CS 116 Winter 2013

Assignment 08

Due at 10:00 am on Wednesday, March 27

- All solutions are to be in Python.
- Each question requires the use of loops (while loop and/or for loop) in a nontrivial way.
- You may use the `range` function in any of your solutions.
- Do NOT use recursion in your solutions. Do NOT use abstract list functions in your solutions (which includes sort and sorted).
- Do not import any modules except the `check` module.
- Download the testing module from the course Web page. Include `import check` in each solution file.
- Be sure to use the strings that exactly match those specified on the assignment and interface. Changing them in any way may result in the loss of correctness marks.
- You are encouraged to user helper functions in your solutions as needed. Include them in the same file as your solution, but make helper functions separate functions from the main function, i.e. do NOT make them local functions. You do not need to provide examples and tests for these helper functions.
- Do not copy the purpose directly from the assignment description. The purpose should be written in your own words and include reference to the parameter names of your functions.
- The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.
- Assignments will not be accepted through email. Course staff will not debug code emailed to them.
- Test data for all questions will always meet the stated assumptions for consumed values.
- Read the course Web page for more information on assignment policies and how to organize and submit your work. Follow the instructions in the style guide. Specifically, your solutions should be placed in files `a08qY.py`, where `Y` is a value from 1 to 4.
- Download the interface file from the course Web page.
- Do not cut-and-past examples (or other text) from the assignment description, as this can result in your code failing all tests.
- For full marks, it is not sufficient to have a correct program. Be sure to follow all the steps of the design recipe given in the Python Style Guide for CS116, including the definition of constants and helper functions where appropriate.

Coverage: Module 8
Language: Python

1. Write a function called `count_freqs`, that consumes a list of strings (`los`) and produces a list of string-int pairs (lists of length 2). For each unique string in `los` one corresponding item should be in the produced list. Along with each string in the produced list there should be an integer which represents how many times the string appeared in the consumed list. The strings in the produced list should appear in the same order as the first occurrence of the string in `los`.

For example,

```
count_freqs(["egg", "bacon", "spam", "spam", "eggs", "spam", "bacon",
"egg", "sausage", "egg", "spam", "spam"]) => [["egg", 3], ["bacon",
2], ["spam", 5], ["eggs", 1], ["sausage", 1]]
```

2. Write a function called `collatz_list`, that consumes a positive int (`n`) and produces a list of ints. The produced list of ints will begin with `n`, the value of each number in the produced list will calculated based on the value of the previous number in the list. If the previous number is even, divide that number by two to get the next number; if the previous number is odd, triple that number and add one to it to get the next number. Continue generating numbers until 1 is reached.
   For example,
   ```
   collatz_list(6) => [6, 3, 10, 5, 16, 8, 4, 2, 1]
   ```
   This sequence is based on the Collatz conjecture. No number has ever been found that does not eventually converge to 1. During testing we will only use numbers known to converge to 1.

3. Write a function called `search_documents`, that consumes a list of strings (`documents`) and produces `None`. Each string in `documents` consists of zero or more words separated by spaces. The function will prompt the user (with the prompt "Query: ") for an input query. The query is a string that consists of one or more words. The function will print out each document where **all** the words in the query are found in the document at least once. The function will repeatedly prompt the user for queries until an empty query is entered, at which point the function will end.
   Words in the query and the document match regardless of case (upper or lower case doesn't matter) or punctuation (periods, commas, questions marks, and exclamation marks) at the start or end of words.
   Matching documents should be printed out after each query in the same order as they are found in the consumed list and each match should be printed on a new line.
   For example,
   ```
   search_documents(["The brown dog jumped.", "He jumped over the log.",
   "A dog, cat, or rabbit would make a good pet."])
   Query: jumped the
   The brown dog jumped.
   He jumped over the log.
   Query: dog
   The brown dog jumped.
   A dog, cat, or rabbit would make a good pet.
   ```

Assignment 08

Due at 10:00 am on Wednesday, March 27

(example continued from previous page)

```
Query: own
Query: the dog
The brown dog jumped.
Query:
```

Note:

- The first query prints strings that contain both the words “jumped” and “the” (even though, in the first string, the case for “the” is different and “jumped” is followed by a period).
- The third query prints nothing because no strings contain the word “own” even though it is part of the word “brown” in the first string.
- The fourth query prints the only string that contains both the words “the” and “dog”, even thought all the strings contain at least one of the words in the query.

4. We can represent tables in Python as lists of lists similarly to how we represented tables in Scheme. For example, if we want to represent the table:

X0	Y0
X1	Y1

We could use `[["X0", "Y0"], ["X1", "Y1"]]`.

Write a function called `merge_tables`, that consumes two list of lists (`table1` and `table2`) and produces a list of lists that corresponds to the merged version of the two consumed tables. In order to merge the tables we follow certain rules:

- The first column of both `table1` and `table2` is called the key and the same key value can occur in both tables or in only one table.
- If `table1` has m columns and `table2` has n columns then the produced table will have m+n−1 columns (the key should not be repeated twice in the produced rows).
- The columns in the produced table should be in the same order as all the columns in `table1` followed by all but the first column in `table2`.
- Every row in a table has the same number of columns.
- The rows in the produced table do not have a specified order.
- If the the same key value occurs in both tables the two rows should be merged in the produced table. The key links the two tables.
- There may be multiple rows in `table1` or `table2` which contain the same key. In this case the rows should be merged with all rows from the other table with a matching key. See key “D” in the example below.

Assignment 08

Due at 10:00 am on Wednesday, March 27

- If a key in one table has no match in the other table the value `None` should be placed in the missing column fields. The first column in the produced table should always be the key.
- If both consumed tables are empty then the empty table is produced.
- If only one of the consumed tables is empty then either `table1` or `table2` is produced, whichever one is non-empty.

With the example,

```
merge_tables([["A", 11], ["B", 22], ["D", 33], ["D", 44], ["E", 55]],
[["A", True, 5.0], ["B", False, 8.0], ["C", True, 4.0], ["D", False,
2.0], ["D", True, 1.0]]) => [["A", 11, True, 5.0], ["B", 22, False,
8.0], ["D", 44, False, 2.0], ["D", 44, True, 1.0], ["D", 33, False,
2.0], ["D", 33, True, 1.0], ["C", None, True, 4.0], ["E", 55, None,
None]]
```

we can represent consumed and produced list of lists as tables (see next page):

A	11
B	22
D	33
D	44
E	55

A	True	5.0
B	False	8.0
C	True	4.0
D	False	2.0
D	True	1.0

A	11	True	5.0
B	22	False	8.0
D	44	False	2.0
D	44	True	1.0
D	33	False	2.0
D	33	True	1.0
C	None	True	4.0
E	55	None	None

Note:

- The order of the produced columns.
- The `None` values for rows “C” and “E”.
- Due to the two “D” rows in `table1` and 2 “D” columns in `table2` there are 4 “D” rows in the produced table.
- This is one possible ordering of the rows, different ordering of rows are also correct.