**Reading**

Read chapter 7 and the week 1 lecture notes.

**Problems**

**Implement the below functions in**[homework1.py](http://reed.cs.depaul.edu/lperkovic/csc242/homeworks/homework1.py)**and submit through COL.**  
**0.**    Lab attendance is required and constitutes 20% of the lab grade.  
 **1.**    Craps is a dice game played at many casinos. A player rolls a pair of normal six-sided dice. If the initial roll sums up to 2, 3, or 12, the player loses. If the roll is 7 or 11, the player wins. Any other initial roll causes the player to "roll for point" which means that the player keeps rolling the dice until either rolling a 7 or re-rolling the value of the initial roll. If the player re-rolls the initial value before rolling a 7, it's a win for the player. Rolling a 7 first is a loss. Write a program that will simulate multiple games of craps and return the fraction of games that the player won. I suggest **your organize your program as follows:**

1. Write a function forPoint that takes as input an initial roll (i.e. an integer between 2 and 12) and repeatedly generates rolls of a pair of dice until either the value of the roll is 7 (in which case 0 is returned) or the value of the roll is the initial roll (in which case 1 is returned).
2. Write a function craps that takes no argument and simulates one game of craps. The program will first roll a pair of dice.  If the value (sum of two dice) of the roll obtained is 7 or 11, then 1 is returned by craps(). If the value obtained is 2, 3 or 12, then 0 is returned by craps(). For all other roll values, forPoint(value) is called and the value returned by it is returned by craps. In other words, craps returns 1 if the player won, 0 otherwise.
3. Write a function testCraps that takeas a positive integer n as input, simulates n games of craps and returns the fraction of games the player won.

Usage:  
>>> craps()  
0  
>>> craps()  
1  
>>> craps()  
1  
>>> testCraps(1000)  
0.51900000000000002  
>>> testCraps(1000)  
0.51100000000000001  
>>> testCraps(1000)  
0.50600000000000001

**2.**    For most fonts, the lowercase letters b and d are mirror images of each other, as are the letters p and q. Furthermore, letters i, o, v, w, and x are naturally mirror images of themselves. Although other symmetries exists for certain fonts, we consider only those specifically mentioned thus far for the remainder of this problem. Because of these symmetries, it is possible to encode certain words based upon how those words would appear in the mirror. For example the word boxwood would appear as boowxod, and the word ibid as bidi. Given a particular sequence of letters, you are to determine its mirror image or to note that it is invalid.

Write function mirror() that takes a string as input and returns its mirror image or string 'INVALID' if the mirror image string is not valid.

Usage:  
>>> mirror('boxwood')  
'boowxod'  
>>> mirror('bidi')  
'ibid'  
>>> mirror('bed')  
'INVALID'  
>>> mirror('ddd')  
'bbb'

**3.**    A checksum is an algorithm that scans a packet of data and returns a single number. The idea is that if the packet is changed, the checksum will also change, so checksums are often used for detecting transmission errors, validating document contents, and in many other situations where it is necessary to detect undesirable changes in data.

1. For this problem, you will implement a checksum function called quicksum. The funtion input (packet) is a string consisting of only uppercase letters and spaces. It always begins and ends with an uppercase letter. Otherwise, spaces and letters can occur in any combination, including consecutive spaces. A quicksum is the sum of the products of each character's position in the packet times the character's value. A space has a value of zero, while letters have a value equal to their position in the alphabet. So, A=1, B=2, etc., through Z=26. Here are example Quicksum calculations for the packets ``ACM" and ``MID CENTRAL":

ACM: 1 \* 1 + 2 \* 3 + 3 \* 13 = 46  
  
MID CENTRAL: 1 \* 13 + 2 \* 9 + 3 \* 4 + 4 \* 0 + 5 \* 3 + 6 \* 5 + 7 \* 14 + 8 \* 20 + 9 \* 18 + 10 \* 1 + 11 \* 12 = 650

Usage:  
>>> quicksum('ACM')  
46  
>>> quicksum('MID CENTRAL')  
650

1. In this problem you will implement function testQuicksum that computes and prints the quicksum of every packet in the file [quicksum.txt](http://reed.cs.depaul.edu/lperkovic/csc242/homeworks/quicksum.txt). The file contains packets followed by a line containing only symbol '-' that signals the end of the input. Each packet is on a line by itself, does not begin or end with a space, and contains from 1 to 255 characters. The testQuicksum function should open the file and enter a loop which calls function quicksum on the next packet, unless the next packet contains '-'. For each packet, your quicksum function should output, on the screen, its quicksum on a separate line in the output. For example, the following should be output when your run testQuicksum on file [quicksum.txt](http://reed.cs.depaul.edu/lperkovic/csc242/homeworks/quicksum.txt).

Usage:  
>>> testQuicksum()  
46  
650  
4690  
49  
75  
14  
15