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:

```python
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:

```python
>>> 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:

```python
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
    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
```
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:

```python
{ '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:

```python
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", "Ed"]},
    "December" : {12 : ["Sharon"], 22 : ["Owen"]
    }
    >>> all_unique(bdm)
    ['December', 'February']
    
    months = []
    for month in bdm:
        unique = True
        for day in bdm[month]:
            if bdm[month][day].count(day) != 1:
                unique = False
                break
        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:
```python
>>> 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:
```python
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', 'b*d')
    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 $k^{\text{th}}$ 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:

```python
# 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()

k = pairs[:k]
for i in range(len(k)):
    k[i] = k[i][1]  # undecorate
return 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()

k = []
kth_count = -1
for pair in pairs:
    if len(k) < k or pair[0] == kth_count:
        k.append(pair[1])  # undecorate
        kth_count = pair[0]
return k
6. What is printed by each of these programs? If the program contains an error, explain it.

(a) \( x = [1, 2, 3] \)

```python
def f1(z):
    return z
print ('z returned!')

print (f1(x))
print (x)
```

Solution:

\[ [6, 2, 3] \]
\[ [6, 2, 3] \]

(b) \( \text{def } f2(x):
    y = x
    x = \{0: 'a', 1: 'b', 2: 'c'\}
    y[-1] = 'e'
    x = ['a', 'b', 'c']
print (f2(x))
print (x)
```

Solution:

None
\[ ['a', 'b', 'e'] \]

(c) \( \text{def } f3(x):
    y = {}  
    for k in x:
        y[x[k]] = k
    return y

x = \{'sun': 'good', 'rain': 'bad'\}
f3(x)
print (y.keys())
```

Solution:
Traceback (most recent call last):
  File "exercise_f3.py", line 9, in <module>
    print y.keys()
NameError: name 'y' is not defined

(d) def f4(x,y):
    t = x
    x = y
    y = t
    x['sun'] = 2
    y['rain'] = 0

    a = {'sun':0}
    b = {'rain':2}
    f4(a,b)
    print (a)
    print (b)

Solution:
{'sun': 0, 'rain': 0}
{'sun': 2, 'rain': 2}
7. Write a recursive function has_6 that takes a list of numbers and returns True if the list contains a 6 and False otherwise. You cannot use the in operator, loops, etc.

Solution:
```python
def has_6(L):
    '''(list of int) -> bool
    Returns True if 6 is in L, False otherwise.
    >>> has_6([1,2,6,3])
    True
    >>> has_6([1,2,3])
    False
    ...'''
    if len(L) == 0:
        return False
    elif L[0] == 6:
        return True
    else:
        return has_6(L[1:])
```

8. Write a recursive function index_6 that takes a list of numbers and returns the index of 6 in the list or -1 if it’s not in the list. You cannot use the index method, loops, etc.

Hint: consider the following snippet of code. What does position equal? It’s not 2!

L = [8, 7, 6, 13]
position = index_6(L[1:])

Solution:
```python
def index_6(L):
    '''(list of int) -> int
    Returns the index of 6 in L, -1 if 6 not in L.
    >>> index_6([1,2,6,3])
    2
    >>> index_6([1,2,3])
    -1
    ...'''
    if len(L) == 0:
        return -1
    elif L[0] == 6:
        return 0
    else:
        return index_6(L[1:])
```
else:
    idx = index.6(L[1:])
    if idx == −1:
        return −1
    else:
        return idx+1

9. We have a number of bunnies and each bunny has two big floppy ears. We want to compute the total number of ears across all the bunnies recursively. Write a recursive function bunny_ears that takes in a number and returns the number of ears. You cannot use loops or multiplication.

Examples:
>>> bunny_ears(0)
0
>>> bunny_ears(1)
2
>>> bunny_ears(2)
4

Solution:
def bunny_ears(how_many):
    '''(int) -> int
    Returns the number of ears for how_many bunnies.''
    >>> bunny_ears(0)
    0
    >>> bunny_ears(1)
    2
    >>> bunny_ears(2)
    4
    ...
    if how_many == 0:
        return 0
    else:
        return 2 + bunny_ears(how_many−1)

10. Write a recursive function count_hi that takes a string and returns the number of times lower-
case 'hi' appears in the string. You **cannot** use the `find` method, loops, etc.

**Solution:**

```python
def count_hi(s):
    '''(str) -> int
    Returns the number of times 'hi' occurs in s.'
    >>> count_hi('xxhixx')
    1
    >>> count_hi('hixhixx')
    2
    >>> count_hi('hxixhxix')
    0
    >>> count_hi('xhi')
    1
    ...
    if len(s) <= 1:
        return 0
    elif s[:2] == 'hi':
        return 1 + count_hi(s[2:]) # safe to skip 2
    else:
        return count_hi(s[1:]) # only move 1 (see last example in docstring)
```

The final exam will have one question that is longer (and worth more points) than the typical programming questions you’ve seen on the midterms. This reflects the fact that after working on homework 10, 11 and especially 12, you have developed your skills in writing more complex programs that involve multiple functions.

Here are some “bigger” practice problems to consider. These are challenging but you are strongly encouraged to practice them.

11. You are given a list of dates of birthdays. Each date is one of two formats: year month day as in '2014/12/31' or month, day, year as in '12/31/2014'. Write a function `build_birthday_dictionary` that takes in such a list and returns a birthday dictionary of a particular form. The keys are month names (you can use three letter abbreviations: “Jan,” “Feb,” etc.). The values are dic-
tionaries. These inner dictionaries have days for keys and each value is a list of years.

Example:

```python
>>> dates = ['1975/12/17', '1976/12/17', '1944/12/11', '1974/03/27']
>>> build_birthday_dictionary(dates)
```

Hint: assume you have a list called `MONTHS` that looks like this:

```python
```

Solution:

```python
def extract_parts(date):
    parts = date.split('/
    if len(parts[0]) == 4:
        year = parts[0]
        month = parts[1]
        day = parts[2]
    else:
        month = parts[0]
        day = parts[1]
        year = parts[2]
    return [month, day, year]

def update_bday(m, d, y, bdays):
    if m not in bdays:
        bdays[m] = {}
    if d not in bdays[m]:
        bdays[m][d] = []
    if y not in bdays[m][d]:
        bdays[m][d] += [y]

def build_birthday_dictionary(dates):
    '''(list of str) -> dict
    Expects a list containing
    ```
```python
bdays = {}
for date in dates:
m, d, y = extract_parts(date)

m = MONTHS[int(m)−1]

update_bday(m, d, y, bdays)

return bdays

dates = ['1975/12/17', '1976/12/17',
         '1944/12/11', '1974/03/27']
print(build_birthday_dictionary(dates))
```

12. This question has two parts:

i. Write a function dna.squish that takes a sequence of DNA (a string of A, C, G, T) and “compresses” it as follows. Subsequences of the same character are replaced to a single copy of the character followed by the number of times that character occurs. For example, 'ACCCGGCAAAAA' would be compressed to 'A1 C3 G2 C1 A5'.

The resulting string is compressed if the original DNA string contains many repeated characters.

ii. Write a function dna.unsquish that takes as input the string produced by the previous problem and reconstructs the original dna string. On input 'A1 C3 G2 C1 A5', it returns 'ACCCGGCAAAAA'.

iii. Challenge edition: do not put spaces between the groups when squishing. When unsquishing, make sure your program works even when a group is 10 or larger, e.g., 'C3A10C1' would produce 'CCCCAAAAAAAAAC'

Solution:

```python
def dna.squish(dna):
i = 0
squish = []
while i < len(dna):
curr_ch = dna[i]
count = 0
```
```python
while i < len(dna) and dna[i] == curr_ch:
    count += 1
    i += 1

squish += [curr_ch + str(count)]

return ' '.join(squish)

def dna_squish_v2(dna):
    i = 0
    squish = []
    dna += 'X' # neat trick: add sentinel to end
    # makes code a bit simpler
    while dna[i] != 'X':
        curr_ch = dna[i]
        count = 0

        while dna[i] == curr_ch:
            count += 1
            i += 1

        squish += [curr_ch + str(count)]

    return ' '.join(squish)

def dna_unsquish(squished):
    dna = ''
    groups = squished.split()
    for group in groups:
        ch = group[0]
        count = int(group[1:])
        dna += ch * count

    return dna

def dna_unsquish_challenge(squished):
    # expects squished string without spaces: like 'A13G2T3G2'
    dna = ''
    i = 0
    while i < len(squished):
        ch = squished[i]
        print 'ch', ch
```
i += 1
count = ' '
while i < len(squished) and squished[i].isdigit():
    count += squished[i]
    i += 1
print 'count', count
count = int(count)
dna += ch * count
return dna

print (dna_unsquish_challenge('C3A10C1'))