# CS116 Winter 2011 Assignment 9
## Due: Tuesday, March 29 at 10:00AM

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.

**Important Information:**

1. Read the course Style Guide for Python (Appendix C in the course notes) for information on assignment policies and how to organize and submit your work. In particular, your solutions should be placed in files a9qY.py, where Y is a value from 1 to 5.
2. Be sure to download the interface file to get started.
3. You may use helper functions in your solutions as needed. Simply include them in the same file with your solution, but do not declare them locally. You should follow the full design recipe for your helper functions (which includes testing).
4. If you use print statements for debugging purposes, be sure to remove them (or comment them out) before you submit your solutions.
5. Do not copy the purpose directly from the assignment description – it should be in your own words.
6. You may assume that all consumed data will satisfy any stated assumptions, unless indicated otherwise.
7. **For this assignment, you may not use recursion for any of the problems. You must use loops, unless otherwise specified.**
8. For functions that produce a list of items, where there is a *fixed* number of items returned, and the items are of mixed types, you may use the following convention for contracts: `(listof type_a type_b ...)` where every "type_x" refers to a specific item in the list and its type. For example, if a function produces a list with only two elements, and those two elements are always a string and an integer, you can specify the type produced as `(listof str int)`

**Language level:** Any Python in the range 2.5.0 to 2.7.1. (Do **not** use Python 3.0 or higher)

**Coverage**: Modules 8 and 9

## Overview

In this assignment, you will gain experience processing and summarizing data.

**Note:** When examples of dictionaries are given in the questions below, the actual order of key-value pairs in the dictionary may vary from the order when printed out in Python, since the order of keys in a dictionary is not defined by Python.

1. In this question, you will write a function

   ```
   read_marks_line(marks_line)
   ```

   that consumes a non-empty string and produces a list of values (specified below).

   The argument passed in (`marks_line`) is a string representing the mark a student received on a particular assignment or test. For example, a potential `marks_line` would be `"A3 94 20100501 Le   Bon  , Simon"`. The specific contents of this string are described below.

   The list returned must conform to the following specification:
   - The first item is a *string* representing the student's ID
   - The second item is a *string* representing the student's last name
   - The third item is a *string* representing the student's first name
   - The fourth item is a *string* representing the name of the assignment or test that was marked
   - The fifth item is a *float or a string* representing the mark the student got on the item marked. This item should be a float if the student received a numerical grade, or the string `"DNW"` (in all caps) if they did not write the assignment or exam

   To convert `marks_line` into the five elements described above, you will need to convert it into separate *tokens*.

   For this question, we define a token to be a set of contiguous characters *without a space or comma*. For example, the string

   ```
   " some tokens  123   Jones,Tina"
   ```

   has five tokens, `"some"`, `"tokens"`, `"123"`, `"Jones"` and `"Tina"`.

   The `marks_line` string will have a set of tokens conforming to the following specification:
   - The first token will be the name of the assignment or test
   - The second token will be the mark on the assignment or test and will either be an integer, decimal number, or the string `"DNW"`
   - The third token will be the student's ID
   - The next set of tokens, up to a comma (","), will be the student's last name
   - The set of tokens after the comma will the student's first name

   All tokens, except the student's last and first names, will be separated by one or more spaces.

Importantly, there may exist multiple tokens for the student's first or last names; you must use the comma to determine where their last name(s) end and their first name(s) begin. You must also remove the comma separating the last and first names – it should not appear in any string in your returned list.

The strings in the returned list should remove any leading or trailing whitespace, and any extra spaces between names if a student has multiple first and last names.

As mentioned, the mark will be either a number grade or the string `"DNW"`. *If the mark is a number, you must convert it to a float.*

The input string will conform to the following specifications:

- The string will be non-empty
- You are guaranteed to have input matching the specification given above (name of assignment or test, mark, student ID, last name, comma, first name, in that order). However, as mentioned, there may be multiple tokens for the first and/or last names (but there *will not* be multiple tokens for the assignment name, the mark, or ID)
- The whitespace used to separate tokens will be a space, and no other whitespace character will be used (for example, the "tab" character will not be used to separate tokens)
- The mark will either be a number (integer or decimal number) or the string `"DNW"` (in all caps)
- There may be multiple whitespace characters surrounding a token
- There will be only one comma in the string, and it will be used to delimit the student's last and first names
- The comma separating the last name and first name may or may not be preceded or followed by whitespace

**For this question, you *may not* use recursion or loops.** You must split the input string into tokens using string's `split` method (though you can use split multiple times). You will also find the `join` method of strings useful. (The `strip` method of strings may also be useful, but is not essential to solving this problem.)

Examples:

```
read_marks_line("A3 94 20100501 Le   Bon  , Simon")
# produces ["20100501", "Le Bon", "Simon", "A3", 94.0]

read_marks_line("midterm DNW 19991007 Thornton, Billy Bob ")
# produces ["19991007", "Thornton", "Billy Bob", "midterm", "DNW")

read_marks_line("   A4    13   19900101 Ng,Mark  ")
# produces ["19900101", "Ng", "Mark", "A4", 13.0]
```

2. For this question, you will write a function

   `convert_marks(list_of_marks, read_marks_line_fn)`

   that consumes a non-empty list of non-empty strings, and a function. It produces a dictionary.

   `convert_marks` will do the following:

   - For every string in `list_of_marks`, it will pass the string to the function `read_marks_line_fn`. The function `read_marks_line_fn` will match the specifications of the function in question 1: It will consume a string and return a list that contains the student's ID, their last name, first name, the test/assignment name, and their mark on the test/assignment, in that order
   - `convert_marks` will use the list returned from `read_marks_line_fn` to update a dictionary (which it must create and return)
   - The dictionary that it returns has the following characteristics:
     - The keys will be the students' IDs
     - The values will be a second dictionary with the following characteristics:
       - The keys will be strings representing a test or assignment name
       - The values will be the mark the student received on the test or assignment
     - Note that the students' first and last names *will not* appear in the dictionary produced, even though `read_marks_line_fn` returns a list that includes the student's first and last names
   - After reading all strings in `list_of_marks`, the function will return the dictionary created

   Example:

```
marks = ["A3 94 20100501 Le Bon, Simon",
"midterm DNW 19991007 Thornton, Billy Bob",
"A4    13   19900101 Ng,Mark",
"A3 95 19900101 Ng, Mark",
"A3 DNW 19991007 Thornton, Billy Bob"]

convert_marks(marks, read_marks_line)
# Produces the following dictionary:
{
"20100501" : { "A3" : 94.0 },
"19991007" : { "midterm" : "DNW",
               "A3" : "DNW" },
"19900101" : { "A4" : 13.0,
               "A3" : 95.0 }
}
```

   (Recall that the order of the key-value pairs may differ because Python does not guarantee the order of key-value pairs when printing out a dictionary.)

You may make the following assumptions about the strings contained in `list_of_marks`:

- For any given student ID, there will be at most one entry per assignment/test name. For example, given a student ID "19900101", there will only be one entry for assignment A1
- You only need to create entries in the dictionary corresponding to the data in the list
- Each string will conform to the specifications given in question 1 for the strings passed to the `read_marks_line` function

You should use the function derived from question 1 as input to this question when testing your solution. (We will test `convert_marks` with our own version of the `read_marks_line` function.)

3. For this question, you will create a function

```
get_marks(marks_dictionary, assessment_name)
```

that consumes a non-empty dictionary and a non-empty string, and produces a *sorted* list of floats (sorted in ascending order).

The dictionary follows the specification of the dictionary produced in question 2:

- The keys are non-empty strings that correspond to student IDs
- The values are non-empty dictionaries that conform to the following specification:
  - The keys are non-empty strings that correspond to the name of an assignment or test
  - The values are a single mark, either a float or the string `"DNW"` (in all caps)

The argument `assessment_name` corresponds to the name of an assignment or exam.

Your function should extract the marks students received for the assignment or exam represented by `assessment_name` and append them to a list that is returned by the function. The list should only contain those marks that are floats – you should filter out all `"DNW"` values. The list should be sorted (in ascending order) when it is returned by the function.

Note that not every student may have a mark for `assessment_name`. If a student does not have a mark, you should not add anything to the list returned for that student.

Example:

```
marks = {
"20100501" : { "A3" : 94.0 },
"19991007" : { "midterm" : "DNW",
               "A3" : "DNW" },
"19900101" : { "A4" : 13.0,
               "A3" : 95.0 }
}

get_marks(marks, "A3")
# will produce: [94.0, 95.0]
```

4. In this question, you will write a function

    ```
    summarize_data(list_of_marks)
    ```

    that consumes a non-empty list of integers in the range 0-100, inclusive, and produces a dictionary with the following characteristics:
    - ○ The key will be a string
    - ○ The value will be a non-empty *sorted list of strings*

    You will convert the `list_of_marks` argument into a dictionary as follows:
    - For each mark, you will convert it into two strings:
        - ○ The first string will consist of the string representing all digits of the number up to, but not including, the last digit
        - ○ The second string will be the last digit of the number
        - ○ For example, the number 37 will be split into the strings `"3"` and `"7"`
        - ○ Similarly, the number 100 will be converted to the strings `"10"` and `"0"`
    - If the number to convert is less than ten, you should make the first string `"0"` and the second string the string representing the number. For example, 9 will turn into `"0"` and `"9"`. The number 0 should be represented by `"0"` and `"0"`

    Given these two strings, you will then add them to the returned dictionary as follows:
    - ○ The first part of the string will be the key in the dictionary
    - ○ The second part of the string will be appended to a list associated with the key in the dictionary. This list must be sorted in ascending order.

    As an example:

    ```
    summarize_data([27, 37, 25, 9, 13, 25])
    ```

    will produce a dictionary with the following entries:

    ```
    {
      "0" : ["9"],
      "1" : ["3"],
      "2" : ["5", "5", "7"],
      "3" : ["7"]
    }
    ```

5. In this final question, you will produce a stem-and-leaf plot. A stem-and-leaf plot is a basic but extremely useful way of visualizing data; it looks very similar to a histogram turned on its side.

   A stem-and-leaf plot represents data in a series of rows and columns. Each row represents a set of values in a given range. For example, a row may represent all of the data between the values 10 and 19, inclusive.

   The stem-and-leaf plot has two columns, separated by a vertical line. The left column represents the start of a number. The right column represents the final digit of the number.

   Each data point (that is, each number) is represented in a stem-and-leaf plot by splitting it into two parts, using precisely the same strategy used in the previous question. The last digit of the number is represented in the right column, with the preceding digits represented in the left column. Each data point has a corresponding entry in the right column.

   As an example, the stem-and-leaf plot of the numbers 9, 13, 25, 25, 27, and 37 is this:

```
0 | 9
1 | 3
2 | 5 5 7
3 | 7
```

   For this question, you will complete the function

```
stem_and_leaf(marks_summarization)
```

   that consumes a non-empty dictionary, and produces a list of strings representing the stem-and-leaf plot for the data. Your list will have *exactly* 11 strings, representing the 11 different rows of the stem-and-leaf plot for data with values between 0 and 100, inclusive.

   The dictionary represented by `marks_summarization` corresponds to the dictionary produced by the function in the previous question:
   - The keys are strings representing numbers
   - The values are non-empty lists of non-empty strings representing single digits, sorted in ascending order

   The data contained in `marks_summarization` will represent student marks in the range 0-100, inclusive.

   As mentioned, your function will produce a stem-and-leaf plot representing 11 rows. However, there may not be any data for a given row. For example, no students may have received marks in the range 10-19. Accordingly, there may be no key in the dictionary for that range. However, your function should still produce a string for this range, as described below.

Each string in the returned list (representing a row in the plot) will have the following elements:

- The beginning part of the number for the given range
- The vertical bar ( | ), with a space before it. If there is data for that row, the vertical bar should have a space after it. If there is no data after the vertical bar, there should be no space after it.
- The digits representing the data/marks in that range, with each digit separated by a space

Importantly, the vertical bar must appear at the same index in each returned string. Thus, if the range represented is only a single digit, the string should start with a space, to ensure the vertical bar lines up across all rows.

Remember that there may not exist data in the dictionary for a particular range. You must still make a string representing that data range in the stem-and-leaf plot. However,  there should be no space after a vertical bar if there are no data in that range.

The strings returned should be sorted in ascending order according to the number represented in the left column.

As an example:

```
marks_summary =
{
  "0" : ["9"],
  "1" : ["3"],
  "2" : ["5", "5", "7"],
  "3" : ["7"]
}

stem_and_leaf(marks_summary)
# Produces the following:
[" 0 | 9",
 " 1 | 3",
 " 2 | 5 5 7",
 " 3 | 7",
 " 4 |",
 " 5 |",
 " 6 |",
 " 7 |",
 " 8 |",
 " 9 |",
 "10 |"]
```