Regular Expressions

CMPUT 396

Motivation

- We often want to match a set of strings against a pattern.
- E.g., what are the words that end in "-ing"?
- Python provides a module that implements **regular expressions**: *import re*
- Regular expressions are extremely useful and powerful tools for programming in general.
- We will learn how to use them on the decryption task.

Regular Expressions

- A regular expression is a sequence of characters that define a **search pattern**.
- The simplest search pattern is the exact string one is searching, like /woodchuck/.
- Regular expression **operators** provide ways to concisely specify a set of strings.
- For example, the set containing the three strings "Handel", "Händel", and "Haendel" can be specified by the pattern /H[ä|ae?]ndel/

RE basic operators

- The dot operator matches any char except '\n'
  - be.t matches best, belt, beet, be3t, …
- The star operator matches 0 or more repetitions of the previous character
  - be*t matches bt, bet, beet, beeet, ….
- The square brackets operator matches any one character listed within the square brackets:
  - be[ls]t matches belt, best
  - be[l-o]t matches belt, bemt, bent, beot
- The caret matches the complement of a set:
  - be[^0-9]t matches belt, best, be#t (but not be4t)
  - be[^xyz]t matches belt, be5t (but not bext, beyt, bezt)

RE anchors

- The caret anchor matches the start of the string
  - ^be matches be, bet, beat (but not abe)
- The dollar anchor matches the end of the string
  - be$ matches be, abe, cube (but not bee)
- They are often used to delimit the pattern
  - ^.*be.*$ matches all strings that contain be

RE advanced operators

- The plus operator matches 1 or more repetitions of the previous character
  - be+t matches bet, beet, beeet, …
- The question mark operator matches 0 or 1 repetitions of the previous character
  - bee?t matches bet, beet
- The | operator is an "or" operator:
  - hello|Hello matches hello, Hello
  - a+|b+ matches a, b, aa, bb, aaa, bbb, …
  - ab+|ba+ matches ab, abb, abbb, …, ba, baa, baaa, …

Regular Expression Summary

Regular Expression	Interpretation
.	match any character (*wildcard*)
[abc]	match *a* or *b* or *c*
[^abc]	match any character other than
[abc]+	match one or more occurrences
[abc]*	match zero or more occurrences
[abc]?	match zero or one occurrences
(*regex*)	create a capture group

RE module functions

- *matchObj = re.match(pattern, string)* - returns a match object if it finds *pattern* in *string* (or *None)*.
- *gList = matchObj.groups()* – returns a list of all capture groups matched.
- *newstring = re.sub(pattern, repl, string)* – like replace but applies to all occurrences of *pattern*.
- *sList = re.findall(pattern, string*) – returns a list of all substrings that match *pattern* in *string*.

Matching words against patterns

```
def checkWord(regex):
    resList = []
    wordFile = open('wordlist.txt')
    for line in wordFile:
        if re.match(regex,line[:-1]):
            resList.append(line[:-1])
    return resList
```

RE exercise

- Write a regular expression pattern to match:
  - all words ending in *ing*.
  - all words with *ss* anywhere in the string.
  - all words beginning and ending with the letter *a*.
  - all the four-letter words where the middle two letters are vowels.
- Write a function that can extract the host name from a URL. The host name is the part of the URL that comes after *http://* but before the next */*

Parsing a URL

```
def getHost(url):
    regex = 'http://([^/]*)/'
    g = re.match(regex, url)
    if g:
        return g.group(1)
    else:
        return None
```