Quiz 9

To get credit for this quiz, use the Quiz tool at eee.uci.edu to enter your answers, within the Sunday-to-Tuesday quiz period.

Problem 1 (8 points) **Topic: Two-dimensional tables**

Suppose an exam has two problems, each worth 20 points. We want to see how students' scores on Problem 1 relate to their scores on Problem 2, so we decide to make a scatter plot.

Our students come in a list of Score records, defined as follows:

```
from collections import namedtuple
Score = namedtuple('Score', 'p1 p2')
TOPSCORE = 20
```

Both `p1` and `p2` are ints between 0 and `TOPSCORE` (inclusive). With this list:

```
scorelist = [Score(p1=0, p2=0),
             Score(p1=1, p2=1),
             Score(p1=1, p2=5),
             Score(p1=4, p2=2),
             Score(p1=5, p2=0)]
```

the scatter plot of scores would look like this (except that we've omitted the 75% of the table that would show scores greater than 5):

```
5| *
4|
3|
2|    *
1| *
0|*    *
  ------
  012345
```

To keep things simpler in our problem, we're going to omit the axes and the labels and print just the 21-by-21 body of the plot.

```
# Initialize the table to a 20-by-20 table of blanks
table = [ ]
for row in range(TOPSCORE+1):
    table_row = [ ]
    for col in range(TOPSCORE+1):
        table_row.append(' ')
    table.append(table_row)

# Populate the table with an asterisk for each student's two scores.  (When two
# students have the same pair of scores, just one asterisk appears.)
for s in scorelist:
    table[s.p2][s.p1] = '*'

# Print the 20-by-20 table
for row in range(TOPSCORE,-1,-1):
    for col in range(TOPSCORE+1):
        print(table[row][col], sep='', end='')
    print() # Print the default end= character, a newline
```

Please answer each of the following questions in just a few English words:

A. Why do we have to say `range(TOPSCORE+1)`?

These answers are a little more complete than would be necessary for credit on an exam. It's also likelier that on an exam, you'd be given some choices rather than be asked to write a prose answer.

range(TOPSCORE) goes from 0 to 19 in this case, but the scores go from 0 to 20. We have 20 possible scores plus zero, for a total of 21.

B. Why do we have to say `table[s.p2][s.p1]` and not `table[s.p1][s.p2]`?

We want to plot the first score (p1) on the x-axis (left to right). But the first index in the table is the row number (y-axis), top to bottom; we want to plot p2 on that axis so p2 has to go first.

C. When we print the table, why do we print the rows in a backwards range (`TOPSCORE` down to 0)?

We think of the scatter plot as having the origin (0,0) in the lower left corner. But we have to print it from the top down, highest row number first.

D. Why do we have `sep=''` and `end=''` when we print a row of the table?

To avoid extra horizontal spaces and vertical spaces; to print the asterisks right next to each other

Problem 2 (17 points) **Topic: Frequency table using dictionaries; also sets**

Suppose we have a list of names, some of which may occur more than once on the list. For example:

```
NL = ['Joe', 'Sam', 'Joe', 'Jill', 'Joe', 'Joe', 'Jill', 'Sam', 'Jane', 'Jane', 'Jane', 'Joe', 'John']
```

And suppose that we want to know which name occurs most frequently.

(a) (5 points) First we can create a dictionary that gives us a collection of each distinct name on the list, along with the number of times it occurs. Fill in the blanks of the following definition, with one identifier, constant, or operator in each blank, to be consistent with the provided code:

```
def tally_names(L: [str]) -> dict:
    ''' Return a dictionary with each unique string in L as the key and
        the number of times that string occurs in L as the value.
    '''
    result = { }
    for s in __________:  L
        if __________ in __________:    s  result
            __________ [__________] __________  __________   result[s] += 1
        else:
            __________ [__________] __________  __________   result[s] = 1
    return __________   result

assert tally_names(NL) == {'Sam': 2, 'Jill': 2, 'Joe': 5, 'Jane': 3, 'John': 1}
```

(b) (2 points) We want to find the most frequently occurring name, but we can't sort the dictionary because dictionaries, as we know, are inherently unsorted (they can't be, because of how they're built [using a "hash table," the details of which are a topic for ICS 33 or ICS 46]). But we can use a function like the one below to convert the dictionary to a list of key-value pairs (where each pair is a two-item list, [key, value]).

```
def dict_to_list(d: dict) -> 'list of [key, value] pairs':
    ''' Convert dictionary (with key/value entries) to a list of [key, value] pairs
    '''
    result = [ ]
    for key in d:
        result.append([key, d[key]])
    return result
```

What does the following statement print, using the definition of NL above? (Hint: Look at the assertion for the previous part.)

```
print(dict_to_list(tally_names(NL)))
```

[['Jane', 3], ['Jill', 2], ['John', 1], ['Joe', 5], ['Sam', 2]]
(The order of the pairs doesn't matter for this problem; it's unpredictable because dictionaries are unordered.)

(c) (5 points) The following sequence of statements prints the most frequently occurring string in the original list `NL`, along with the number of times it occurs. Complete each statement below to produce this result, supplying one identifier, operator, or constant for each blank. (Hint: The problem contains many clues.)

```
def second_item(L: list) -> 'any':
    ''' Return second field (L[1]) of a list, to use with key= in Sort() method
    '''
    return __________ [ __________ ]   L   1   (that's "L" and "one")

list_of_string_frequency_pairs = __________ (__________(NL))   dict_to_list   tally_names
__________ . __________(key=__________, reverse=True)   list_of_s_f_pairs   sort   second_item
most_frequent_pair = __________ [ __________ ]   list_of_string_frequency_pairs   0
print("The string '", __________ [ __________ ], "' occurs ",   most_frequent_pair   0
    __________ [ __________ ], ' times.', sep='')   most_frequent_pair   1
```

(d) (1 point) What data structure (chosen from dictionaries, sets, or tuples) could we use (instead of the two-item key-value list) to represent each word with its frequency? (One word.)

A tuple (unnamed). Instead of ['John', 3], we could use ('John', 3). There's not a major reason to prefer tuples over lists, but technically, tuples are immutable which lets Python store them more simply than it can store (mutable) lists.

(e) (1 point) The result of `tally_names(NL)` in part (a) above is a dictionary with strings (names) as the keys and numbers (frequency counts) as the values.

Describe in just a few English words the meaning of `list(tally_names(NL).keys())`.

A list of the unique names occurring in NL. [For full credit you must say "unique" or "without repetitions".]

[The keys() method gives a view of all the keys in the dict; list() turns the view into a list.]

(f) (3 points) Consider this code:

```
def remove_duplicates(L: [str]) -> [str]:
    ''' Return a list containing the strings in L with no duplicates
    '''
    result = set()
    for s in L:
        result.add(s)
    return list(result)
```

(f.1) What data structure (chosen from dictionaries, sets, and tuples) does this code use? (One word.)

sets

(f.2) Why does this code call `list()` in the `return` statement?

The variable result is a set and the function is supposed to return a list.

(f.3) Would you expect this assertion to pass (i.e., be True)?

```
assert sorted(remove_duplicates(NL)) == sorted(list(tally_names(NL).keys()))
```

Yes. Each side is an alphabetized list of the unique strings in NL.

Both sets and dicts in Python are unordered (i.e., there's no guarantee what order they'll be stored in), but converting each to a list and then sorting the list guarantees the order of each will be the same.

One more useful tip: The set constructor function set() will do the whole remove_duplicates task automatically: This assertion would pass:

```
assert list(set(NL)) == remove_duplicates(NL)
```

Note that set() returns a set and remove_duplicates returns a list.