CSE 101: Introduction to Computational and Algorithmic Thinking

Stony Brook University

Lab Assignment #3

Spring 2018

Assignment Due: February 16, 2018 by 11:59 pm

Assignment Objectives

This lab assignment will give you practice with if-statements, lists and relational operators. You will also debug and fix a buggy function.

Getting Started

Visit Piazza and download the "bare bones" file `lab3.py` onto your computer. Open `lab3.py` in PyCharm and fill in the following information at the top:

1. your first and last name as they appear in Blackboard
2. your Net ID (e.g., jsmith)
3. your Stony Brook ID # (e.g., 111999999)
4. the course number (CSE 101)
5. the assignment name and number (Lab #3)

Do not, under any circumstances, change the names of the functions or their argument lists. The automated CodeLoad testing system will be looking for exactly those functions provided in `lab3.py`. You will be able to test your work by uploading your file to CodeLoad.

Submit your final `lab3.py` file to Blackboard by the due date and time. Late work will not be graded.

Code that crashes and cannot be graded will earn no credit. It is your responsibility to test your code by running it through CodeLoad and by creating your own test cases.

Part I: String Sanitizer (20 points)

Write a function `sanitize()` that takes one argument, `st`, which is a string that is *supposed to* contain only *uppercase letters*, *lowercase letters* and *digits* (numbers in the form of a string).

The function should analyze the string, then separate uppercase letters, lowercase letters and digits into three different *result strings*. In the end, the function puts these three strings into a single list, and returns the list.

Note: Please make sure the order of the strings in your final list matches this order:

```
[string of uppercase letters, string of lowercase letters, string of digits]
```

You should also make sure that when you construct the result strings, you should read `st` from left to right. Please check the examples below for more detail.

If you encounter a character that's neither an *uppercase letter*, nor a *lowercase letter* nor a *digit*, your function should simply ignore that character and proceed through the remainder of the string.

Hint: You may find the following methods helpful. Please also consult the lecture slides.

- Python `string.isupper()` method (https://www.tutorialspoint.com/python/string_isupper.htm)
- Python `string.islower()` method (https://www.tutorialspoint.com/python/string_islower.htm)
- Python `string.isdigit()` method (https://www.tutorialspoint.com/python/string_isdigit.htm)
- Python `string.isalpha()` method (https://www.tutorialspoint.com/python/string_isalpha.htm)

Examples:

Function Call	Return Value
`sanitize('kBfmR@93&BvS')`	`['BRBS', 'kfmv', '93']`
`sanitize('zs4*ia0QFer5')`	`['QF', 'zsiaer', '405']`
`sanitize('@YIc!8pY14nM')`	`['YIYM', 'cpn', '814']`
`sanitize('4utb8Z')`	`['Z', 'utb', '48']`
`sanitize('7P8VC508@e2LTh')`	`['PVCLT', 'eh', '785082']`
`sanitize('2YERJ3pKPZ9qU7nxR1')`	`['YERJKPZUR', 'pqnx', '23971']`
`sanitize('B4MP22EY4L6')`	`['BMPEYL', '', '42246']`
`sanitize('8um̂3bv2gtq3U6wW8O9f')`	`['UWO', 'umbvgtqwf', '8323689']`
`sanitize('Z9NYRvfHx%S0P̂')`	`['ZNYRHSP', 'vfx', '90']`
`sanitize('IA8mH!R')`	`['IAHR', 'm', '8']`

Note: The quotation marks displayed in the example return values are there to emphasize that the return values are strings. You should not add quotation marks to your return values.

Part II: Party Cost Calculator (20 points)

Write a function `party()` that takes two arguments, in the following order:

1. `prices`: A list of integers that serves as a price book. This list always has four positive integers, with each element representing the price of a particular item, and always in this order:
   - The first element of the list (`prices[0]`) represents the cost of one `'pizza'`.
   - The second element of the list (`prices[1]`) represents the cost of one `'cup'`.
   - The third element of the list (`prices[2]`) represents the cost of one `'plate'`.
   - The fourth element of the list (`prices[3]`) represents the cost of one `'soda'`.
2. `shopping_list`: A list of strings that represents a shopping list. Each item is always one of `'pizza'`, `'cup'`, `'plate'` or `'soda'`. When an item appears multiple times, it means that multiple units of the item are bought. You may assume that the list will not contains any strings other than these four.

The function calculates and returns the total cost of the shopping list. However, there are a few discounts that *might* apply:

1. If five or more sodas are bought, apply an 8% discount to the total cost of all sodas. This discount can be used in addition to the *entire shopping list discount*, described next.
2. Only one of the following discounts to the entire cost might apply:
   a. If exactly two pizzas are bought, apply a 10% discount to the *total cost of the entire shopping list*. This discount applies to the total cost *after* discount #1 is considered.
   b. If three or more pizzas are bought, apply a 20% discount to the *total cost of the entire shopping list*. This discount applies to the total cost *after* discount #1 is considered.

   Again, only one of discounts 2(a) or 2(b) can be applied, not both.

An Example in Detail:

Function call: `party([15, 1, 3, 2], ['soda', 'soda', 'pizza', 'cup', 'cup', 'soda', 'pizza', 'plate', 'soda', 'soda', 'soda'])`

Sort it out a little bit, we have:

- `'pizza'` $\times$ 2, at \$15 each. We have a 10% discount on the entire cost.
- `'cup'` $\times$ 2, at \$1 each
- `'plate'` $\times$ 1, at \$3 each
- `'soda'` $\times$ 6, at \$2 each. We have an 8% discount on the soda.

Calculate the total price *before* the 10% total discount is applied:

$$(15 \times 2) + (2 \times 1) + (1 \times 3) + (6 \times 2) \times 0.92 = 46.04$$

Now we can apply the 10% total discount, and get our final return result:

$$46.04 \times 0.9 = 41.436$$

Examples:

Function Call	Return Value
`party([[13, 1, 1, 1], ['plate', 'soda', 'soda', 'plate', 'plate', 'pizza']])`	`18`
`party([[10, 1, 2, 1], ['soda', 'pizza', 'pizza', 'soda', 'plate', 'plate', 'cup', 'soda', 'cup', 'cup', 'plate', 'soda', 'cup']])`	`30.6`
`party([[20, 2, 2, 3], ['soda', 'soda', 'soda', 'plate', 'cup', 'pizza', 'soda', 'soda', 'soda', 'plate', 'soda', 'pizza', 'plate']])`	`60.587999999999994`
`party([[13, 2, 4, 2], ['cup', 'plate', 'plate', 'soda', 'cup', 'soda', 'plate', 'soda', 'plate', 'soda', 'plate', 'cup', 'pizza', 'plate', 'cup', 'soda', 'cup', 'cup']])`	`58.2`
`party([[11, 2, 3, 2], ['plate', 'plate', 'plate', 'cup', 'soda', 'pizza', 'plate', 'plate', 'soda', 'pizza', 'pizza']])`	`43.2`
`party([[12, 2, 2, 3], ['soda', 'cup', 'cup', 'soda', 'soda', 'cup', 'cup', 'cup']])`	`19`
`party([[18, 1, 4, 2], ['soda', 'cup', 'plate', 'soda', 'cup', 'soda', 'soda', 'soda', 'plate', 'cup', 'soda', 'pizza', 'plate', 'cup', 'soda', 'soda', 'cup', 'plate', 'soda', 'pizza']])`	`66.20400000000001`
`party([[16, 2, 4, 1], ['soda', 'soda', 'cup', 'cup', 'cup', 'plate', 'pizza', 'pizza', 'plate', 'pizza', 'plate', 'cup', 'cup', 'soda']])`	`58.400000000000006`
`party([[19, 1, 1, 1], ['soda', 'pizza', 'cup', 'plate', 'soda', 'cup', 'plate', 'cup', 'pizza', 'soda', 'soda', 'soda', 'pizza', 'cup']])`	`54.08`
`party([[19, 2, 3, 3], ['soda', 'plate', 'cup', 'cup', 'soda', 'cup', 'plate', 'soda', 'soda', 'soda', 'soda', 'pizza', 'plate', 'soda', 'pizza', 'soda', 'plate', 'plate', 'cup']])`	`74.772`

Part III: Debug the Code: Which School Will Junior Attend? (20 points)

In this part you are given some broken code to fix. You are free to just re-write the whole function yourself, but it is a good time to practice using the debugger.

Fix the function `school_dilemma()` that takes a single argument, `age`, which represents the age of a person. You may assume that `age` is a non-negative integer.

The function is supposed to return which school the person is best suited for, depending on their age:

- if `age` is less than `6`, then return `'Too young for school'`.
- if `age` is more than or equal to `6` and less than `11`, then return `'Elementary school'`.

- if `age` is more than or equal to `11` and less than `14`, then return `'Middle school'`.
- if `age` is more than or equal to `14` and less than `18`, then return `'High school'`.
- if `age` is more than or equal to `18`, then return `'College'`.

Remember that Python is a case-sensitive language. Lowercase and uppercase letters are not the same symbols!

Examples:

Function Call	Return Value
`school_dilemma(16)`	`'High school'`
`school_dilemma(3)`	`'Too young for school'`
`school_dilemma(44)`	`'College'`
`school_dilemma(7)`	`'Elementary school'`
`school_dilemma(2)`	`'Too young for school'`
`school_dilemma(4)`	`'Too young for school'`
`school_dilemma(24)`	`'College'`
`school_dilemma(37)`	`'College'`
`school_dilemma(13)`	`'Middle school'`
`school_dilemma(32)`	`'College'`

Note: The quotation marks displayed in the example return values are there to emphasize that the return values are strings. You should not add quotation marks to your return values.

How to Submit Your Work for Grading

To submit your `.py` file for grading:

1. Login to Blackboard and locate the course account for CSE 101.
2. Click on "Assignments" in the left-hand menu and find the link for this assignment.
3. Click on the link for this assignment.
4. Click the "Browse My Computer" button and locate the `.py` file you wish to submit. Submit only that one `.py` file.
5. Click the "Submit" button to submit your work for grading.

Oops, I messed up and I need to resubmit a file!

No worries! Just follow the above directions again. We will grade only your last submission.