15-110 Fall 2019
Hw 05

Out: Friday 4th October, 2019 at 18:00 AST
Due: Thursday 10th October, 2019 at 17:00 AST

Introduction

In this homework you will practice with lists and tuples.
The total number of points available from the questions is 110. 10 points are *bonus* points (i.e., you only need 100 points to get the maximum grade).
In your `.zip` file (see general instructions below), you need to include only the file `hw05.py` with the python functions answering the questions. In the handout you have found a file `hw05.py` with the functions already defined but with an empty body (or partially filled). You have to complete the body of each function with the code required to answer to the questions.

General Instructions for Submitting the Assignments

Submissions are handled through Autolab, at `https://autolab.andrew.cmu.edu/courses/15110q-f19`

You are advised to create on your computer/account a folder named `110-hw`. For each new homework, you should create a new sub-folder named `01`, `02`, etc. where you can put the files related to the homework. In this way you work will be nicely organized and information and files will be easily accessible.
You can also create an equivalent struture for the *laboratories*, where in this case the root folder should be named `110-lab`.

When you are ready with the homework and want to submit your solutions, you need to go in the current homework folder (e.g., `01`), *select all files you will submit* (that can include both `.pdf` files with written answers to questions and python code files, `.py`) and compress them in one single `.zip` file.[1]

According to the OS you are using, you migh have different options for making the zip file. For instance, on Windows, after selection of the files, you should right-click and select `Send to: Compressed folder`, while on macOS, you can select `Compress` on the menu appearing from the right-click.

The compression action will produce a zip file containing the files to be handed in for the assignment. The file should be named `hwXX-handin.zip` (e.g., for this homework, the name of the file should be `hw05-handin.zip`). Then, open Autolab, find the page for this assignment, and submit your `hw05-handin.zip` file via the "Submit" link.

☛ The number of submissions is limited to 5. The last submission is the one that will be graded.

Style

Part of your grade on assignments are style points, that can be lost if your code is too disorganized, unreadable or unnecessarily complicated. To avoid loosing style points, please follow the guidelines at `https://web2.qatar.cmu.edu/cs/15110/resources/style.pdf`.

[1]The (single) zip file is needed, even when the files handed for the assignment consists of one individual file.

1 Complete the code

Problem 1.1: (13 points)

Complete the code of the function sum_odd_numbers(n) that takes as input an integer, n, and returns the sum of all the odd numbers from 0 up to (included) n.

For instance, if n is 11, then the function returns 36, which is the sum of 1 + 3 + 5 + 7 + 9 + 11.

```
def sum_odd_numbers(n):
    odds_sum = 0
    for i in range(  ,   ,  ):
        odds_sum
    return odds_sum
```

Problem 1.2: (16 points)

Complete the code of the function sum_and_extend(L) that takes as input a list of numbers, L, and keeps extending the list by adding each time a new element new_val equal to the sum of the last two items in the list.

If the value of new_val exceeds 100, the item is not added to the list and the function ends, returning a tuple of two elements. The first element is the value of new_val, the second element is number of items in the list.

If the number of items in the list L is less than 2, the function returns a tuple with None and the number of items in L.

For instance, sum_and_extend([6,10]) returns (110, 4).

```
def sum_and_extend(L):
    if len(L)
        cnt = 0
        while True:
            new_val = L[-1] +
            if new_val < 100:
                L.
                cnt +
            else:
                return (new_val,    )
    else:
        return (None,    )
```

Problem 1.3: (15 points)

Complete the code of the function `discount_products(L, discount, value, max_items)` that takes as input a list `L`, two floats, `discount` and `value`, and an integer, `max_items`.

The function applies a discount `discount` to the first `max_items` in the list whose value is higher than or equal to `value`.

Note that the function doesn't need to return anything since the list `L` passed as argument will be changed once the function is invoked.

For instance if `x = [10, 4, 5, 6, 2, 7]`, invoking `discount_products(x, 0.5, 6, 2)` results in `x` becoming: `[5.0, 4, 5, 3.0, 2, 7]`.

```
def discount_products(L, discount , value , max_items):
    discounted_items = 0
    index = 0
    while discounted_items
        if L[    ] >= value:
            L[    ] *= discount
            discounted_items +=
        index +=
```

2 Use `for` loops, lists, strings

Problem 2.1: (33 points)

Implement the function `get_average_price(car_infos, ref_year)` that takes as inputs a list of information about cars, `car_infos`, and an integer, `ref_year`. The list `car_infos` is a list of tuples of the form (`model`, `year`, `price`), where `model` is a string (e.g., ``Toyota Corolla''), `year` is an integer, and `price` is a float.

The function returns a tuple, where the first element is the average price of the cars whose year is greater or equal to `ref_year` (note: the average of n numbers $x_1, x_2, \ldots, x_n$ is computed as $\frac{1}{n}\sum_{i=1}^{n} x_i$).

The second element of the tuple is a list of strings with all the different car models that have been used for computing the average (i.e., the cars whose year $\geq$ `ref_year`) without duplicates. Let's call this list of strings `models`. If two cars of the same model (e.g., ``Toyota Corolla'') have been used in the average, that model should be included only once in the list `models`, in correspondence of the first time that model was encountered in the list `car_infos` starting from position index 0.

For instance, if `car_infos` is the following:

```
car_infos = [("Corolla", 2012, 100), ("Jeep", 2015, 200), ("Ferrari", 2016, 220),
             ("Corolla", 2013, 110), ("Jeep", 2012, 85), ("Cherokee", 2013, 180),
             ("Ferrari", 2015, 200)]
```

`get_average_price(car_infos, 2013)` returns: `(182.0, ['Jeep', 'Ferrari', 'Corolla', 'Cherokee'])`

`get_average_price(car_infos, 2016)` returns: `(220.0, ['Ferrari'])` `get_average_price(car_infos, 2017)` returns: `(0.0, [])`

Note that if two model strings in `car_infos` are different for any reason (e.g., `'Ferrari'` vs. `'ferrari'`) they are to be considered as different (yes, it is not a smart thing to do, but we fix this in the next question!).

Expected number of code lines: 12

Problem 2.2: (33 points)

Implement the function `compare_strings(L)` that fixes the not-that-smart string comparison done in the previous question. More precisely, the function, takes as input a list of strings, `L`, and returns a new list of strings where all the words in the strings have been capitalized (all characters lower cased except the first) and separated by a single white space, all extra white spaces have been removed, and all duplicate strings have been removed.

Let's illustrate the behavior of the function through an example. Let `L` be the following:

```
L = ["toyota␣Corolla", "fca␣␣␣jeep", "Ferrari␣␣␣testarossa", "Toyota␣␣␣corolla",
      "FCA␣␣␣␣jeep", "ferrari␣Testarossa"]
```

then, the function must return:

```
["Toyota␣Corolla", "Fca␣Jeep", "Ferrari␣Testarossa"]
```

You may notice that, for instance, `'toyota␣Corolla'` and `'Toyota␣␣␣corolla'` are seen as the same string, such that only one entry of it (no duplicates) is included in the final list of strings. They are the same string once all extra white spaces are removed, all characters are lower cased, and the first character of each word composing the string is capitalized (*Hint*: you need precisely to transform each string in this way in order to make the comparisons and produce the returned list of strings).

Hint: use the string methods `split()` (to create a list of strings out of a string of words) and then `join()` (to create a string of words out of a list of strings). A description of both functions can be found in the PDF of lecture of Sep 24.

Expected number of code lines: 10