Work with your neighbor. (This will be graded for participation only.)

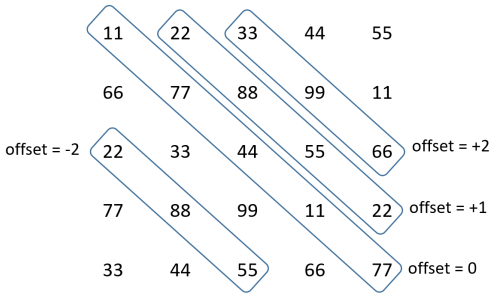
1. Write a function `print_some_words(filename,n)` that takes a filename as a string argument and for each line in the file, finds and prints the individual words of *length great than or equal to* `n` on a separate line.
   A word is defined as a string of characters separated by white space. When considering words, the punctuation characters `".,;?"` should be omitted. For example, if the file `poem.txt` consists of the following lines,

   ```
   Two roads diverged in a yellow wood,
   And sorry I could not travel both
   And be one traveler, long I stood
   And looked down one as far as I could
   To where it bent in the undergrowth;
   ```

   the function `print_some_words("poem.txt", 6)` would print the words below:

   ```
   diverged
   yellow
   travel
   traveler
   looked
   undergrowth
   ```

2. Write a function `sum_diag_UL_LR(grid, offset)` that takes as arguments a grid of numbers and an offset and returns the result of summing the numbers on a specified diagonal of grid. This function considers diagonals running from the upper-left of the grid to the lower-right (hence the 'UL_LR' in the function name). The offset is used to select which diagonal to sum, as shown on the right; the grid shown in this figure is represented in the program as a list of lists:




```
[ [11, 22, 33, 44, 55],
  [66, 77, 88, 99, 11],
  [22, 33, 44, 55, 66],
  [77, 88, 99, 11, 22],
  [33, 44, 55, 66, 77] ]
```

Your program can assume that the argument grid is in fact a grid (i.e., a list of equal-length lists of numbers) but should be able to handle grids of any size.

How to start: Start by writing out the indices of the diagonals for each offset. First do the positive offsets and write the code for the positive offsets. Then do the negative offsets and write the code for the negative offsets. Then write out the two separate loops. (Consider using while loops rather than for loops.)

```
def sum_diag_UL_LR(grid, offset):
    sum = 0
    if offset >= 0:
       # write the code for the positive case
    else:
       # write the code for the negative case
    . . .
    return sum
```

3. Write a function `print_keys(d)` that prints the keys in the dictionary `d`. For example, if the dictionary passed in is

    {"I": 1, "V": 5, "X": 10, "L": 50}

    then the function prints the following:

    I

    V

    X

    L