Python: Nested loops and Dictionaries

Dissecting the for loop

```
>>> alist = [6, 3, 7, 4, 5, 2, 8]
>>> total = 0
>>> for num in alist:
...     total = total + num
...     print total
...
6
9
16
20
25
27
35
```

- Remember that every item in alist is passed to num in turn
- Why does num keep increasing?

A for loop within a for loop

```
>>> mynums = ['1', '2', '3']
>>> myalph = ['a', 'b', 'c']
>>> for num in mynums:
...     for alph in myalph:
...             print num + alph
...
1a
1b
1c
2a
2b
2c
3a
3b
3c
```

- When a loop is nested in another loop, every element of loop 2 is done for a given element of loop 1 before moving on to the next element of loop 1.
- You can nest any amount of for loops that you want, but the processing time will increase exponentially

Dictionaries

- Dictionaries are another **data type** in Python, and they are specified with **curly brackets**
- They connect a **unique key** to a **value**, allowing fast retrieval of values
- Biology has many natural dictionaries:
  - DNA complements
  - Transcription
  - Translation

```
>>> dnacomp = {'A':'T', 'T':'A', 'C':'G', 'G':'C'}
>>> dnacomp['C']
'G'
>>>
>>> transcript = {'A':'U', 'U':'A', 'C':'G', 'G':'C'}
>>> transcript['U']
'A'
```

- Dictionaries follow the form

  dictname = {key1 : value1, key2 : value2 }
- The keys and values don't have to be strings

Dictionaries

- Dictionaries are another **data type** in Python, and they are specified with **curly brackets**
- They connect a **unique key** to a **value**, allowing fast retrieval of values
- Biology has many natural dictionaries:
  - DNA complements
  - Transcription
  - Translation

```
>>> dnacomp = {'A':'T', 'T':'A',
'C':'G', 'G':'C'}
>>> dnacomp['C']
'G'
>>>
>>> transcript = {'A':'U', 'U':'A',
'C':'G', 'G':'C'}
>>> transcript['U']
'A'
```

- Dictionaries follow the form

  dictname = {key1 : value1, key2 : value2 }
- The keys and values don't have to be strings

An example of a bioinformatics dictionary

- An excellent application of dictionaries is in simple protein translation (I'm doing this one)
- Every 3-letter codon has a corresponding amino acid, and there are special start and stop codons
  - A protein must begin with a start codon and must end when a stop codon appears
  - We will assume that the reading frame given is the only reading frame (generally not true)

The translation dictionary

```
translate = {'UUU' : 'F', 'CUU' : 'L', 'AUU' : 'I', 'GUU' : 'V', 'UUC' : 'F', 'CUC' : 'L', 'AUC'
: 'I', 'GUC' : 'V', 'UUA' : 'L', 'CUA' : 'L', 'AUA' : 'I', 'GUA' : 'V', 'UUG' : 'L', 'CUG' : 'L',
'AUG' : 'M', 'GUG' : 'V', 'UCU' : 'S', 'CCU' : 'P', 'ACU' : 'T', 'GCU' : 'A', 'UCC' : 'S',
'CCC' : 'P', 'ACC' : 'T', 'GCC' : 'A', 'UCA' : 'S', 'CCA' : 'P', 'ACA' : 'T', 'GCA' : 'A',
'UCG' : 'S', 'CCG' : 'P', 'ACG' : 'T', 'GCG' : 'A', 'UAU' : 'Y', 'CAU' : 'H', 'AAU' : 'N',
'GAU' : 'D', 'UAC' : 'Y', 'CAC' : 'H', 'AAC' : 'N', 'GAC' : 'D', 'UAA' : 'Stop', 'CAA' : 'Q',
'AAA' : 'K', 'GAA' : 'E', 'UAG' : 'Stop', 'CAG' : 'Q', 'AAG' : 'K', 'GAG' : 'E', 'UGU' : 'C',
'CGU' : 'R', 'AGU' : 'S', 'GGU' : 'G', 'UGC' : 'C', 'CGC' : 'R', 'AGC' : 'S', 'GGC' : 'G',
'UGA' : 'Stop', 'CGA' : 'R', 'AGA' : 'R', 'GGA' : 'G', 'UGG' : 'W', 'CGG' : 'R', 'AGG' : 'R',
'GGG' : 'G'}
```

- You can see all the elements of the dictionary:
  - A unique set of keys
  - Format is Key : Value, etc.
  - Curly brackets

```
>>> rna =
'AUGGCCAUGGCGCCCAGAACUGAGAUCAAUAGUACCCGUAUUAACGGGUGA'
>>> codons = [ ]
>>> for i in range(0, len(rna), 3):
...         codon = rna[i:i+3]
...         codons.append(codon)
...
>>> codons
['AUG', 'GCC', 'AUG', 'GCG', 'CCC', 'AGA', 'ACU', 'GAG', 'AUC', 'AAU', 'AGU',
'ACC', 'CGU', 'AUU', 'AAC', 'GGG', 'UGA']
>>>
>>> protein = [ ]
>>> for codon in codons:
...         aa = translate[codon]
...             if aa == 'Stop':
...                 break
...             else:
...                 protein.append(aa)
...
>>> protein
['M', 'A', 'M', 'A', 'P', 'R', 'T', 'E', 'I', 'N', 'S', 'T', 'R', 'I', 'N', 'G']
>>> protein = ''.join(protein)
>>> protein
'MAMAPRTEINSTRING'
```

Make a reverse complement

```
>>> dna = 'CACATAGCTAGCATATGTCAAGATTCGCATGCTTA'
```

- Problem: Make a new file called revcom.py and, using a for loop through the dna characters, get the reverse complement of the sequence.
  - Hint 1: don't forget to save to an empty list by appending
  - Hint 2: reversing a string is just like slicing a string, but the start and end are default and the step is -1
- In the end, you should have a list of each reverse/complemented nucleotide
  - Hint 3: remember the **'delim'.join(list)** method? Turn the list into a string

Make several reverse complements

- Open the file dna_data.txt

```
>>> with open('dna_data.txt', 'r') as file:
        data = file.read().splitlines()
```

- Problem: Make a new file called multirevcom.py and, using a for loop through the dna sequences, get the reverse complement of each of the sequences by wrapping what you previously wrote in a for loop
  - Hint 1: save to a larger empty list by appending (call it multicomp = [ ] or something)
  - Hint 2: join the list to get a sequence before appending to the final list
- When you're done, write to a file

```
>>> with open('dna_data.txt', 'w') as file:
        for line in multicomp:
            f.write(line + '\n')
```