COMP 204: Regular Expressions
A brief introduction

Yue Li
based on materials from Christopher J.F. Cameron and Carlos G. Oliver
Some familiar sequence pattern matching problems

- Find a substring containing only hydrophobic residues: (G, A, V, L, I, P, F, M, W), e.g., ELIFE
- Find a substring that starts with ‘AUG’, have multiple of 3 DNA letters in the middle, and ends at one of the three stop codons ”UAG”, ”UAA”, or ”UGA” (e.g., AUGACGTGCUUAG or AUGGUAUAA)
- Does a sequence contain a substring with ‘AACGAGA’ repeated 3 times but with at most 2 letters between the repeated segments (e.g., AACGAGAACGAGATAACGAGA)
- Extract ICD-9 group code ranges (e.g., Intestinal infectious diseases (001-009))

While we can use for-loop or string indexing to find patterns, there is a much more elegant way to find these patterns – regular expression.
What are regular expressions?

A **regular expression (or regex)** is a sequence of characters
▶ that helps match or find other strings or sets of strings
▶ using a specialized syntax held in a pattern

For example:
▶ `r'(.* are (.* than .*)' is a regex pattern
▶ that would match the following string:
  "Dogs are smarter than cats"
Why use regex?

Once you learn the syntax of regex
► you’ll gain a powerful time-saving tool

It’s much faster to write regex patterns
► than to write multiple:
  ► conditional statements
  ► loops
  ► lists
  ► variables

Python also makes it very easy to implement regular expressions
► using the `re` module
► API: [https://docs.python.org/3/library/re.html](https://docs.python.org/3/library/re.html)
Regex in Python and raw strings

When particular characters are used in regular expressions
▶ they take on a special meaning
▶ e.g., \r'.\' means to match any single character except a newline (i.e., '\n ')

To avoid any confusion while dealing with regular expressions
▶ in Python, we use **raw strings** for the pattern

To indicate a raw string in python
▶ prefix the pattern string with the \r character
▶ e.g., \r'regex pattern' 
▶ e.g., \r'.\.*' is different from '\.*'
Regular Expression Patterns

Except for **control characters**, all characters match themselves

- control characters: +, ?, ^, $, ( ), [ ], { }, |, \n
- meta characters that give special meaning to the regex

For example, without a control character:

- the pattern `r'o'` means match the letter ‘o’
- applying the pattern to the string ‘Tom likes noodle’
- would return ‘o’ from ‘Tom’ and two ‘o’s from ‘noodle’

With a control character:

- `r'o{2}'` means match exactly two occurrences of ‘o’
- would return ‘oo’ from ‘noodle’
Control characters

1. `r'^'` - matches the start of a string (e.g., `r'^Cat.*'` find all strings that start with ‘Cat’)

2. `r'$'` - matches the end of a string (e.g., `r'UAA$'` find all strings that end with ‘UAA’)

3. `r'. '`- matches any single character except newline

4. `r'[^...]'` - matches any single character in brackets
   - e.g., `r'[a-zA-Z]'` matches one occurrence of any ASCII character

5. `r'[^...]'` - matches any single character not in brackets
   - similar to Python’s not in this context
Control characters #2

6. `r'*'` - matches 0 or more occurrences of preceding expression (e.g., `r'[ATCG]*'` matches both XXXX and AAAAA)

7. `r'+'` - matches 1 or more occurrence of preceding expression (e.g., `r'[ATCG]+'` matches AAAAA but not XXXX)

8. `r'? '` - matches 0 or 1 occurrence of preceding expression

9. `r'{n}'` - matches exactly $n$ occurrences of the preceding expression
   - `r'o{2}'` matches ‘oo’ in ‘noodle’

10. `r'a|b'` - matches either ‘a’ or ‘b’
Regex character classes

Character classes (or sets)

- define patterns that match only one out of several characters

For example:

1. `r'\[Pp\]ython'` - match ‘Python’ or ‘python’

2. `r'\[aeiou\]'` - match any one lowercase vowel

3. `r'\[0-9\]'` - match any digit (same as `r'\[0123456789\]'`)

4. `r'\[^0-9\]'` - match anything other than a digit

5. `r'\[a-zA-Z0-9_\]'` - match any ASCII letter or digit
   - which is the same as `r'\w'`
The `search()` function from `re` Python library

- function searches for first occurrence of pattern anywhere within string

- syntax:
  ```python
  re.search(pattern, string)
  ```

- parameters:
  1. `pattern` - regular expression to be matched
  2. `string` - string to be searched
Regex in Python: `search()`

**The `search()` function**

- returns a match object on success
  - `None` on failure

- to get the matching string
  1. `group(num=0)` - method returns entire match
     - or specific subgroup `num`
  2. `groups()` - returns all matching subgroups in a tuple
     - empty if there weren’t any
import re

line = "Dogs are smarter than cats"

searchObj = re.search( r'(.* ) are \(.*?\) .*', line)

if searchObj:
    print("searchObj.group():", searchObj.group())
    print("searchObj.group(1):", searchObj.group(1))
    print("searchObj.group(2):", searchObj.group(2))
else:
    print("No match!!")

# searchObj.group() : Dogs are smarter than cats
# searchObj.group(1) : Dogs
# searchObj.group(2) : smarter
**Regex search() example: extract phone area code**

**phone_book.txt:**

1. Mike (514) 123-4567
2. Maria (604) 323-4568
3. Linda (617) 812-1234
4. Tom (216) 451-5789

```python
import re

f = open("phone_book.txt", 'r')
for line in f:
    # extract user name and their area code
    m = re.search(r'^\w+\t\(\d+\)', line)
    print(f"User name: {m.group(1)}; Area code: {m.group(2)}")

f.close()
```

#User name: Mike; Area code: (514)
#User name: Maria; Area code: (604)
#User name: Linda; Area code: (617)
#User name: Tom; Area code: (216)
FASTA example revisit

```python
def getSeqNames(filename):
    f = open(filename, 'r')
    for line in f:
        values = line.split()
        if values[0][0] == '>':
            print(values[0][1:])
    f.close()
```
```python
def getSeqNames_regex(filename):
    f = open(filename, 'r')
    for line in f:
        mymatch = re.search(r'>(\w+)', line)
        if mymatch:
            print(mymatch.group(1))
    f.close()
```
Regex search(): FASTA example revisit

```python
filename = " /Users/yueli/Lectures/20/seq.fa"

print("getSeqNames:")
getSeqNames(filename)
    #getSeqNames:
    #Human
    #Chimp
    #Mouse

print("getSeqNames_regex:")
getSeqNames_regex(filename)
    #getSeqNames_regex:
    #Human
    #Chimp
    #Mouse
```
import re
f = open("icd9_info.txt", 'r')

for line in f:
    m = re.search(r'\((\[V|E]\d+/\-[V|E]\d*)\)$', line.rstrip())
    if m:
        print(m.group(1))

f.close()
Search and Replace

The `sub()` function

- one of the most important `re` methods
- replaces all occurrences of the pattern in string with `repl`

- syntax:
  ```python
  re.sub(pattern, repl, string, max=0)
  ```

- parameters:
  1. `repl` - string to replace pattern
  2. `max` - replace all occurrences unless set

- returns a modified string
import re

phone = "2004-959-559 # This is a Phone Number"

# Delete Python-style comments
num = re.sub(r'#.*$', '', phone)
print("Phone Num : ", num)
# prints: Phone Num : 2004-959-559

# Remove anything other than digits
num = re.sub(r'[^0-9]', '', phone)
print("Phone Num : ", num)
# prints: Phone Num : 2004959559
Closing comments

We’ve only covered the basics of **regular expressions**

- there is A LOT more to regex
- for more information:
  [https://docs.python.org/3/howto/regex.html](https://docs.python.org/3/howto/regex.html)

Regular expressions are not only limited to Python

- **Perl**: a popular scripting language because of its regex functionality
- **grep**: a Bash command line tool for quick search among files
- **awk**: Bash command line tools efficient for one liner code
- Many more