Programming Bootcamp 2015: Lab 4

Sarah Middleton

Last Updated: June 10, 2015

Name: ANSWER KEY

If you would like to receive points for your work, copy and paste your code into this document, save it, and send it to **sarahmid@mail.med.upenn.edu** with "Bootcamp Lab4" as the subject line **before 9am on 5/16**.

1 Guess the output: list practice (1pt)

You know the drill. Remember that if you do these problems in the interpreter, you must hit 'enter' a second time after an indented block (basically, if you still see the `...` prompt, the interpreter thinks you're still adding to the block. Once you see `>>>`, you can continue to write code.

The following variables will be used. **Remember to reset lists if they get changed!!!**

```
names = ["Wilfred", "Manfred", "Wadsworth", "Jeeves"]
ages = [65, 34, 96, 47]
cat = "Mitsworth"
str1 = "Good morning, Mr.  Mitsworth."
```

Code	Predicted output	Actual output
`print len(ages)`		4
`print len(ages) == len(names)`		True
`for age in ages:` `  print age`		65 34 96 47 (diff lines)
`print ages[1:3]`		[34, 96]

`if "Willard" not in names:`
`  names.append("Willard")`
`print names`
		['Wilfred', 'Manfred', 'Wadswo

`for i in range(len(names)):` `  print names[i],"is",ages[i]`		Wilfred is 65\nManfred is 34\
`ages.sort()` `print ages`		[34, 47, 65, 96]
`ages = ages.sort()` `print ages`		None (because .sort() returns
`print max(ages)`		96
`for i in range(len(cat)):` `  print cat[i]`		M i t s w o r t h (diff lines)
`print cat[:4]`		Mits

`parts = str1.split()`
`print parts`
`print str1`
		['Good', 'morning,', 'Mr.', 'Mits

`parts = str1.split(",")` `print parts`		['Good morning', ' Mr. Mitswo
`print names[-1]`		Jeeves

`oldList = [2, 2, 6, 1, 2, 6]`
`newList = []`
`for item in oldList:`
`  if item not in newList:`
`  newList.append(item)`
`print newList`
		[2, 6, 1]

2 File reading and lists (3pts)

For this problem, use the file `genes.txt`. Remember to spot-check your answers!

(a) **(1 pt)** Read the file and print to the screen only the gene IDs that contain "uc007".

```
inFile = "genes.txt"
ins = open(inFile, 'r')
for line in ins:
    line = line.rstrip('\n')
    if "uc007" in line:
        print line
```

(b) **(1 pt)** Print to the screen only unique gene IDs (remove the duplicates). Do not assume repeat IDs appear consecutively in the file. *Hint: see problem 1 for an example of duplicate checking.*

```
inFile = "genes.txt"
geneList = []
ins = open(inFile, 'r')
for line in ins:
    line = line.rstrip('\n')
    if line not in geneList: #keep track of what we've already seen using a list
```

(c) **(1 pt)** Print to the screen only the gene IDs that are still unique after removing the ".X" suffix (where X is a number).

```
inFile = "genes.txt"
geneList = []
ins = open(inFile, 'r')
for line in ins:
    line = line.rstrip('\n')
    splitLine = line.split(".") #could also do this using string slicing: line[:-2]
```

3 Practice with `.split()` (4pts)

Use `init_sites.txt` to complete the following. This file contains a subset of translation initiation sites in mouse, identified by Ingolia et al. (Cell, 2011). Note that this file has a header, which you will want to skip over.

(a) **(2 pt)** Write a script that reads this file and computes the average CDS length (i.e. average the values in the 7th column).

```
fileName = "init_sites.txt"
totalLen = 0
numLines = 0
ins = open(fileName, 'r')
ins.readline()
for line in ins:
```

(b) **(2 pt)** Write a script that reads this file and prints the "Init Context" from each line (i.e. the 6th column) if and only if the "Codon" column (column 12) is "aug" for that line.

```
fileName = "init_sites.txt"
ins = open(fileName, 'r')
for line in ins:
    line = line.rstrip('\n')
    lineParts = line.split()
    if lineParts[11] == "aug":
```

(c) **(0 pt)** For fun (?), copy and paste your output from (b) into `http://weblogo.berkeley.edu/logo.cgi` to create a motif logo of the sequence around these initiation sites. What positions/nt seem to be most common?

ATG at positions 4,5,6 (as expected, since this is the canonical initiation codon) and a preference for A/G at position 1

4 Cross-referencing (4pts)

Here you will extract and print the data from `init_sites.txt` that corresponds to genes with high expression. There isn't gene expression data in `init_sites.txt`, so we'll have to integrate information from another file.

First, use `gene_expr.txt` to create a list of genes with high expression. We'll say high expression is anything ≥ 50. Then read through `init_sites.txt` and print the GeneName (2nd column) and PeakScore (11th column) lines that match an ID in your high-expression list. Finally, separately compute the average PeakScore for high expression genes and non-high expression genes. Print both averages to the screen.

```
initFile = "init_sites.txt"
exprFile = "gene_expr.txt"

# create a list of "high expression" genes
highExpr = []
ins = open(exprFile)
```

5 All-against-all comparisons (6pts)

A common situation that arises in data analysis is that we have a list of data points and we would like to compare each data point to each other data point. Here, we will write a script that computes the "distance" between each pair of strings in a file and outputs a distance matrix. We will define "distance" between two strings as the number of mismatches between two strings when they are lined up, divided by their length.

First we'll use a toy dataset. Create a list as follows:

```
things = [1, 2, 5, 10, 25, 50]
```

We'll start off by doing a very simple type of pairwise comparison: taking the numerical difference between two numbers. To systematically do this for all possible pairs of numbers in our list, we can make a nested `for` loop:

```
things = [1, 2, 5, 10, 25, 50]

for i in range(len(things)):
    for j in range(len(things)):
        print abs(things[i] - things[j]) #absolute value of the difference
```

Try running this code yourself and observe the output. Everything prints out on its own line, which isn't what we want – we want a matrix-type format. Try this slightly modified code:

```
things = [1, 2, 5, 10, 25, 50]

for i in range(len(things)):
    for j in range(len(things)):
        print abs(things[i] - things[j]), "\t",
    print ""
```

This gives us the matrix format we want. Make sure you understand how this code works. FYI, the `"\t"` is a tab character, and much like `"\n"`, it is invisible once it's printed (it becomes a tab). The comma at the very end of the print statement suppresses the `\n` that `print` usually adds on to the end.

So now we know how to do an all-against-all comparison. But how do we compute the number of mismatches between strings? As long as the strings are the same length, we can do something simple like follows:

```
str1 = "Wilfred"
str2 = "Manfred"
diffs = 0

for k in range(len(str1)):
    if str1[k] != str2[k]: #compare the two strings at the same index
        diffs = diffs + 1

print "dist =", round(float(diffs) / len(str1), 2)
```

So this outputs the distance between the two strings, where the distance is defined as the

fraction of the sequence length that is mismatched.

Using these two pieces of code as starting points, complete the following:

(a) **(2 pt)** Change `things` to be a list of a few short strings of the same length. For example, `things = ["bear", "pear", "boar", "tops", "bops"]`. Write a script that prints a distance matrix for this list. As in the last example, use the fraction of mismatches between a given pair of words as the measure of their "distance" from each other. Round the distances to 2 decimals.

```
things = ["bear", "pear", "boar", "tops", "bops"]
for i in range(len(things)):
    for j in range(len(things)):
        str1 = things[i]
        str2 = things[j]
        diffs = 0
```

(b) **(2 pt)** Now, in place of using a hard coded list, create a list of DNA sequences by reading in the file `sequences2.txt`. Compute the distance matrix between these sequences. Looking at this matrix, do you see a pair of sequences that are much less "distant" from each other than all the rest? Later we'll learn how to display this sort of data in R as a dendrogram, which allows us to quickly identify more closely related sequences.

```
# create list of sequences
things = []
ins = open("sequences2.txt", 'r')
for line in ins:
    line = line.rstrip('\n')
    things.append(line)
```

(c) **(2 pt)** As you can see, the distance matrix is symmetrical around the diagonal. This means dist(i,j) is the same as dist(j,i), so we're doing some redundant calculations. Change the code so that we don't do any unnecessary calculations (including comparing a sequence to itself, which always is 0). (For any calculations you skip, you can print "-" or some other placeholder to keep the printed matrix looking neat.) *Hint: There's a really simple way to do this! Think about the range of the second loop...*

```
### METHOD #1

print ""
print "METHOD 1: lower triangle"
print ""
```