Programming Bootcamp 2015: Lab 5

Sarah Middleton

Last Updated: June 16, 2015

Name:
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 Lab5" as the subject line **before 9am on 5/19**.

1 Guess the output: dictionary practice (1pt)

The following dictionary will be used. You won't need to reset it this time.
`fruits = {"apple":"red", "banana":"yellow", "grape":"purple"}`

Code	Predicted output	Actual output
`print fruits["banana"]`		
`query = "apple"` `print fruits[query]`		
`print fruits[0]`		
`print fruits.keys()`		
`print fruits.values()`		
`for key in fruits:` `  print fruits[key]`		

Code	Predicted output	Actual output
`del fruits["banana"]` `print fruits`		
`print fruits["pear"]`		
`fruits["pear""] = "green"` `print fruits["pear"]`		
`fruits["apple"] += " or green"` `print fruits["apple"]`		

2 Simple file writing (3pts)

(a) **(1 pt)** Write a script that prints "Hello, world" to a file called `hello.txt`

(b) **(1 pt)** Write a script that prints the following text to a file called `meow.txt`. It must be formatted exactly as it here (you will need to use `\n` and `\t`):

```
Dear Mitsworth,

    Meow, meow meow meow.

Sincerely,
A friend
```

(c) **(1 pt)** Write a script that reads in the gene IDs from `genes.txt` and prints the **unique** gene IDs to a file called `genes_unique.txt`. (You can use your

code from the previous lab.)

3 Simple dictionaries (5pts)

Use the following information for the questions below.

```
Wilfred, 555-1234
Manfred, 555-3939
Wadsworth, 555-8765
Jeeves, 555-1001
Mitsworth, 555-5555
```

(a) **(1 pt)** In a script, hard-code a dictionary that would allow us to look up the phone numbers using the person's name (i.e. use the names as the keys and the phone numbers as the values). Then add code to prompt the user for a person's name, and print the phone number of the person they requested. Run your code to test that it works.

(b) **(1 pt)** Instead of prompting the user, use a loop to print each entry in the dictionary in the format:

```
<NAME>'s number is <NUMBER>
```

(c) **(1 pt)** Print just the names in alphabetical order. Use the sorting example from the slides.

(d) **(1 pt)** Print just the names in sorted order based on their phone number. Use the value-sorting example from the slides.

(e) **(1 pt)** Thinkin' question: imagine that we have a much larger set of names and numbers, and we're not sure if some people might have the same name. What problem would this cause when we create our dictionary?

4 The "many counters" problem (3pts)

Write a script that reads a file of sequences and tallies how many sequences there are of each length. Use `sequences3.txt` as input to test your code. After reading through all the sequences, print the sequence length that was the most common.

Hint: you can use a dictionary to keep track of all the tallies, e.g.:

```
seq = "ATGCTGATCGATATA"
length = len(seq)
if length not in tallyDictionary:
       tallyDictionary[length] = 1 #initialize if first occurrence
else:
       tallyDictionary[length] += 1 #otherwise just increment
```

5 Codon table (6pts)

For this question, use `codon_table.txt`, which contains a list of all possible codons and their corresponding amino acids. We will be using this info to translate a nucleotide sequence into amino acids.

(a) **(1 pt)** Thinkin' question: If we want to create a codon dictionary and use it to translate nucleotide sequences, would it be better to use the codons or amino acids as keys?

(b) **(1 pt)** Read in `codon_table.txt` and use it to create a codon dictionary (note that it has a header line). Then prompt the user to enter a single codon (e.g. ATG) and print the amino acid corresponding to that codon to the screen.

(c) **(2 pt)** Now we will adapt the code in (b) to translate a longer sequence. Instead of prompting the user for a single codon, allow them to enter a longer sequence. First, check that the sequence they entered has a length that is a multiple of 3 *(Hint: use the mod operator, %)*, and print an error message if it is not. If it is valid, then go on to translate every three nucleotides to an amino acid. Print the final amino acid sequence to the screen.

(d) **(2 pt)** Now, instead of taking user input, you will apply your translator to a set of sequences stored in a file. Read in the sequences from `sequences3.txt` (assume each line is a separate sequence), translate it to amino acids, and print it to a new file called `proteins.txt`.

6 Parsing fasta files (8pts)

Write a script that reads sequences from a fasta file and stores them in a dictionary according to their header (i.e. use the header line as the key and sequence as the value). You will use `horrible.fasta` to test your code. If you are not familiar with fasta files, they have the following general format:

```
>header line for entry1
ATCGCTAGTCGATCGATGGTTTCGCGTAGCGTTGCTAGCGTAGCTGATG
GCTGGATGGCTAGCTGATGCTAG
>header line for entry2
ATCGATGGGCTGGATCGATGCGGCTCGGCGATCGA
...
```

There are many slight variations; for example the header often contains different information, depending where you got the file from, and the sequence for a given entry may span any number of lines. To write a good fasta parser, you must make as few assumptions about the formatting as possible. This will make your code more "robust".

For fasta files, pretty much the only things you can safely assume are that a new entry will be marked by the > sign, which is immediately followed by a (usually) unique header, and all sequence belonging to that entry will be located immediately below. However, you can't assume how many lines the sequence will take up.

With this in mind, write a robust fasta parser that stores each sequence in a dictionary according to its header line. Remove any newline characters from sequences that span multiple lines, so that the whole sequence will print on one line. Don't include the > sign in the header *(Hint: use string slicing or `.lstrip()`)*.

Use `horrible.fasta` as your input file. Also add the following lines of code to the very end of your script (hopefully you can copy and paste this; if not I'll post it in plain text). This will spot-check whether you did everything correctly. Replace "seqDict" with whatever you named your dictionary:

```
error = False
if ">varlen2_uc001pmn.3_3476" in seqDict:
        print "Remove > chars from headers!"
        error = True
elif "varlen2_uc001pmn.3_3476" not in seqDict:
        print "Something's wrong with your dictionary: missing keys"
```

```
        error = True
if "varlen2_uc021qfk.1>2_1472" not in seqDict:
        print "Only remove the > chars from the beginning of the header!"
        error = True
if len(seqDict["varlen2_uc009wph.3_423"]) > 85:
        if "\n" in seqDict["varlen2_uc009wph.3_423"]:
                print "Remove newline chars from sequences"
                error = True
        else:
                print "Length of sequences longer than expected for some reason"
                error = True
elif len(seqDict["varlen2_uc009wph.3_423"]) < 85:
        print "Length of sequences shorter than expected for some reason"
        error = True

if error == False:
        print "Congrats, you passed all my tests!"
```

Hint: although there are many ways to solve this problem, one way uses a concept related to what you did in the "many counters" problem. Think string concatenation instead of integer incrementing.