CS1210 Computer Science I: Foundations
Exam 2
Thursday, October 29, 2015

This test consists of 4 sections, all with multiple items, worth a total of 62 points. Because they are of varying difficulty, it is probably good strategy not to get stuck too long on any one question. You have 90 minutes: **please write legibly** and **don't write in tiny letters**. Remember, real people with poor eyesight (*i.e.*, me) have to read it.

1. Regular Expressions [6 points]

Indicate whether or not the following patterns match the given string (carefully) box the part that matches (or indiciate if there was no match). You can assume we are using the re.search() function to obtain the match, although please note the pattern given is not correctly "escaped" for use directly in re.search().

2 points each

Pattern	*String*	
[^abc]{3}\s+	'abcdefghi 123'	**'ghi '**
\d{3}-\d{2}-\d{5}	'1234-56-8910bcd123-45-67890123-k22'	**'123-45-67890'**
test\s+test\s*test\stest$	'testtest test test testtest test'	**None**

2. Simple Functions [30 points]

(1) Recall the word net dictionary we constructed in class, where each key was a 5 letter word and each value was a list of other 5 letter words that differed by exactly one letter from the key. Complete the following function, which takes a word, *w*, and the word net dictionary, *D*, as inputs and returns a list of words reachable from *w* in *exactly* two hops ("friends of friends," so to speak) *but not less*.

```
def FoF(w, D):
    S = set()
    for w2 in D[w]:
        for w3 in D[w2]:
            if w3 not in D[w] and w3 != w:
                S.add(w3)
    return(list(S))
```

5 points

(2) Write a function that takes a string, S, as its input and returns a floating point number indicating the average length of the words in S. So:

```
>>> from string import punctuation
>>> punctuation
'!"#$%&´()*+,-./:;<=>?@[\]^_`{|}~'
>>> avgLen('This is a test of the emergency broadcast system')
4.444444444444445
```

While you can presume there's at least one space between every word, you should strip any punctuation marks from both ends of the individual words before computing the average length (you can assume punctuation is already imported from the string module).

```
from string import punctuation
def avgLen(S):
    W = [len(w.strip(punctuation)) for w in S.split()]
    return(sum(W)/len(W))
```

5 points

(3) Write a function that takes a single argument, L, such that L is a list containing integers or lists, with the embedded lists also containing integers or lists, and so on. Write a function that returns the sum of all the integers in the input argument, regardless of the level of nesting. So, for example:

```
>>> sumall([1, 2, [3, 4], [5, [-7, -3]], -5])
0
>>> sumall([1, 2, [3, [4, -2, -2]], [5, [-7, -3]], -10])
-9
```

```
def sumall(L):
    if type(L) == int:
        return(L)
    return(sum([sumall(x) for x in L]))
```

5 points

(4) Consider interpreting a list of lists as a representation for a matrix. For example, the matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

would be represented as:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

where each element of the list corresponds to a row of the matrix. You may assume that the matrix is well formed, that is, that every row is of the same length, but you cannot assume that the number of rows and columns is the same. Write two functions that manipulate (that is, mutate) the matrix in the following ways. The first function exchanges (*i.e.*, swaps) two specified rows, while the second exchanges two specified columns:

```
>>> m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
>>> swapRows(m, 0, 2)
>>> m
[[7, 8, 9], [4, 5, 6], [1, 2, 3]]
>>> m
>>> swapCols(m, 0, 2)
[[9, 8, 7], [6, 5, 4], [3, 2, 1]]
```

```
def swapRows(M, i, j):
    if 0 <= i < len(M) and 0 <= j < len(M):
        M[i],M[j] = M[j],M[i]

def swapCols(M, i, j):
    if 0 <= i < len(M) and 0 <= j < len(M):
        for r in M:
            r[i],r[j] = r[j],r[i]
```

4 points each

Write a function that flips the matrix about the diagonal axis by mutating the matrix (that is, not creating a new copy), but only if the matrix is square, that is, it has the same number of rows as columns.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
def flip(M):
    if len(M) == len(M[0]):
        for i in range(len(M)):
            for j in range(i+1, len(M[i])):
                M[i][j],M[j][i] = M[j][i],M[i][j]
```

7 points

3. Algorithms and Efficiency [6 points]

Given a collection of N things, you want to select a subset of size $M \leq N$ from the original collection at random, while ensuring that every element in the original collection has an equal probability of being selected. To make the problem a little simpler, let's assume that the N things are simply the values in $list(range(N))$, and you would like the selected numbers to be returned in order.

Your general strategy is as follows: consider each of the N numbers in order. When considering the ith element, select it for inclusion with probability $\frac{j}{N-i}$, where $0 < j \leq M$ is the number of elements left to be selected. When you select an element, decrement j, otherwise leave it alone.

This should remind you of our shuffle from a previous homework. If $M = 1$, then you simply consider selecting each successive element in $range(N)$ with probability $\frac{1}{N-i}$. When you finally select an element, the probability another element gets selected drops to $\frac{0}{N-i}$. On the other hand, if $M = N$, then every element is considered for selection with probability $\frac{M}{N-i}$ which remains 1 as i gets larger and j gets smaller in lockstep.

(1) Complete the following outline for the function in question.

3 points

```
def MofN(N, M):
   L = []                                          # 1
   j = M                                           # 2
   for i in range(N):                              # 3
      if random() <= (j/(N-i)):                    # 4
         L.append(i)                               # 5
         j=j-1                                     # 6
   return(L)                                       # 7
```

(2) Using "order of" notation, what is the runtime of this algorithm in terms of *N*? Explain how you computed the expected runtime; feel free to use the numbers out to the right of the function to refer to individual lines.

3 points

The runtime is O(N).
(1,2) The first two assignment statements are both O(1).
(3) The for loop executes N times.
(4) The body of the loop contains an if, where the condition is O(1).
(5, 6)The body of the if consists of two O(1) assignments.
(3-6) The body of the for loop is O(1), so the for loop is O(N).
(7) The return is O(1).
Thus the whole function is O(1)+O(N)+O(1) or O(N).

4. Objects and Classes [20 points]

Recall the Python set datatype, which one can think of as "dictionaries without values." The main idea here is that, unlike lists, no item can be repeated in a set. But what if Python did not include the set datatype?

Assume its up to you to create the *Set* class. Assume further that your new class stores the items in the set internally in a list, L, associated with the object.

A transcript of your new class in operation might therefore look like:

```
>>> a=Set()
>>> a
{}
>>> a.add(1)
>>> a.add(2)
>>> a
{1, 2}
>>> a.add(2)                          # Add second copy of 2?
>>> a.L
[1, 2]
>>> b=Set(1,2,3,4,5,5,5,'a','b')     # Multiple copies of 5?
>>> b.L
```

```
[1, 2, 3, 4, 5, 'a', 'b']
>>> b.remove(4)
>>> b.remove('c')                    # Remove nonexisting element?
>>> b.L
[1, 2, 3, 5, 'a', 'b']
```

Note the following three observations:

- Multiple copies (*e.g.*, several 5's) are not allowed;
- Removal of a nonexisting element (*e.g.*, the 'c') does not throw an error; and
- The constructor allows any number of initial elements.

Complete the basic class given here, including the constructor and the methods add(), remove(), and clear().

```
class Set():
  # Constructor initializes the empty list and the type.
  def __init__(self, *elements):
    '''Initialize the set.'''
    self.L=[]
    for e in elements:
      self.add(e)
```

3 points

```
  def add(self, element):
    '''Add element to self.L.'''
    if element not in self.L:
      self.L.append(element)

  def remove(self, element):
    '''Remove element from self. Could instead use self.L.del(element).'''
    if element in self.L:
      self.L.remove(element)

  def clear(self):
    '''Reset the set to the empty set.'''
    self.L=[]
```

3 points each

Let's add a few more advanced features to our implementation. One of these more advanced features is a more appealing printed representation of the set. Instead of behaving as follows:

```
>>> a=Set()
>>> a
<__main__.Set object at 0x7f45583bce80>
>>> print(a)
<__main__.Set object at 0x7f45583bce80>
```

We would like our implementation to behave like this:

```
>>> a=Set(1, 2, 5)
>>> a
{1, 2, 5}
>>> print(a)
{1, 2, 5}
```

Name the method that is required to achieve this behavior and complete its definition:[1]

```
def __repr__(self):
    '''Return a string representation of the set.'''
    S=str(self.L)
    return('{{{0}}}'.format(S[1:len(S)-1]))
```

4 points

Finally, our set also exhibits the following nifty behavior:

```
>>> a=Set(1, 2, 3, 4, 5)
>>> len(a)
5
```

Name the method that is required to achieve this behavior and complete its definition.

```
def __len__(self):
    '''Return the number of elements in the set.'''
    return(len(self.L))
```

4 points

[1] Hint: if you elect to use the string.format() method in your code (which is technically not required to solve this problem), you will need to know how to print explicit curly brackets '{' and '}' (since these characters have special meaning to the format() method). The hint is that an explicit '{' can be printed as '{{' and '}' can be printed as '}}', whereas the single '}' and '{' retain their special meanings to format().