The program 'analyze.py' (see homepage) reads a file, splits it into words, counts which sequences of length 2 and 3 of letters appear in words how often, and then displays the 5 most frequent sequences. In this assignment, you will refactor this program according to the instructions below, and discuss the effect of the changes on performance and readability of the program. Please do the following **in exactly that order**. **Each step uses the result of the previous step.**

1. Find a project partner (this assignment must be done in a 2-person team).
2. Obtain 'analyze.py' and data (tiny English data, small German data, large Turkish data) from the course website. Run the program, read the source code and understand how it works.
3. Refactor the program as follows:
   - Create a class NGramCounter(n) that counts sequences of length n of words.
   - Replace BigramCounter and TrigramCounter by NGramCounter.
   - Check if the program works correctly and save it as 'analyzeRefactor1NGram.py'.
4. Run 'analyzeRefactor1NGram.py' on all three data sets and observe the memory and time usage (for your report) (memory is printed by the function 'showMemTime').
5. Reduce the memory usage by deleting big data when it is no longer needed (see the 'del' keyword). Observe time and memory usage after this change. Save that file as 'analyzeRefactor2Delete.py'.
6. Refactor the program so that you never load the complete file. You can achieve this using the iterator interface:
   ```
   f = open(filename,'r')
   for line in f:
     # do everything you need to do with 'line' here
     # (for example counting bigrams/trigrams)
     # (the line data will be deleted automatically by python)
   ```
   Observe memory and time usage after this change. Hint: convert each line to unicode instead of the whole file. Save that file as 'analyzeRefactor3Iterator.py'.
7. Refactor the program again:
   - Create a function 'def displayKMostFrequentNGramsInFile(k,n,filename)' which opens the file, creates a n-gram counter class for m, counts the n-grams, and then prints the k most frequent n-grams.
   - In your main function, do only the following: call that function 5 times as follows: show the 30 most frequent 2-grams, show the 20 most frequent 3-grams, show the 15 most frequent 4-grams, show the 15 most frequent 5-grams, show the 15 most frequent 6-grams. (This will read the whole file 5 times, this is ok.)
   - Observe memory and time usage after this change. Save that file as 'analyzeRefactor4More.py'.
8. Put the 4 programs you created so far into a .zip archive.
9. Run the programs and fill in the empty cells of the following table:

program	lines of code (#)	resource usage					
		English data		German data		Turkish data	
		mem (max MB)	time (s)	mem (max MB)	time (s)	mem (max MB)	time (s)
analyze.py							
analyzeRefactor1NGram.py							
analyzeRefactor2Delete.py							
analyzeRefactor3Iterator.py							
analyzeRefactor4More.py							

10. Write a report (1 page) where you describe:
    - The time and memory usage of steps 3, 4, 5, 6, and the table from step 8.
    - Comment on the lines of code, memory, and time usage of the programs.
    - Is there any duplicate code left which could be eliminated?
    - Which step above made the program more understandable/readable and which made it less understandable/readable?
    - What conclusions do you get from this assignment?
11. Send your report (PDF) and your .zip archive to
    berna.altinel@gmail.com and in CC to peter.schuller@marmara.edu.tr .
    Include both of your names and both email addresses in the email!