Lect 5 – Files and RegEx

Rob Capra

INLS 490-172

Text Files

- Collections of characters
- Can be edited with a text editor
- Are typically line-oriented

Text Files

```
First Name, Last Name, Position, Team, Completions, Attempts, Yards, TDs Ints, Comp%, Rating
```

```
Colt McCoy QB CLE  135 222 1576    6   9   60.8%   74.5
Josh Freeman QB TB 291 474 3451    25  6   61.4%   95.9
Michael Vick QB PHI    233 372 3018    21  6   62.6%   100.2
Matt Schaub QB HOU 365 574 4370    24  12  63.6%   92.0
Philip Rivers QB SD    357 541 4710    30  13  66.0%   101.8
Matt Hasselbeck QB SEA 266 444 3001    12  17  59.9%   73.2
Jimmy Clausen QB CAR   157 299 1558    3   9   52.5%   58.4
Joe Flacco QB BAL  306 489 3622    25  10  62.6%   93.6
Kyle Orton QB DEN  293 498 3653    20  9   58.8%   87.5
Jason Campbell QB OAK  194 329 2387    13  8   59.0%   84.5
Peyton Manning QB IND  450 679 4700    33  17  66.3%   91.9
Drew Brees QB NO   448 658 4620    33  22  68.1%   90.9
Matt Ryan QB ATL   357 571 3705    28  9   62.5%   91.0
Matt Cassel QB KC  262 450 3116    27  7   58.2%   93.0
Mark Sanchez QB NYJ    278 507 3291    17  13  54.8%   75.3
Brett Favre QB MIN 217 358 2509    11  19  60.6%   69.9
David Garrard QB JAC   236 366 2734    23  15  64.5%   90.8
Eli Manning QB NYG 339 539 4002    31  25  62.9%   85.3
Carson Palmer QB CIN   362 586 3970    26  20  61.8%   82.4
Alex Smith QB SF   204 342 2370    14  10  59.6%   82.1
Chad Henne QB MIA  301 490 3301    15  19  61.4%   75.4
Tony Romo QB DAL   148 213 1605    11  7   69.5%   94.9
Jay Cutler QB CHI  261 432 3274    23  16  60.4%   86.3
Jon Kitna QB DAL   209 318 2365    16  12  65.7%   88.9
Tom Brady QB NE    324 492 3900    36  4   65.9%   111.0
Ben Roethlisberger QB PIT  240 389 3200    17  5   61.7%   97.0
```

Open/Close

- Open a file before using it
- Close the file when you are done

Open

- Can open a file for
  - Reading
  - writing

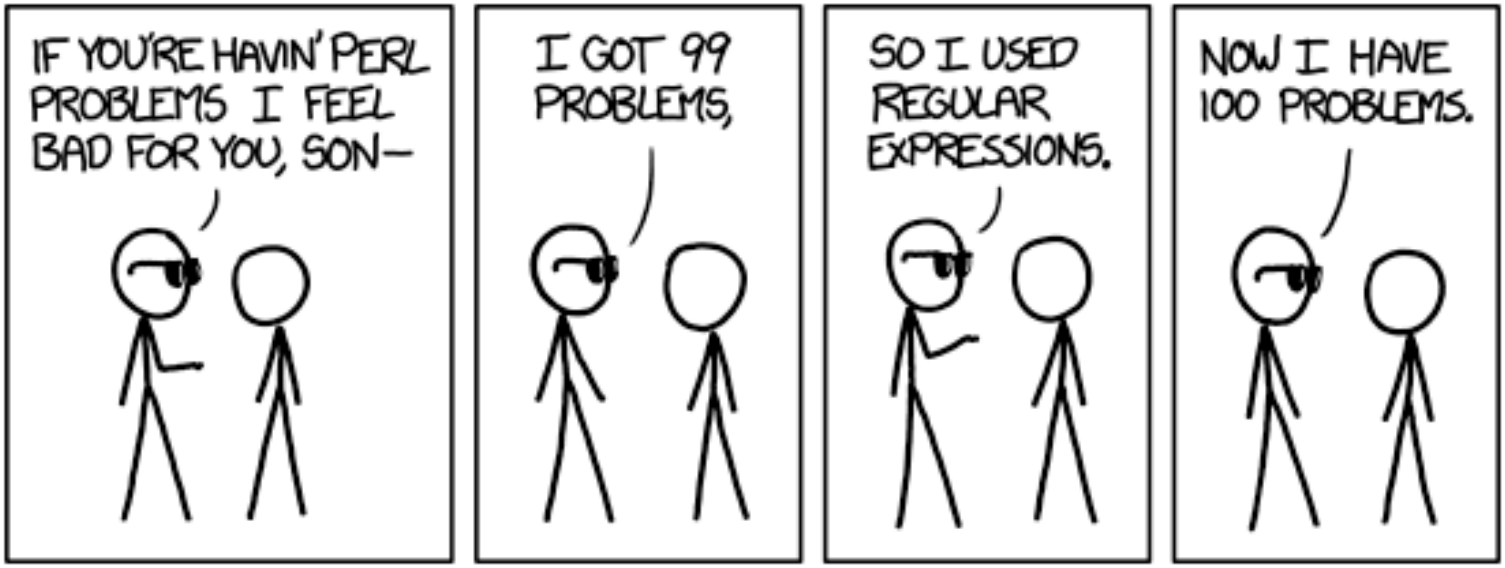
Processing a file

```
qbfile = open("qbdata.txt","r")

for aline in qbfile:
    values = aline.split()
    print('QB ', values[0], values[1],
        'had a rating of ', values[10] )

qbfile.close()
```

Regular Expressions



Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.
– Jaime Zawinski (JZW)

RegEx

- Pattern matching
- Match lines that contain a pattern of chars
- Example:
  - [01][0-9]-[0123][0-9]-[12][09][0-9][0-9]

  - 12-25-1992
  - 07-04-1776
  - 15-19-2012
  - 00-00-1000

These RegEx slides are based on slides at:
http://homes.cs.washington.edu/~ruzzo/courses/gs559/09wi/lectures/11b-regexp.pdf

RegEx Matching Rules

- Search proceeds from start to end
- Search stops at first match found
- All of the pattern must be matched
- But not all of the string

These RegEx slides are based on material from:
http://homes.cs.washington.edu/~ruzzo/courses/gs559/09wi/lectures/11b-regexp.pdf
https://developers.google.com/edu/python/regular-expressions

RegEx in Python

- Uses the Python “re” module
- General form:

```
match = re.search(pattern, string)
```

RegEx in Python

- Example:

```
import re
astring = "uncle"
match = re.search(r'unc', astring)
if match:
    print 'found = ', match.group()
else:
    print 'not found'
```

Pattern Specification Basics

- Letters and numbers match themselves
- Case sensitive
- Punctuation usually has a special meaning

One Character Matches

Pattern	What it matches
a, X, 9	Regular chars match themselves exactly
. ^ $ * + ? { [ ] \ | ()	Meta-chars that do not match themselves, but instead have special meaning
.	any single char except a newline
\	Inhibits specialness, so \. matches a dot
\w	a single “word” char: [a-zA-Z0-9_]
\t, \n, \r	Tab, newline, return
\d	Decimal digit [0-9]
^, $	Start, end of the string
\s	whitespace char: [\n\r\t\f]
\S	Any non-whitespace char

One Char Examples

```
import re
a = "uncle"
b = "March 19, 1995"
match = re.search(r'nc',a)
print match.group()
match = re.search(r'19',b)
print match.group()
match = re.search(r'19\d\d',b)
print match.group()
```

One Char Examples

```
import re
b = "March 19, 1995"
c = "on March 9, 1995"
match = re.search(r'19',c)
print match.group()
match = re.search(r'1?9',c)
print match.group()
match = re.search(r'^March',b)
print match.group()
match = re.search(r'^March',c)
if match:
    print match.group()
else:
    print "not found"
```

Repetition

Pattern	What it matches
+	1 or more occurrences of the pattern to its left
*	0 or more occurrences of the pattern to its left
?	Match 0 or 1 occurrences of the patter to its left

Leftmost & Largest

First, find the leftmost match for the pattern, then try to use as much of the string as possible (“greedy”)

9+	one or more 9s
9*	zero or more 9s
9?	zero or one 9s

Repetition Examples

```
import re
a = "199987659955"
match = re.search(r'9+',a)
print match.group()
match = re.search(r'19*',a)
print match.group()
match = re.search(r'9*',a)
print match.group()
match = re.search(r'9+.*9+',a)
print match.group()
```

Repetition Examples

```
import re
match = re.search(r'pi+', 'piiig')
print match.group()
match = re.search(r'i+', 'piigiiii')
print match.group()
match = re.search(r'\d\s*\d\s*\d', 'xx1 2   3xx')
print match.group()
match = re.search(r'\d\s*\d\s*\d', 'xx12  3xx')
print match.group()
match = re.search(r'\d\s*\d\s*\d', 'xx123xx')
print match.group()
```

Matching Alternatives

- Square brackets – any listed char can match
  - `[ab]` means either `a` or `b` can match
  - `[a-d]` matches `a` or `b` or `c` or `d`
- Use caret for negation
  - `[^a-d]` matches any char except `a`, `b`, `c`, or `d`

Matching Alternatives

- Square brackets – any listed char can match
  - `[ab]` means either `a` or `b` can match
  - `[a-d]` matches `a` or `b` or `c` or `d`
- Use caret for negation
  - `[^a-d]` matches any char except `a`, `b`, `c`, or `d`

Alternatives Practice – Part nubmer

```
import re
# A or B, followed by 3 digits
# dash, then four digits
# dash, then 3 letters or numbers
a = "A765-2781-ZFQ"  #accept
b = "B923-5743-HP3"  #accept
y = "Z843-1234-YUP"  #reject
z = "A765-8201ZFQ"   #reject
match = re.search(r'',a)
print match.group()
```

Group Extraction

- Parenthesis can be used to group parts
- These parts are then available in the group()
- Groups are referenced 1, 2, 3... left to right

```
import re
a = "A765-2781-ZFQ"
match =
    re.search(r'([AB])([0-9]+)-([0-9]+)-([A-Z0-9]+)',a)

print match.group()
print match.group(1)
print match.group(2)
print match.group(3)
print match.group(4)
```

Alternatives with more than one char

```
import re
a = "crate"
b = "state"
match = re.search(r'(cr|st)ate',a)
print match.group()
match = re.search(r'(cr|st)ate',b)
print match.group()
```

Regex Golf

Is there an algorithm for regex golf?

(set cover problem is NP-hard, so use an approximation approach)
http://nbviewer.ipython.org/url/norvig.com/ipython/xkcd1313.ipynb
Peter Norvig

Exercise #5

- Download this webpage: http://sils.unc.edu/people/faculty
  - Load in browser, right-click, Save as... faculty.html
- Open faculty.html for reading in Python
- For each line of the file, see if there is an email address on the line (use regex)
- If so, save the address in a list
- When done, close file and print list of email addresses found
- Submit your Python code to Sakai as youronyen_ex4.py