Assignment 4 - Regexes

CS 234

due March 3, 11:59pm

0 Introduction

This assignment comes in two parts. The first part is on paper. The second part is in code.

This assignment is to be completed individually, but feel free to collaborate according to the course's external collaboration policy (which can be found in the syllabus).

The deliverables consist of one `.pdf` file and one `.py` file. The deliverables should be submitted electronically to by the deadline. Put any attribution text in the `.pdf` file. You may also consider adding an experience report to the `.pdf` describing your experience with the assignment: how long did it take, how hard/fulfilling was it, etc.

Your `.pdf` file should be named like `FLast_cs234_aX.ext` where `F` is your first initial, `Last` is your last name, `X` is the assignment number, and `ext` is the appropriate file extension. For example, Alexandra Silva's `.pdf` file should be given the name `ASilva_cs234_assignment4.pdf`. (Alexandra Silva is researcher who works with regular expressions and automata. I've also coauthored a paper with her!)

1 Regexes on Paper

Please complete the following exercises from the textbook in your `.pdf` submission. Clearly label your responses with the exercise number.

This part is worth **92 points** with each question worth around **4 points** in expectation, except for the final 3 proof-based questions worth around **8 points** each in expectation.

- 5.2, 5.3, 5.5
- 5.6, 5.10, 5.11, 5.14, 5.17, 5.18, 5.20, 5.24
- 6.2, 6.6, 6.9
- 6.13, 6.15, 6.16
- 7.2, 7.8, 7.10 (Make sure you practice good proof-writing! Look back at the tips to make sure you follow them, and refer to definitions where appropriate!)

2 Coding with Regexes

Please complete the following tasks in your `.py` submission. This part is worth **8 points** in total. Make absolutely sure to implement the indicated function names, and do not import any Python libraries. Also, please to not have your file print when loaded as a module.

Regexes are used commonly in industrial settings because they are quite useful. Not only can regular expressions express relatively complex ideas, but because they can be compiled down to DFAs, they can query strings in $O(n)$ time, which is quite efficient! They are so useful that Python has built in support for them. You will use that Python support in this task to build functions that query strings to find particular patterns, with each function being worth **4 points** to implement.

Documentation on Python's regex library can be found here: `https://docs.python.org/3/library/re.html`. You will want to use the `findall` method, which finds all substrings matching a given regex pattern.

A regex in Python is just a string with some special syntax. You should probably prefix that regex string with `r`, like `r"hello"`—this makes the string a raw string which changes how escaping characters works. Both the link and the textbook contain information about the syntax of regexes in Python. But here a primer anyway:

- `a` just matches the symbol `a`. Similar syntax allows for other direct character matches. Don't forget that space is a symbol!
- `[abc]` matches the set of characters given by $a + b + c$. Similar syntax allows for other sets of characters.
- `[A-Z]` matches the set of capital letters. Similar syntax allows for other ranges of characters. To refer to the dash symbol specifically, you will need to write `\-`; otherwise it would not be clear if you wanted a range of characters or just the dash symbol.
- `[^abc]` matches the *complement* of the set of characters given by $a+b+c$. Similar syntax complements other character sets.
- `.` matches any symbol (except newline). Actual period symbols need to be escaped using a backslash, like `\.`
- `\w` matches any alphanumeric or underscore symbol. Similarly, `\d` gives digits and `\s` gives whitespace.
- `\W` matches symbols that `\w` does not. Similarly, `\D` gives non-digits and `\S` gives non-whitespace.
- `\(` represents the symbol `(`, and `\)` represents the symbol `)`.
- `|` is the equivalent of the normal regex $+$.

- `*` is the normal Kleene star, representing any number of repetitions of what it directly follows.
- `R{n}` is the equivalent of R^n.
- `R?` is the equivalent of $1 + R$.
- `R+` is the equivalent of RR^*, matching 1 or more occurences of R, but not 0 occurences
- Regexes are concatenated by putting them next to each other, with no special symbol.
- `(?:R)` creates a non-capturing group around a regex `R`. Parentheses act a little confusingly in Python regexes. While they do group regex operations just like normal, they also simultaneously create what are called "capture groups" which makes `findall` behave differently. A better move for the purposes of this assignment is to use non-capturing groups whenever you would want parentheses.

With that out of the way, here is what you must implement.

1. Implement a function `findPhones` taking in text as a string and returning a list of strings of all (US-formatted) phone numbers in the text. These phone numbers can occur anywhere in the string, which corresponds with what `findall` finds—you should just return the result of `findall` used with the appropriate pattern.

   For this problem, a phone number is 10 digits in groups of 3, 3, and 4. Each group may be separated by a dash, a space, a period, or simply not be separated. The first group of 3 may optionally be enclosed in parentheses. Thus `(123).4567890` is a valid phone number, but `1-234567890` is not. (Technically phone numbers also could have country codes and other forms, but ignore all that for this question. Only implement according to the specification provided.)

2. Implement a function `findEmails` taking in text as a string and returning a list of strings of all email addresses in the text. These email addresses can occur anywhere in the string, which corresponds with what `findall` finds—you should just return the result of `findall` used with the appropriate pattern.

   For this problem, an email address can contain the symbols a-z, A-Z, 0-9, dashes, underscores, periods, and exactly one @. Crucially, every period and @ must be adjacent to symbols that are not periods or @, and they cannot be at the start or end of the email. For example, `username@domain.co.uk` is a valid email, but `@hello` is not. (Technically, valid email addresses are more complicated than this, but ignore all that for this question. Only implement according to the specification provided.)