CSE 101: Introduction to Computational and Algorithmic Thinking

Lab #5

Fall 2018

Assignment Due: Saturday, October 20, 2018, by 11:59 pm

Assignment Objectives

By the end of this assignment you should be able to design, code, run and test original Python functions that solve programming problems involving `if` statements, strings, lists, and `for` and `while` loops.

Getting Started

The assignments in this course require you to write Python code to solve computational problems. To help you get started on each assignment, we will give you a "bare bones" file with a name like `lab5.py`. These files will contain *function stubs* and a few tests you can try out to see if your code seems to be correct. Stubs are functions that have no bodies (or have very minimal bodies). You will fill in the bodies of these functions for the assignments. **Do not, under any circumstance, change the names of the functions or their parameter lists.** The automated grading system will be looking for exactly those functions provided in `lab5.py`. **Please note that the test cases we provide you in the bare bones files will not necessarily be the same ones we use during grading!**

Directions

Solve the following problems to the best of your ability. This assignment has a total of 4 problems, worth a total of 16 points in all. The automated grading system will execute your solution to each problem several times, using different input values each time. Each test that produces the correct/expected output will earn 1 point. You must earn at least 12 points in order to pass (receive credit for) this lab.

- At the top of the `lab5.py` file, include the following information in comments, with each item on a separate line:
  - your full name ***AS IT APPEARS IN BLACKBOARD***
  - your Stony Brook ID #
  - your Stony Brook NetID (your Blackboard username)
  - the course number (CSE 101)
  - the assignment name and number (Lab #5)
- ⚠ Each of your functions must use the names and parameter lists indicated in the starter code file. Submissions that have the wrong function names (or whose functions contain the wrong number of parameters) can't be graded by our automated grading system, and may receive a grading penalty (or may not be graded at all).
- ⚠ Be sure to submit your final work as directed by the indicated due date and time. Late work will not be accepted for grading. Work is late if it is submitted after the due date and time.
- ⚠ Programs that crash will likely receive a grade of zero, so make sure you thoroughly test your work before submitting it.

Part I: You Haven't Seen the Last of Me! (4 points)

Complete the `removeLast()` function, which takes two string arguments. This function returns a new string based on the first string argument, with the following modification: the *last* occurrence of each letter from the second string has been removed (if a given letter appears *n* times in the second string, up to the last *n* occurrences of that letter will be removed from the first string, depending on the number of times that letter appears in the first string).

Your basic strategy should be to process the *second* string argument, character by character. For each character, locate its last occurrence within the first string argument (we suggest using Python's `rfind()` method, described in the lecture slides). If the character is present, remove the rightmost copy of that character from the first string argument. When you have processed every character in the second string argument, return whatever is left of the first string argument. Please note that:

- characters in the second string are ***NOT*** guaranteed to appear in the first string in the same numbers (or at all)
- there is **NO** guarantee regarding the relative lengths of the two strings
- `removeLast()` is CASE-SENSITIVE; it only removes characters from the first string if they appear in the same case as they do in the second string

For example, consider the strings "quicksilver" and "blink":

1. The first letter of "blink" ("b") does not appear in "quicksilver", so we move on
2. The second letter of "blink" ("l") appears in "quicksilver", so we remove the last (and in this case, only) occurrence to get "quicksiver"
3. The third letter of "blink" ("i") appears twice in "quicksiver"; we remove the last (rightmost) instance to get "quicksver"
4. The fourth letter of "blink" ("n") does not appear in "quicksver", so we move on
5. The final letter of "blink" ("k") appears in "quicksver", so we remove it to get "quicsver" as our final answer

Hint: Remember that Python strings are *immutable*; they cannot be modified directly. In order to remove a character from a string, you will need to construct a new string (most likely using slicing) and assign it to the old string variable, replacing the original. We have provided a stub for a helper function named `excise()` that you may wish to call from within your `removeLast()` function. It takes a string and an integer (representing the index of a character to remove) as its arguments. If the index is less than 0 or greater than or equal to the length of the string argument, the function returns the string argument unchanged. Otherwise, it uses slicing to return a new string that contains all of the characters from the beginning of the string up to (but not including) the specified index, followed by all of the characters following the specified index.

Examples:

Function Call	Return Value
`removeLast("starlord", "thor")`	`"sarld"`
`removeLast("graymalkin", "rag")`	`"aymlkin"`
`removeLast("Booster Gold", "Aquaman")`	`"Booster Gold"`

Part II: Credit Card Statements (4 points)

Complete the `statement()` function, which takes two arguments: a starting credit card balance (a floating-point number) and a list of transactions (note that the second argument is actually a *list of lists*; each element is a 2-element list whose first item is a string indicating the type of transaction and whose second item is a number to apply to the current account balance according to the rules below). The function returns the final balance after every transaction has been applied.

- A "credit" transaction subtracts the associated value from the current balance
- A "debit" transaction adds the associated value to the current balance
- An "interest" transaction multiplies the current balance by (1 plus its associated value) (for example, .04 represents a 4% interest rate)
- Any other transaction label is ignored

For example, suppose we have a starting balance of 105.92 and the following list of transactions:

```
[["debit", 65.01], ["debit", 14.87], ["deposit", 61.30], ["interest", .16], ["credit",
29.45]]
```

We would perform the following sequence of adjustments:

1. The first transaction is ["debit", 65.01]. Its first element is "debit", so we add 65.01 to our balance to get 170.93
2. The second transaction is ["debit", 14.87], so we add 14.87 to our current balance to get 185.80
3. The third transaction is ["deposit", 61.30]. "deposit" isn't a recognized keyword, so we do nothing.
4. The fourth transaction is ["interest", .16], so we multiply the current balance by 1.16 (1 plus the second list item) to get 215.528
5. The fifth transaction is ["credit", 29.45], so we subtract 29.45 from the current balance to get a final answer of 186.078 (don't worry about rounding the result to 2 decimal places)

Examples:

Function Call	Return Value
`statement(219.73, [["credit", 102.88], ["credit", 468.64], ["interest", 0.23], ["debit", 128.48]])`	`-304.22169999999994`
`statement(40.22, [["debit", 474.23], ["interest", 0.32], ["debit", 452.68], ["debit", 158.95], ["interest", 0.34], ["credit", 34.47]])`	`1695.0733600000003`
`statement(1220.61, [["credit", 393.09], ["credit", 62.42], ["credit", 284.84], ["payment", 88.19], ["debit", 153.12], ["payment", 122.98], ["credit", 69.1], ["interest", 0.06]])`	`598.1368000000001`

Part III: Laser Limbo (4 points)

In the International Laser Limbo World Championship, each contestant is scored by a panel of seven judges. Each judge is from a different country, and each score is a number in the range 1.0 through 10.0 (inclusive). Note that the panel **may** or **may not** contain a judge from the same country as the contestant. The contestant's final score is calculated as follows:

1. If a judge is from the same country as the contestant, then that judge's score is added *once* to the contestant's point total, along with a fixed "neutral" score of 5.0. If the judge is from a different country than the contestant, then the judge's score is added *twice* to the point total.
2. The final score is equal to the total points earned, divided by 14.

Complete the `limboScore()` function, which takes two arguments: a list of scores (each score is represented by a 2-element list; each element or sub-list contains a country name (a string) and the numerical score from that country's judge) and a string representing the contestant's country. This function should compute and **return** (not print!) the contestant's final score according to the rules above (don't worry about rounding the result to any specific number of decimal places).

For example, suppose that you have the following list of seven scores:

```
scores = [ ["USA", 4.4], ["Canada", 8.2], ["Russia", 7.0], ["China", 7.7],
           ["England", 10.0], ["France", 7.8], ["Germany", 1.0] ]
```

`limboScore(scores, "China")` indicates that the contestant is from China (which has a judge on the current panel). The contestant's point total is (2 * 4.4) + (2 * 8.2) + (2 * 7.0) + (7.7 + 5.0) + (2 * 10.0) + (2 * 7.8) + (2 * 1.0), or 89.5. This function call would return 6.39, which is equal to 89.5 divided by 14.

`limboScore(scores, "Mexico")` indicates that the contestant is from Mexico (which does not have a judge on the current panel). The contestant's point total is (2 * 4.4) + (2 * 8.2) + (2 * 7.0) + (2 * 7.7) + (2 * 10.0) + (2 * 7.8) + (2 * 1.0), or 92.2. This function call would return 6.5, which is equal to 92.2 divided by 14.

Examples:

Function Call	**Return Value**
`limboScore([["Kenya", 1.8], ["France", 7.8], ["Ethiopia", 1.4], ["India", 5.7], ["South Korea", 8.9], ["Norway", 3.4], ["Morocco", 4.1]], "Thailand")`	`4.728571428571429`
`limboScore([["USA", 8.4], ["Finland", 2.5], ["Germany", 7.5], ["Jamaica", 1.5], ["Russia", 3.6], ["Poland", 7.9], ["South Africa", 3.7]], "Egypt")`	`5.014285714285714`
`limboScore([["Norway", 1.4], ["Aruba", 1.3], ["Russia", 8.4], ["Jamaica", 5.5], ["Spain", 2.6], ["USA", 3.3], ["Morocco", 3.0]], "Spain")`	`3.8142857142857145`

Part IV: Partial Hostname Extraction (4 points)

A *Uniform Resource Locator* (URL) is a unique address for a document on the World-Wide Web. A URL has the following form:

scheme://hostname/path/filename

- *scheme* is a protocol such as `http`, `ftp`, `irc` or `file`
- *hostname* consists of 1 or more alphanumeric strings (with length 1 or more) followed by single periods, followed by a *top-level domain* (TLD) like `com`, `edu`, or `net`
- *path* is optional, and consists of 1 or more alphanumeric strings (with length 1 or more), each followed by a forward slash
- *filename* is also optional, and is an alphanumeric string (of length 1 or more)

URL Examples

```
irc://foo.b25.com/readme.txt
file://bar.edu/path/to/file.html
ftp://12q.net/
```

Complete the `getHost()` function, which takes a single string argument representing a URL and returns a string that corresponds to the *next-to-last* section of the hostname. For example, given the URL "http://www.example.com/", the function would return the string "example". Given the URL "ftp://this.is.a.long.name.net/path/to/some/file.php", the function would return the string "name". While the *path* and *filename* sections of the URL are optional, you may assume that the full hostname is always followed by a single forward slash ("/").

The easiest way to solve this problem is by using `find()` and `rfind()` to strategically and systematically remove pieces of the original string:

1. Start by locating and removing the *scheme* section, along with its trailing "://" substring.
2. Next, locate the first forward slash and remove it and everything after it, to reduce the source string to just the (full) hostname.
3. Third, locate the last period in the hostname, and remove it and everything that follows (to eliminate the TLD).
4. If the remaining string still contains at least one period, find the last period and remove everything before it (and the last period itself).

Examples:

Function Call	**Resulting String**
`getHost("irc://foo.com/")`	`foo`
`getHost("http://i.am.a.hostname.edu/blah/blah/")`	`hostname`
`getHost("ftp://some-like.it-hot.net/something/filename.pdf")`	`it-hot`