15-110 Fall 2019
Hw 03

Out: Saturday 14th September, 2019 at 1:30 AST
Due: Thursday 19th September, 2019 at 17:30 AST

Introduction

In this homework you will practice with strings.

The total number of points available from the questions is 100 + 10, where 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 the file `hw03.py` with the python functions answering the questions in Section 2. In the handout you have found a file `hw03.py` with the functions already defined but with an empty body. 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 `hw03-handin.zip`). Then, open Autolab, find the page for this assignment, and submit your `hw03-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.

HINT: For all the following questions, you can find the description of the methods and of the operators to use in the lecture slides (and in the book). It might be also helpful to checkout the methods available for strings on the Python docs: https://docs.python.org/3.7/library/stdtypes.html#string-methods. The docs are your friends, learn how to use them!

1 Extracting parts of a string

Problem 1.1: (18 points)

Implement the function `evens(s)` that takes a string `s` as input and returns another string composed only by the characters at even positions. For example, `evens('abcde')` should return "ace".

However, if the middle character of `s` is the same as the first and last characters of `s`, the function shall return a string of the same length as `s` but with all characters being the same as the middle character of `s`. For example, `evens('GATTGGAAHTAG')` should return `GGGGGGGGGGGG`.

Note that the *middle point* of a sequence of n elements depends on whether n is even or odd. For instance, if $n = 11$, the middle point is the element at position 5 (counting from 0). Instead, if $n = 10$ (even) the notion of middle point is not precisely defined, because it should be "between" the elements at positions 4 and 5. In these cases, we will consider the middle point being the element at position $\frac{n}{2}$. For examples, if `s` is the string `'0123456789'`, which consists of 10 characters, the middle point character is `'5'`. If `s` is the string `'1234567890*'`, the middle point character is `'6'`.

Problem 1.2: (18 points)

Data records are usually organized in *fields* of fixed width in terms of characters. For instance, a monthly record of temperature measurements can be organized in 4 fields, one field per week, where each field reports measured data during the week.

Implement the function `extract_data(s, n)` that extracts from the input string `s` the characters at every `n` positions and concatenate them in a new string. If the new string contains white spaces, all the white spaces need to be removed. The resulting string must be printed out and be returned by the function.

For example, if the `s` is the string `'123␣456␣789␣012'`, and `n` is 4, the function must return (and print out) the string `'1470'`. If `s` is `'123␣456␣78␣␣␣97'`, the returned string is `'147'`.

2 Check the properties of a string

Problem 2.1: (18 points)

A string `s` is said to be *palindrome* if the reverse of `s` *reads* the same as `s`. For example, this is the case of `'radar'`, `'civic'`, `'abba'`, `'10801'`. Implement the function `palindrome(s)` that takes as input a string `s` and returns `True` if `s` is palindrome, `False` otherwise.

Note that the string `'Abba'` is palindrome in spite of the fact that the two strings `'Abba'` and `'abbA'` aren't exactly the same from a python point of view.

3 Count the occurrences of a substring

Problem 3.1: (26 points)

Implement the function `count_occurrences(s, ss)` that takes as inputs two strings, `s` and `ss`. The purpose of the function is to check whether the substring `ss` is contained in the string `s`.

As an example, let's assume that `s` is the string `'Mountains, sea, lakes. Sea with green waters.'` and `ss` is the string `'sea'`.

- If `ss` is a substring of `s`, the function returns a multi-line output string. In the case of the example case above, the output would be as follows:

```
''The substring 'sea' has been found 2 times.
The first occurrence is at position 11.
The string content following the last occurrence of 'sea' is ' with
    green waters.''
```

  Note that the output must be a *multi-line string* (more precisely, a string that would print out over three lines).

  Note also that the substring can be found in *any lower/upper case combination*. E.g., both `'Sea'` and `'sea'` count.

- If `ss` is not a substring of `s`, the function returns the following substring: `''The string 'sea'is not part of the input string''`, where of course `'sea'` should be replaced by the value of the given string `ss`.

4 Construct valid strings

Problem 4.1: (30 points)

Users like to give their files all sorts of creative names. Unfortunately, computer systems can be limited in what they understand as a file name. Suppose that a system only allows file names that are composed of two parts, a *name* and an *extension* (`filename.ext`), and that follow the rules below for defining the name and the extension:

1. There must be one and only one "dot" (.).
2. The dot must separate the name and the extension.
3. The extension must be formed by exactly three characters.
4. There shall be **no** white spaces.

5. The name must have at least one character.

6. The name cannot start with a number.

7. The name can only contain alphanumeric characters (letters and digits from 0 to 9).

Implement the function `is_valid_filename(s)` that takes a string as input and returns `True` if this string is a valid file name according to the rules above, or `False` otherwise.