CSE 101: Computer Science Principles (Fall 2019)

Lab #6

Assignment Due: Saturday, November 2, 2019, by 11:59 PM

Assignment Objectives

By the end of this assignment you should be able to develop original Python functions to solve simple programming problems involving lists and strings.

Getting Started

This assignment requires you to write Python code to solve several computational problems. To help you get started, we will give you a basic starter file for each problem. These files will contain *function stubs* and a few tests you can try out to see if your code seems to be correct (**note that the test cases we give you in the starter files are just examples; we will use different test inputs to grade your work!**). You need to complete (fill in the bodies of) these functions for the assignments. **Do not, under any circumstances, change the names of the functions or their parameter lists.** The automated grading system will be looking for functions with those exact names and parameter lists; if you change any function headers, the grading program will reject your solution and mark it as incorrect.

Directions

Solve each of the following problems to the best of your ability. We have provided a code skeleton for each of the programming problems. The automated grading system will execute your solution to each programming problem several times, using different input values each time. Each test that produces the correct/expected output will earn 1 or more points. This assignment contains 3 problems, and is worth a total of 30 points. **Note that not every problem may be worth the same number of points!**

- ⚠ Each starter file has a comment block at the top with various important information. Be sure to add your information (name, ID number, and NetID) to the first three comment lines, so that we can easily identify your work. **If you leave this information out, you may not receive credit for your work!**
- ⚠ Submit your work as a set of individual files (one per problem). **DO NOT** zip or otherwise compress your files, and **DO NOT** place all of your work in a single file. If you do so, we may not be able to grade your work and you will receive a failing grade for this assignment!
- ⚠ Every function **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).
- ⚠ Every function must explicitly ***return*** its final answer; the grading program will ignore anything that your code prints out. Along those lines, do **NOT** use `input()` anywhere within your functions (or anywhere before the `if __name__ == "__main__":` statement); your functions should get all of their input from their parameters. Programs that crash will likely receive a failing grade, so test your code thoroughly **with Python 3.7.4 or later** before submitting it.
- ⚠ Blackboard will provide information on how to submit this assignment. You **MUST** submit your completed work as directed by the indicated due date and time. We will not accept (or grade) any work that is submitted after the due date and time, or that is submitted before you have signed the course's Academic Honesty agreement.
- ⚠ **ALL** of the material that you submit (for each problem) **MUST** be your own work! You may not receive assistance from or share any materials with anyone else, except for the instructor and the (current) course TAs.

Part I: Duplicate Removal (10 points)

(Place your answer to this problem in the "remove_duplicates.py" file)

Function Summary

Parameters: A string that only contains lowercase letters and digits.

Return Type: string of lowercase letters and digits based on the parameter.

Complete the `remove_duplicates()` function, which takes one string argument. This function returns a new string based on the original string, with all duplicate characters have been removed (more specifically, only the ***first*** occurrence of a character is left; later copies of that character **do not** appear in the result). For example, `"aaaaaaaah"` becomes `"ah"`.

Hint: One way to solve this problem is to start with an empty list. For each character, check to see if it's in your list. If it isn't there, append it to both the output string and your list; if it's already in your list, don't do anything.

Examples:

Function Call	Return Value
`remove_duplicates("aaaaaaaah")`	`"ah"`
`remove_duplicates("abcd")`	`"abcd"`
`remove_duplicates("csee101")`	`"cse10"`
`remove_duplicates("byyyyeeeeeseeeyouuulaterrrrr")`	`"byesoulatr"`

Part II: Shopping Expedition (10 points)

(Place your answer to this problem in the "shopping.py" file)

Function Summary

Parameters: Two lists. Each list contains an arbitrary number of two-element lists.

Return Type: An integer if the function is successful, or a string if it is not.

Complete the `shopping()` function, which takes two lists as its parameters. The first list represents the store's inventory, and contains a series of two-element lists; each sublist consists of a string (an item name) and an integer (the item price), e.g., `["eggs", 3]`. The second list represents a shopping list that consists of two-element lists; each sublist contains a string (the name of an item to purchase) and an integer (the quantity of that item to buy), e.g., `["stapler", 1]`.

If all of the items named on the shopping list (the second parameter) are available for sale (i.e., appear somewhere in the first list), the function should return the total purchase price for all of the items on the shopping list. If at least one item from the shopping list does not appear in the store's inventory, the function should return the string "invalid purchase" instead.

Hint: Use a nested list for this problem. For each item in the shopping cart, search through the inventory for that item. If the item is present, record that fact and update your total. If the item is not present in the inventory, return an error message.

Examples:

Function Call	Return Value
`shopping([["phone", 800], ["laptop", 1000], ["earphone", 200], ["watch", 500]], [["phone", 2], ["laptop", 1]])`	`2600`
`shopping([["phone", 800], ["laptop", 1000], ["earphone", 200], ["watch", 500]], [["phone", 2], ["charger", 1]])`	`invalid purchase`

Part III: No Free Lunch (10 points)

(Place your answer to this problem in the "lunch.py" file)

Function Summary

Parameters: A list whose first element is a number. The second and subsequent elements of the list are two-element lists, each of which consists of a string and a number.

Return Type: A number.

Lunch can be a complicated affair; at least, determining the final bill is complicated. Once the bill arrives, I need to perform a series of calculations to figure out how much I ultimately owe.

Complete the `lunch()` function. This function takes a single argument: a list whose first element is a numerical value (the base price of my lunch order) and whose remaining elements (if any) are 2-element sublists that modify the base price as follows:

- A sublist whose first element is the string "tip" means to add a tip of the given percentage (the second element) to the total so far.
- A sublist whose first element is the string "extra" means to add the specified amount (the second element) to the total so far.
- A sublist whose first element is the string "coupon" means to subtract the specified amount (the second element) from the total so far.
- A sublist whose first element is the string "discount" means to subtract the specified percentage (the second element) from the total so far.

There may be any number of tips, extras, coupons, and discounts in the list. They **MUST** be applied in the order that they appear. The function returns the final bill amount, or 0 if the final amount is less than 0 (meaning that my lunch was free).

Example 1: The list

```
[12.34, ["tip", 15], ["discount", 25], ["extra", 1.5]]
```

means that we start with a base price of $12.34. We then multiply by 1.15 to add the 15 percent tip, giving us $14.191 (don't worry about rounding for this problem). Next, we apply a 25 percent discount by multiplying by 0.75 to get $10.64325. Finally, we add an extra charge of 1.5 to the bill, for a final total of $12.14325.

Example 2: The list

```
[10, ["tip", 20], ["discount", 20], ["coupon", 2], ["coupon", 2], ["coupon", 1],
["discount", 50], ["coupon", 2.55]]
```

means that we start with a base price of $10. We then multiply by 1.20 to add a 20 percent tip, getting $12. Next, we multiply by 0.80 to apply a 20 percent discount, getting $9.6. We apply three coupons in turn, subtracting $2, $2, and then $1, to get $4.6. Applying another 50 percent discount gives us $2.3. Finally, we subtract $2.55 for one final coupon, giving us a final value of -$0.25. This value is less than 0, so the function will return 0 (and they say there's no such thing as a free lunch!).

Examples:

Function Call	Return Value (unrounded)
`lunch([29.77, ["tip", 1]])`	`30.0677`
`lunch([40.32])`	`40.32`
`lunch([40.41, ["extra", 9.55], ["discount", 15]])`	`42.465999999999994`
`lunch([19.34, ["discount", 69], ["tip", 62], ["extra", 2.53], ["coupon", 5.34], ["extra", 3.25]])`	`10.152548000000001`
`lunch([4.55, ["discount", 11], ["coupon", 5.63], ["coupon", 6.28], ["discount", 61], ["extra", 2.42], ["tip", 65], ["discount", 40], ["tip", 6]])`	`0`