# Lab 8: Dictionaries and Matrices

**Pre-Lab Work:**

Before coming to the lab session you may want to play around with dictionaries from within the interactive Python shell. At this point in the course, you should be able to create such exercises for yourself!

What values or errors do the following Python expressions produce? Get the lab files and review `helpful_numpy_examples.py` if necessary, and then predict each result below by drawing "box and arrow" diagrams, and then check your answers by typing (or copy-pasting) each group of statements into an interactive python session. Within each group, some of the results depend on prior steps, so remember to enter all the steps, in the order given, to check an answer.

- ```
  x=1.0/3
  3*x

  y=10*x - 3
  3*y

  x-y
  ```
- ```
  array=[300.0, 300.0, 300.0]
  array
  array[1]

  matrix=[array, array, array, array]
  matrix

  matrix[1]

  matrix[1][1]
  matrix[1][1] = 305
  matrix[1][1]
  matrix[1]
  matrix
  array
  ```
- ```
  from numpy import *
  array=zeros(3, dtype=double)
  array
  array[:]  = 300
  array
  array[1]
  matrix=zeros([4,3], dtype=double)
  matrix[:,:]  = 111
  matrix
  matrix[1][1]
  matrix[1, 1]
  matrix[1, :]
  matrix[:, 1]
  matrix[1] = array
  matrix
  matrix[:]  = array
  matrix
  matrix[1]
  matrix[1][1] = 305
  matrix[1][1]
  matrix[1]
  matrix
  array
  ```

**Lab Work:** For this lab, the starter files that you are given will be essentially empty. You are still responsible for making sure that your program follows the usual good coding practices, including writing a test suite and creating preconditions and postconditions. A few tests (of not yet defined functions) are also included. You should make these tests pass as well as adding your own tests to the test suite.

The goal for this lab will be to explore DNA fragments and determine how to put them together into a single DNA sequence. Multiple methods for determining sequence overlap and analyzing such fragments will be explored.
  ```

1. Dot matrices allow biologists to visually compare two DNA sequences to determine alignment. These square matrices compare two sequences of equal length. The entry in the $i$th row and $j$th column of the matrix should be blank (use 0) if the $i$th character of the first sequence is not equal to the $j$th character of the second sequence and be a dot (use 1) if these characters are equal. Thus, two equal sequences (with no repeating characters) will have the identity matrix as their dot matrix.
   (a) Create a function that takes two sequences and returns its dot matrix.
   (b) Describe (in the comment to your dot matrix function) what patterns you expect to see in the dot matrix if the sequences overlap, how you would determine the start of this overlap, and what you expect to see if the reverse of a substring of one of the sequences matches the other. The last part of this question is interesting since this type of mutation is common.
2. Amino acids are encoded by three-character DNA sequences. Using the provided dictionary (see `genome_lib.py`) to convert from DNA sequence to amino acid abbreviation, find the amino acid composition of the given sequence. You may assume that the first amino acid is the first three characters, the second amino acid is the second three characters (character indices 3 - 5) of the string, and so on. This composition should be reported by returning a dictionary mapping from amino acid abbreviation to the count of the number of times that amino acid was found in the sequence.
3. In the provided `genome_lib.py` library, you will find a function that provides a list of DNA fragments. Your job for this part of the lab is to put these fragments together into a single DNA sequence. You should create a function that takes as input such a list of fragments and returns the single recreated sequence. There may be multiple valid ways to do this. You should return the recreated sequence that most causes the fragments to overlap, i.e. the shortest possible resulting sequence. You may do this however you would like, but may find the following hints helpful:
   - Although the first parts of this lab are helpful when analyzing fragment overlaps by hand, there are 300 fragments that you need to put together. Doing this by hand would be tedious - please don't.
   - You may use a combination of recursive and iterative techniques for this lab, but will likely find the iterative techniques to provide a more straightforward solution.

   Solving this pure version of this problem is hard (and interesting!). For the purposes of this lab, you may make the following additional assumptions:
   (a) The first fragment in a given list is the first fragment in the resulting sequence.
   (b) Each fragment should be followed directly in the sequence by the fragment that it overlaps with the most. You may break ties arbitrarily.
   (c) Although true DNA fragments may include sequences taken from either strand, i.e a sequence could match if 'T's are paired with 'A's and 'C's are paired with 'G's, we will assume that all fragments are given in the 5' to 3' direction (see wikipedia for more information on this convention) and will expect exact character matches. Similarly, you may assume that the fragments are given in the correct orientation, i.e. that the string doesn't need to be reversed.

It's ok if you find that even with these assumptions running your program on the full set of fragments takes a long time (on the order of minutes, but not hours).

**Extra Credit: Do not start this until you have completed all parts of the lab.**

Remove restriction 3c above. You may still assume that the first fragment is the first in the sequence and that it is not complemented or reversed, but any other fragments may have either of these problems. This should allow your code to determine the solution to the fasta input file including possibly complemented fragments (see `genome_lib.py`).

If you'd like an extra challenge (this is *much* harder), you can try to work with the original input file in full (300 fragments). These fragments may be assembled with gaps or a few mismatching base pairs and the first one in the list may not be the first in the resulting sequence. This is the true version of this hard and interesting problem.