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 \texttt{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

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([3,2,1,0,3,3,3,0,0,2,1,1])
   3

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'].

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).

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

```python
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.
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)
```

(b) \( \text{def } f2(x):
\)

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

    x = ['a', 'b', 'c']
print (f2(x))
print (x)
```

(c) \( \text{def } f3(x):
\)

```python
def f3(x):
    y = {}
    for k in x:
        y[x[k]] = k
    return y

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

(d) \( \text{def } f4(x, y):
\)

```python
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)
```
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.

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!

   ```python
   L = [8, 7, 6, 13]
   position = index_6(L[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:
   ```python
   >>> bunny_ears(0)
   0
   >>> bunny_ears(1)
   2
   >>> bunny_ears(2)
   4
   ```

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.

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 dictionaries. 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
```

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'