Review for final exam

Some of these exercises are fairly challenging. On the final exam, you can expect that we will ask a few more challenging programming questions in which you might have to tackle more complex problems, writing helper functions, etc.

For the first two questions, suppose we have a list of numbers where each number represents the points earned on a basketball player's shooting attempt. Thus, the value of the number is either:

- 0 - a miss
- 1 - a made free throw
- 2 - a made 2 pointer
- 3 - a bucket from downtown!

We are interested in calculating the number of shooting streaks. We'll write a few versions using different definitions of a shooting streak.

1. Write a function `streaks` that takes such a list and returns the number of shooting streaks. A streak is defined as one or more consecutive baskets. Examples:

```
>>> streaks([0,3,2,1,3,0])
1
>>> streaks([0,3,2,1,0,3])
2
```

Solution:

```
def streaks(shots):
    '''(list of int) -> int
    shots is a list of numbers that represent
    the points earned by a basketball player's
    shooting attempts.

    Returns the number of shooting streaks --
    the number of times the player made one or more
    shots in a row.

    >>> streaks([0,3,2,1,3,0])
    1
    >>> streaks([0,3,2,1,0,3])
    2
    '''
    streaks = 0
    in_streak = False
    for shot in shots:
```

```
        if shot > 0 and not in_streak:
            streaks += 1
            in_streak = True
        elif shot == 0:
            in_streak = False
    return streaks
```

2. Same as previous question but this time a streak is defined as a sequence of three or more consecutive baskets. Examples:

```
>>> streaks([0,3,2,2,2,1,3,0])
1
>>> streaks([0,3,2,2,0,3,2])
1
>>> streaks([3,2,1,0,3,3,3,0,0,0,2,1,1])
3
```

Solution:

```
def streaks(shots):
    '''(list of int) -> int
    shots is a list of numbers that represent
    the points earned by a basketball player's
    shooting attempts.

    Each number in the list represents a single
    shot attempt.  The value of the number is
    either:
      0 - a miss
      1 - a made free throw
      2 - a made 2 pointer
      3 - a bucket from downtown!

    streaks returns the number of shooting streaks --
    the number of times the player made three or more
    shots in a row.

    >>> streaks([0,0,3,2,2,0,0])
    1
    >>> streaks([1,2,3,3,0,2,2,3,0,1,1,2])
    3
    '''
    streaks = 0
```

```
    curr_streak = 0
    for shot in shots:
        if shot > 0:
            curr_streak += 1
        else:
            curr_streak = 0  # streak ends
        if curr_streak == 3:
            streaks += 1
    return streaks
```

The remaining problems are not related to basketball.

3. Given a birthday month dictionary such as:

```
{'February' : {13 : ['Catherine']},
 'May' : {3 : ['Katie'], 8 : ['Peter', 'Ed']},
 'December' : {12 : ['Sharon'], 22 : ['Owen']}
}
```

Write a function that takes a birthday month dictionary and returns a list of month names where a month is included if and only if every birthday in that month is unique – i.e., no two people share a birthday in that month. On above example, function would return `['February', 'December']`.

Solution:

```
def all_unique(bdm):
    """
    Given a birthday month dictionary bdm, this
    returns a list of month names for those months
    where every birthday in that month is unique -- i.e.,
    no two people share a birthday in that month.
    >>> bdm = {"February" : {13 : ["Catherine"]}, "May" : {3 : ["Katie"], 8 : ["Peter"
    >>> all_unique(bdm)
    ['December', 'February']
    """
    months = []
    for month in bdm:
        unique = True
        for day in bdm[month]:
            if len(bdm[month][day]) > 1:
                unique = False
        if unique:
            months.append(month)
    return months
```

4. Write a function `find_match` that takes two parameters, a string `s` and another string `pattern`, and returns the index of the first occurrence of `pattern` in `s`, or -1 if it does not occur.

   The pattern is a string, possibly with wildcards. The wildcard character * can match any single character.

   Examples:

```
>>> find_match('xyzabcd', 'b*d')
4
>>> find_match('abcd', 'a**d')
0
>>> find_match('abcd', 'b**d')
-1
```

   For this problem, you *must* write a helper function. Hint: consider taking a substring of `s` that is *exactly* the same length as `pattern` and checking to see if that substring is a match for the pattern. If we repeat this for each substring of `s` we can find the match (if one exists).

Solution:

```
def is_match(s, p):
    '''(str, str) -> bool
    Returns True if s matches p where p
    may have wild cards.
    >>> is_match('bcd', 'b*d')
    True
    >>> is_match('bcd', 'bd*')
    False
    '''
    matches = 0

    # if different lengths can't match
    if len(s) != len(p):
        return False

    # this loop assumes they are the same length
    for i in range(len(s)):
        if s[i] == p[i] or p[i] == '*':
            matches += 1
    return matches == len(s)

def find_match(s, pattern):
    '''
```

```
    Return index of first occurrence of pattern
    in string s, or -1 if pattern does not occur.

    Pattern is a string, possibly with wildcards.
    The wildcard character * can match any single
    character.

    >>> find_match('xyzabcd', 'b*d')
    4
    >>> find_match('abcd', 'a**d')
    0
    >>> find_match('abcd', 'b**d')
    -1
    '''
    # be careful about indexing...  we don't need to
    # go all the way to end of s because we need at
    # least len(pattern) characters to find a match
    for i in range(len(s) - len(pattern) + 1):
        if is_match(s[i:i+len(pattern)], pattern):
            return i
    return -1
```

5. For this question, imagine that we have a list of votes for prettiest spring campus. It might look something like this:

```
votes = ['colgate', 'dartmouth', 'colgate', 'UVA', 'cornell']
```

   (a) Write a function that takes in a list of votes and returns the name of the school that received the most votes. You cannot use any list methods (e.g., count).

   (b) Write a function that takes in a list of votes and a number k and returns a list of the names of the top k most popular schools. (Don't worry about ties in the k^{th} position.)

   (c) Same as the previous question, but revise the function so that any school that is tied for the k^{th} most votes is included in the final list. The list might end up being more than k names long. For example, on the list above, the function would return the whole list when $k = 2$ because there are three schools tied for the second spot.

Solution:

```
# helper function
def counter(L):
    '''(list of str) -> dict of str:int
    Returns a dictionary mapping each string in L
```

```
    to the number of times it occurs in L.
    >>> counter(['a', 'b', 'b', 'c', 'b', 'a'])
    {'a': 2, 'c': 1, 'b': 3}
    '''
    counts = {}
    for item in L:
        if item in counts:
            counts[item] += 1
        else:
            counts[item] = 1
    return counts


def mode(L):
    '''(list of str) -> str
    Returns most frequently occurring string in L.
    >>> mode(['a', 'b', 'b', 'c', 'b', 'a'])
    'b'
    '''
    counts = counter(L)
    max_count = max(counts.values())
    for item in counts:
        if counts[item] == max_count:
            return item

def top_k(L, k):
    '''(list of str, int) -> list of str
    Returns the top k most frequently occurring strings in L.
    >>> top_k(['a', 'b', 'b', 'c', 'b', 'a'], 1)
    ['b']
    >>> top_k(['a', 'b', 'b', 'c', 'b', 'a'], 2)
    ['b', 'a']
    '''
    counts = counter(L)
    pairs = []
    for item, count in counts.items():
        pairs.append([count, item])  # decorate
    pairs.sort() # sort
    pairs.reverse()
    top_k = pairs[:k]
    for i in range(len(top_k)):
        top_k[i] = top_k[i][1] # undecorate
    return top_k
```

```
def top_k_with_ties(L, k):
    '''(list of str, int) -> list of str
    Returns the top k most frequently occurring strings in L,
    including ties at the kth position.
    >>> top_k_with_ties(['a', 'b', 'b', 'c', 'b', 'a'], 1)
    ['b']
    >>> top_k_with_ties(['a', 'b', 'b', 'c', 'b', 'a'], 2)
    ['b', 'a']
    >>> top_k_with_ties(['a', 'c', 'b', 'c', 'b', 'a', 'd'], 2)
    ['c', 'b', 'a']
    '''
    counts = counter(L)
    pairs = []
    for item, count in counts.items():
        pairs.append([count, item])  # decorate
    pairs.sort() # sort
    pairs.reverse()
    top_k = []
    kth_count = -1
    for pair in pairs:
        if len(top_k) < k or pair[0] == kth_count:
            top_k.append(pair[1]) # undecorate
            kth_count = pair[0]
    return top_k
```