CSE 101: Introduction to Computers – Section 2 (MW 2:30-3:50 pm)

Stony Brook University

Lab #9

Spring 2017

Assignment Due: April 14, 2017 by 11:59 pm

Assignment Objective

This lab will give you some hands-on practice writing regular expressions to identify strings that match certain patterns.

Getting Started

For the labs and homework assignments in this course you will be writing functions that solve computational problems. To help you get started on each assignment, we will give you on Piazza a "bare bones" file with a name like `lab9.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 `lab9.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.
- At the top of the `lab9.py` file, include the following information in comments, with each item on a separate line:
  - your first and last name as they appear in Blackboard
  - your Stony Brook ID #
  - the course number (CSE 101)
  - the assignment name and number (Lab #9)
- Your functions must be named as indicated below. Submissions that have the wrong function names can't be graded by our automated grading system.
- Upload your `.py` file to Blackboard 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.

Preliminaries

For this lab you will be working with regular expressions in Python. Various functions for working with regular expressions are available in the `re` module. Fortunately, Python makes it pretty easy to see if a string matches a particular pattern.

At the top of the file we must import the `re` module:

```
import re
```

Then we can use the `search()` function to test whether a string matches a pattern. In the example below, the regular expression has been saved in a string called `pattern` for convenience:

```
phone = '123-456-7890'
pattern = r'^\d{3}-\d{3}-\d{4}$'
if re.search(pattern, phone):
    print('The string matches the pattern.')
else:
    print('The string does not match the pattern.')
```

The `r` that precedes the pattern string is not a typo. Rather, the `r` indicates that the string is a "raw" string. In a raw string, as opposed to a "normal" string, any backslash character is interpreted as simply a backslash, as opposed to defining an escape sequence like `\n` or `\t`. Make sure you use raw strings in your Python code when defining regular expressions.

The `^` and `$` at the beginning and end of the regular expression indicate that the entire string must match the regular expression, and not just part of the string. Make sure you include these symbols in your regular expressions too!

Part I: Categorizing Product Labels (3 points)

Write a function `category_counts()` that takes a single argument consisting of a list of product labels given as strings and returns a list of four integer counts representing how many products fall into each of four categories:

- **Category 1**: the labels starts with the letters PQ, followed by a 5-digit number (00000 through 99999), and ending with either an uppercase or lowercase letter X
- **Category 2**: the labels starts with the letters RFT, followed by any combination of 7 uppercase letters and digits, and always ending with one additional digit
- **Category 3**: the labels starts with the letter A, followed by 7-10 uppercase letters, followed by 3-7 digits
- **Category 4:** the label does not match any of the above patterns

Examples:

Function Call	Return Value
`category_counts(['PQ92819x', 'TKSNC82J4A', 'AJEIZMFWEL271827', 'RFTJ9WZMS28', 'CIW92KJSD'])`	`[1, 1, 1, 2]`
`category_counts(['PQ38400X', 'pq38323X', 'RFT8291912', 'AJSKWJS83928'])`	`[1, 0, 0, 3]`
`category_counts(['PQ99999XX', 'RFT00000009', 'RRFTKMZSDDS4', 'CSE101WOWZA', 'RFTK9DMG828', 'BAWDKMJDW8291242', 'PQ12341X'])`	`[1, 2, 0, 4]`

Part II: Calculating Enrollments (3 points)

Write a function `enrollments()` that takes a list of strings representing course IDs (e.g., "CSE 220") and returns a count of how many strings define valid courses. Courses *should* be expressed as a three-letter subject in mixed uppercase/lowercase letters, followed by an optional space, followed by a three-digit number. So, ideally, each course is given in a form similar to `"CSE 101"` or `"Phy 132"`. One obstacle to solving this problem is that sometimes extra characters – specifically, combinations of periods, dashes and commas – are inadvertently inserted in between letters. In such cases we still consider the course IDs to be valid.

Examples:

Function Call	Return Value
`enrollments([CSE 101', 'AMS 310', 'PHY 132', 'Wrt 102'])`	`4`
`enrollments(['CSE114', 'ECO330', 'CHNN 101', 'Ams 261', 'MAT  200', 'WRT101', 'frn1012', 'che 299'])`	`5`
`enrollments(['C-S-E 114', 'C.S..E215', 'AMS-211', 'B,,,I.-O 255', '-ECO 102'])`	`3`

Part III: Identifying Credit Cards (3 points)

Write a function `credit_card()` that takes an *integer* representing a credit card number and returns the brand of the credit card. The returned brand is given in all uppercase letters. If the card does not match any brand, the function returns `None`. The table below gives the details about the credit card number formats for several companies:

Brand	# of Digits	Allowable Starting Digits
ALPHA	16	4026, 417500, 4405
BETA	16	500-549 (range of values)
GAMMA	16-19 (range of possible lengths)	62
OMEGA	14	300-305

Examples:

Function Call	Return Value
`credit_card(4175007282312321)`	`'ALPHA'`
`credit_card(5025007282312321)`	`'BETA'`
`credit_card(62250072875454563)`	`'GAMMA'`
`credit_card(30428763928172)`	`'OMEGA'`
`credit_card(417500728231987)`	`None`
`credit_card(50550072823127458)`	`None`
`credit_card(622500728754412545 63)`	`None`
`credit_card(30628763928172)`	`None`

Note that the quotation marks displayed in the examples 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 `http://blackboard.stonybrook.edu` and locate the course account for CSE 101.
2. Click on "Assignments" in the left-hand menu and find the link for this lab assignment.
3. Click on the link for this lab assignment.
4. Click the "Browse My Computer" button and locate the `.py` file you wish to submit. You should be submitting only one 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.