our own. The solution (clearly) is a computer program that produces prose in the style of (favourite author here), but without copying an actual work of (same favourite author here).

Here's how the program will work. You give it an example of your favourite author's work, plus a parameter that we'll call *prefLen* (for prefix-length), which is a positive whole number. For the sake of concreteness, let's suppose that *prefLen* is 3. The program that you will help build scans your author's work, looking for groups of 3 adjacent words (prefixes), and for each prefix it keeps a list of which words follow it. For example, in Lewis Carroll's "Alice in Wonderland," the three-word prefix ('Alice', 'was', 'beginning') is followed by one of the words in the list: ['to', 'very'].

Once your program has recorded all the possible completions that Carroll would use for three-word prefixes, we begin our literary adventure by starting out with the first three words of "Alice in Wonderland," which are ('Alice', 'was', 'beginning'), and then randomly choose one of the words from the list [to, very]. Suppose the next word selected is 'very'. We then check our list to see which words follow the three-word prefix ('was', 'beginning', 'very'), and continue building our chain of words, until we have a brand-new piece of literature in the style of Lewis Carroll.

In the previous section you had an exercise to copy "Alice in Wonderland" from my home directory to your directory *A2*, in order to experiment with your program. You may also download lots of public-domain literature from http://www.gutenberg.org.

Your contribution

I have written two short functions (or programs) in the file *lazyLit.py*. Emulate the finger exercise for copying *alice30.txt* to your *A2* directory to copy *lazyLit.py* to your *A2* directory.

Change your current working directory to *A2*. In your terminal window type *idle lazyLit.py* and then press enter. Two windows will pop up: one where you can edit the definitions of functions *buildTable* and *useTable*, the other where you can experiment with your creations. You can switch between these windows by clicking them with your mouse.

The function definitions for *buildTable* and *useTable* are incomplete. Above each function definition are a few lines of text beginning with a single # character that describe the function that follows. These are comments: python ignores the line following the # character, but human readers (you, for example) should not.

The first line of each function definition gives it a name (for example *buildTable*, and a tuple of parameters. When somebody uses the function they store actual values (e.g. numbers or character strings) in the locations named by those parameters.

The remaining lines of each function definition are indented. Some of the lines beginning with a double ##. These are comments that tell you what python code you need to put immediately underneath the ##. Make sure the code you put in lines up horizontally with the left end of the ## (you should be able to reach this with tabs).

If you complete the definitions correctly (you'll probably want to review the finger exercises a lot, and ask your TA and instructor lots of questions), you have a working duo of lazyLit functions. Switch to the

other window and try:

```
>>> import lazyLit
>>> bookTable = {}
>>> lazyLit.buildTable('alice30.txt', 3, bookTable)
>>> lazyLit.useTable(100, 3, bookTable)
```

If your functions are working, you'll get some random prose in the style of Lewis Carroll. Otherwise, you can return to the function definitions, fix things up, return to the window for experimentation, type F6, and repeat the above commands. Whatever happens, record your observations and explanations in *journalA2*. If there is some part of the task that is stumping everyone, your TAs and I will consider some judicious hints.

SOLUTION: Here's a version of the code that worked:

```
import random # some tools to make things random

# buildTable names a function.  You provide values for the parameters
# filename (a string of characters naming a file)
# prefLen (a positive integer)
# bookTable (a lookup table)
def buildTable(filename, prefLen, bookTable):
    ## store a tuple of prefLen '\n' characters at prefix
    prefix = ('\n',) * prefLen
    bookFile = file(filename, 'r')
    for nextLine in bookFile:
        for nextWord in nextLine.split():
            if not bookTable.has_key(prefix):
                ## store an empty list at bookTable[prefix]
                bookTable[prefix] = [] # empty list to start
            ## append nextWord to bookTable[prefix]
            bookTable[prefix].append(nextWord)
            prefix = prefix[1:] + (nextWord,) # next prefix
    bookTable[prefix] = ['\n'] # special signal value for end-of-book
    bookFile.close()
    print 'All done building table.'

# useTable names another function.  The names created in buildTable are
# not visible inside useTable.  You provide values for the parameters:
# numWords (a positive integer)
# prefLen (a positive integer)
# bookTable (a lookup table)
def useTable(numWords, prefLen, bookTable):
    ## store a tuple of prefLen '\n' characters at prefix
    prefix = ('\n', ) * prefLen
    ## store 0 at wordCount
    wordCount = 0
    while 1:
        ## increase value in wordCount by 1
        wordCount += 1
        random.shuffle(bookTable[prefix])
        ## store the 0th string from bookTable[prefix] in nextWord
        nextWord = bookTable[prefix][0]
        ## print nextWord, followed by a space
        print nextWord, " ",
        if nextWord == '\n' or wordCount > numWords: break
        prefix = prefix[1:] + (nextWord, )
```