50 points total. 70+% correctness (35+ points) is needed to pass. Remember: you must pass all assignments to pass the class.

1. **Textual analysis**
   An *n-gram* is a sequence of n consecutive words in a text. For example, the 2-grams of:

   "I love the Python programming language"

   are "I love", "love the", "the Python", "Python programming", and "programming langue"; the 3−grams are "I love the", "love the Python", "the Python programming", and "Python programming language"; and so on for larger values of n. We say that there are no 7-grams for this sentence because there are only 6 words.

   n-grams are sometimes used to analyze patterns in text (for example, see Google's Ngram Viewer at http://books.google.com/ngrams). In this assignment, you are going to implement a function that computes the k most frequently occurring n-grams in a text file.

   Implement `ngram()` in the file `ngram.py`. The function definition is:

   ```
   def ngram(n, k, document):
   ```

   n specifies that we are computing n-grams, k specifies that we want the k most frequently occurring n-grams (breaking ties arbitrarily), and *document* is the name of a file containing the text. If there are fewer than k n-grams, return all n-grams). The return value should be a dictionary with keys given by n-grams and values given by n-gram frequency in the text (if no n-grams are found, say for n too big, return an empty dictionary).

   You can use `test1()` in this file to test your implementation. This test uses the file `course_description.txt` which is packaged with the starter code. (20 points)

   Assume that the file `document` contains no punctuation. Also assume that all words are separated by a single space. Note that capital letters constitute different words, so "Python programming" and"python programming" would be counted as different 2-grams.

   Finally, only compute n-grams that occur on a single line of the text (not n-grams that contain words from the end of one line and the beginning of the next line). Therefore, you can follow the examples from lecture on reading a file line-by-line.

2. **Money classes**
   The file `stocks.py` contains the skeleton code for a stock portfolio class. This class is similar in spirit to the one presented in class, but it is a different class with different functionality.

   (a) Implement the `add_stock()` and `most_money()` functions. What gets printed from calling the function `test1()`? (8 points)

   (b) Implement the `__contains__()` function. Remember from lecture that this is a special function that works with the `in` operator. What gets printed from calling the function `test2()`? (7 points)

3. **NumPy**
In this problem, we will explore the basics of NumPy. Place all of your code into a file called `myfirstnumpy.py`.

(a) The files `a.txt` and `b.txt` packaged with the starter code each contain data for a vector. Each row of each file contains one floating point number. Implement the function `read_data(fname)` which takes the name of a file formatted like `a.txt` or `b.txt`, reads the file, and stores the data into a list. Return this list. (4 points).

(b) Implement `elementwise_write( fout="c.txt" )` which takes two lists of floats (like the output of the previous question). Convert your lists to the `numpy.array` data type. Use the normal multiplication operator `*` to compute the entry-wise multiplication of the vectors into a new vector, `c`. Write the elements of `c` to the file `c.txt`, one entry per line (the same as the representations in `a.txt` and `b.txt`). (3 points)

(c) The one norm of a vector x of length n is given by:

$$||x||_1 = \sum_{i=1}^{n} |x_i|,$$

i.e., the sum of the absolute values of the entries.

Implement the function `norm_sub(a, b)` which should compute the one norm of the vector $a - b$. With the a and b being `numpy.array` data type. You can use the normal subtraction operator `-` to compute $a - b$. Do not manually compute the norm. Look up the NumPy norm function (what is it?) and use it to compute the one norm. (3 points).

(d) Implement the function `scale()` which takes a 2D-array as input, multiplies the first and last rows by 2 and then multiplies all of the diagonal entries by 5. The function definition should be:
For example, if originally $A = \begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 8 & 9 \end{pmatrix}$, after calling `scale(A)`, A should be $\begin{pmatrix} 20 & 6 \\ 4 & 25 \\ 16 & 18 \end{pmatrix}$.
(5 points)