Warm up discussion problem:

This was an actual job interview question at Qualcomm.

Given 20 "destructible" light bulbs (which breaks at certain height), and a building with 100 floors, how do you determine the height that the light bulb breaks?

All light bulbs are identical in that they all have the following properties:

- If a light bulb is dropped from a given floor and breaks, then any other light bulb will also break if dropped from that or any higher floor.
- If a light bulb is dropped from a given floor and does not break, then neither it nor any other light bulb will break if subsequently dropped from that or any lower floor.
- A light bulb may be dropped any number of times until it breaks, after which it is unusable and cannot be dropped again.

  A "drop" consists of taking a single light bulb in the elevator up to a particular floor, dropping the light bulb from that floor, taking the elevator back down, and checking whether the dropped light bulb broke. Your objective is to determine the highest floor from which a light bulb may be dropped without breaking, using as few drops (i.e., making as few elevator trips) as possible. What should your strategy be to minimize the worst case number of drops required? Given your strategy, what's the minimum number of light bulbs that you need?

Workshop:
Data Collections

1. Most languages do not have the flexible built-in list (array) operations that Python has. Write an algorithm for each of the following Python operations. For example, as a function, `reverse(mylist)` should do the same as the list class method `myList.reverse()`. Obviously, you are not allowed to use the corresponding Python method to implement your function.

   (a) `count(myList, x)` similar to `myList.count(x)`

   (b) `isin(myList, x)` similar to `x in myList`

(c) `index(mylist, x)` similar to `myList.index(x)`

(d) `reverse(myList)` similar to `myList.reverse()`

(e) `sort(myList)` similar to `myList.sort()`

2. Write a function `shuffle(myList)` that scrambles a list into a random order, like shuffling a deck of cards

3. Write a function `removeDuplicates(myList)` that removes duplicate values from a list. While the algorithm for this problem might seem trivial, its implementation is not. In general, it is not advisable to modify a list while you are iterating on it, as in the following pseudo-code:

```
for element in myList:
   if <condition>:
      <remove element(s)from myList> # recipe for disaster!
```

4. Explain what how the dictionary data structure works.

5. Define a function which creates a dictionary where the keys are between the letters "a" and "h", inclusive, and the values are square of keys' ordinal value (*Hint: check an ASCII table*). The function should **`return`** the dictionary back to the caller.

   - Using the returned dictionary from your function, print the entire dictionary.
   - Using the returned dictionary from your function, print only the values of the dictionary.
   - Using the returned dictionary from your function, print only the keys of the dictionary.

6. Write a program, using Python dictionaries, that analyzes text documents and counts how many times each word appears in the document. This kind of analysis is sometimes used as a crude measure of the style similarity between two documents and is also used by automatic indexing and archiving programs (such as Internet search engines). *At the highest level, this is just a multi-accumulator problem. We need a count for each word that appears in the document.*