DS 100: Principles and Techniques of Data Science **Date: February 16, 2018**

Discussion #4

Name:

Regular Expressions

Here's a complete list of metacharacters:

`. ^ $ * + ? { } [ ] \ | ( )`

Some reminders on what each can do (this is not exhaustive):

"^" matches the position at the beginning of string (unless used for negation "[^]")

"$" matches the position at the end of string character.

"?" match preceding literal or sub-expression 0 or 1 times. When following "+" or "*" results in non-greedy matching.

"+" match preceding literal or sub-expression *one* or more times.

"*" match preceding literal or sub-expression *zero* or more times

"." match any character except new line.

"[ ]" match any one of the characters inside, accepts a range, e.g., "[a-c]".

"( )" used to create a sub-expression

"\d" match any *digit* character. "\D" is the complement.

"\w" match any *word* character (letters, digits, underscore). "\W" is the complement.

"\s" match any *whitespace* character including tabs and newlines. \S is the complement.

"\b" match boundary between words

Some useful `re` package functions:

`re.split(pattern, string)` split the `string` at substrings that match the `pattern`. Returns a list.

`re.sub(pattern, replace, string)` apply the `pattern` to `string` replacing matching substrings with `replace`. Returns a string.

Reading Regex

1. Given the text,

```
<record> Joseph Gonzalez <jegonzal@berkeley.edu> Faculty </record>
<record> Jake Soloff <jake_soloff@berkeley.edu> TA </record>
```

   Which of the following matches exactly to the email addresses (including angle brackets)?

   (a) `<.*@.*>`
   (b) `<\w*@.*?>`
   (c) `<[^>]*>`

Solution: (b) Note `<\w*@` matches as many word characters as possible between a `<` and an `@` symbol. Similarly `@.*?>` matches as few characters between `@` and `>`, thereby closing the right tag as soon as possible after the `@`.

2. Which strings contain a match for the following regular expression, `abc?$`

   (a) `Know your abcs`
   (b) `Did you say abc?`
   (c) `Hi ab`

Solution: (c) Recall that `c?` matches on *at most one* occurrence of the letter `c`, and `$` marks the end of the string.

3. For each pattern specify the starting and ending position of the first match in the string.

	`abcdefg`	`abcs!`	`ab abc`	`abc, 123`
`abc*`	**1–3**			
`[^\s]+`				
`ab.*c`				
`[a-z1,9]+`				

Solution:

	abcdefg	abcs!	ab abc	abc, 123
abc*	**1–3**	1–3	1–2	1–3
[^\s]+	1–7	1–5	1–2	1–4
ab.*c	1–3	1–3	1–6	1–3
[a-z1,9]+	1–7	1-4	1–2	1–4

Writing Regex

4.

(a) Write a regular expression that matches a string that contains only lowercase letters and numbers (including empty string).

Solution:

```
regx = '^[a-z0-9]*$'
```

(b) Given **`text = "123 Fake Street"`**, use methods in `RE` module to abbreviate **`"Street"`** as **`"St."`**. The result should look like **`"123 Fake St."`**.

Solution:

```
re.sub('Street',  'St.', text)
```

(c) Given **`text2 = "October 10, November 11, December 12, January 1"`**, use methods in `RE` module to extract all the numbers in the string. The result should look like **`["10", "11", "12", "1"]`**.

Solution:

```
re.findall(r'\d+',  text2)
```