Regular Expressions

Using string methods and regular expressions to work with textual data

Sean Kang

Goals For This Lecture

Working With Text Data

- Canonicalizing text data.
- Extracting data from text.
  - Using **split.**
  - Using **regular expressions.**

String Canonicalization

Goal 1: Joining Tables with Mismatched Labels

	County	State
0	De Witt County	IL
1	Lac qui Parle County	MN
2	Lewis and Clark County	MT
3	St John the Baptist Parish	LA

	County	Population
0	DeWitt	16798
1	Lac Qui Parle	8067
2	Lewis & Clark	55716
3	St. John the Baptist	43044

join

???

A Joining Problem

	County	State
0	De Witt County	IL
1	Lac qui Parle County	MN
2	Lewis and Clark County	MT
3	St John the Baptist Parish	LA

	County	Population
0	DeWitt	16798
1	Lac Qui Parle	8067
2	Lewis & Clark	55716
3	St. John the Baptist	43044

join

???

To join our tables we'll need to **canonicalize** the county names.
- Canonicalize: Convert data that has more than one possible presentation into a standard form.

Canonicalizing County Names

County

De Witt County
Lac qui Parle County
Lewis and Clark County
St John the Baptist Parish

County

DeWitt
Lac Qui Parle
Lewis & Clark
St. John the Baptist

County

dewitt
lacquiparle
lewisandclark
stjohnthebaptist

```python
def canonicalize_county(county_name):
    return (
        county_name
        .lower()                      # Lower case
        .replace(' ', '')             # remove spaces
        .replace('&', 'and')          # replace &
        .replace('.', '')             # remove dot
        .replace('county', '')        # remove county
        .replace('parish', '')        # remove parish
    )
```

```python
def canonicalize_county(county_name):
    return (
        county_name
        .lower()                      # Lower case
        .replace(' ', '')             # remove spaces
        .replace('&', 'and')          # replace &
        .replace('.', '')             # remove dot
        .replace('county', '')        # remove county
        .replace('parish', '')        # remove parish
    )
```

Canonicalization

Canonicalization:

- Replace each string with a unique representation.
- Feels very "hacky", but messy problems often have messy solutions.

Can be done slightly better but not by much →

- Code is very brittle! Requires maintenance.

Tools used:

```python
def canonicalize_county(county_name):
    return (
        county_name
        .lower()                    # Lower case
        .replace(' ', '')           # remove spaces
        .replace('&', 'and')        # replace &
        .replace('.', '')           # remove dot
        .replace('county', '')      # remove county
        .replace('parish', '')      # remove parish
    )
```

Replacement	str.replace('&', 'and')
Deletion	str.replace(' ', '')
Transformation	str.lower()

Extracting From Text Using Split

Goal 2: Extracting Date Information

Suppose we want to extract times and dates from web server logs that look like the following:

```
169.237.46.168 - - [26/Jan/2014:10:47:58 -
0800] "GET /stat141/Winter04/ HTTP/1.1" 200
2585 "http://anson.ucdavis.edu/courses/"
```

Goal 2: Extracting Date Information

Suppose we want to extract times and dates from web server logs that look like the following:

```
169.237.46.168 - - [26/Jan/2014:10:47:58 -
0800] "GET /stat141/Winter04/ HTTP/1.1" 200
2585 "http://anson.ucdavis.edu/courses/"
```

There are existing libraries that do most of the work for us, but let's try to do it from scratch.

- Will do together, just a little bit at a time.

Extracting Date Information

```
169.237.46.168 - - [26/Jan/2014:10:47:58 -
0800] "GET /stat141/Winter04/ HTTP/1.1" 200
2585 "http://anson.ucdavis.edu/courses/"
```

One possible solution:

```python
day, month, rest = line.split(' [')[1].split(']')[0].split('/')
```

```python
year, hour, minute, seconds = rest.split(' ')[0].split(':')
time_zone = rest.split(' ')[1]
```

Extracting Date Information

```
169.237.46.168 - - [26/Jan/2014:10:47:58 -
0800] "GET /stat141/Winter04/ HTTP/1.1" 200
2585 "http://anson.ucdavis.edu/courses/"
```

One possible solution:

```python
day, month, rest = line.split(' [')[1].split(']')[0].split('/')
```

```python
year, hour, minute, seconds = rest.split(' ')[0].split(':')
time_zone = rest.split(' ')[1]
```

What if webserver
changes log formats,   ⇒ *This solution breaks!! (brittle)*
or has a bug?

Regular Expression Basics

Extracting Date Information

Earlier we saw that we can hack together code that uses split to extract info:

```python
day, month, rest = line.split(' [')[1].split(']')[0].split('/')
```

```python
year, hour, minute, seconds = rest.split(' ')[0].split(':')
time_zone = rest.split(' ')[1]
```

An alternate approach is to use a so-called "regular expression":

- Implementation provided in the `re` library built into Python.
- We'll spend some time today working up to expressions like shown below.

```python
import re
pattern = r'\[(\d+)/(\w+)/(\d+):(\d+):(\d+):(\d+) (.+)\]'
day, month, year, hour, minute, second, time_zone = re.search(pattern, line).groups()
```

26/Jan/2014:10:47:58 -0800

Regular Expressions

A *formal language* is a set of strings, typically described implicitly.

- Example: "The set of all strings of length < 10 that contain 'horse'"

A *regular language* is a formal language that can be described by a *regular expression* (which we will define soon).

Example: [0-9]{3}-[0-9]{2}-[0-9]{4}

The language of SSNs is described by this regular expression.

3 of any digit, then a dash, then 2 of any digit, then a dash, then 4 of any digit.

```
text = "My social security number is 123-45-6789.";
pattern = r"[0-9]{3}-[0-9]{2}-[0-9]{4}"
re.findall(pattern, text)
```

Regex101.com (or the online tutorial regexone.com)

There are a ton of nice resources out there to experiment with regular expressions (e.g. regex101.com, regexone.com, sublime text, python, etc).

I recommend trying out regex101.com, which provides a visually appealing and easy to use platform for experimenting with regular expressions.
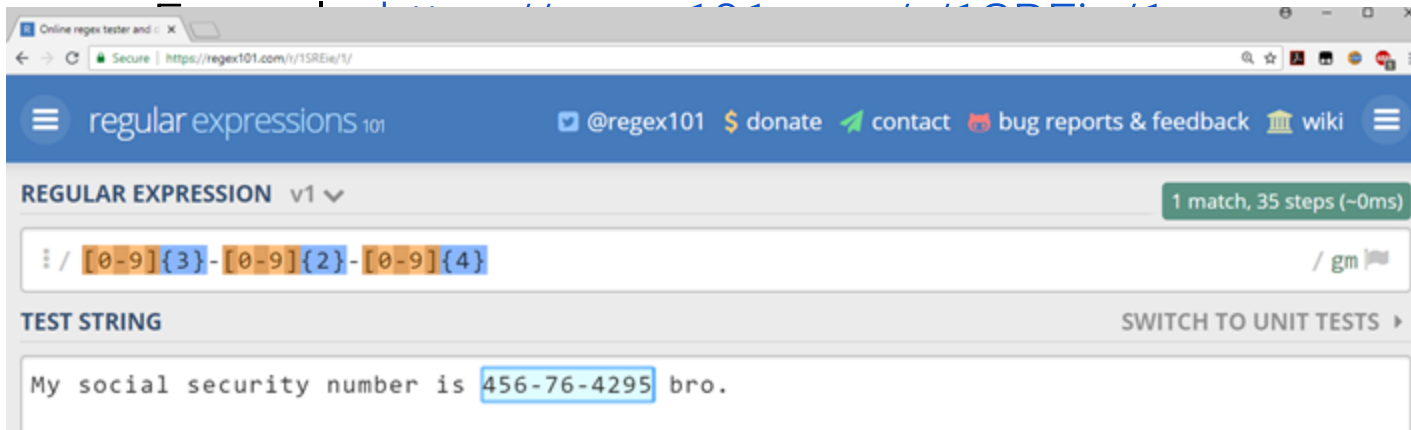
Regular Expression Syntax

The four basic operations for regular expressions.

- Can technically do anything with just these basic four (albeit tediously).

operation	order	example	matches	does not match
concatenation	3	AABAAB	AABAAB	every other string
or	4	AA|BAAB	AA BAAB	every other string
closure (zero or more)	2	AB*A	AA ABBBBBBA	AB ABABA
parenthesis	1	A(A|B)AAB	AAAAB ABAAB	every other string
		(AB)*A	A ABABABABA	AA ABBA

Regular Expression Syntax

AB*: A then zero or more copies of B: A, AB, ABB, ABBB

(AB)*: Zero or more copies of AB: ABABABAB, ABAB, AB,

Matches the empty string!

operation	order	example	matches	does not match
concatenation	3	AABAAB	AABAAB	every other string
or	4	AA|BAAB	AA BAAB	every other string
closure (zero or more)	2	AB*A	AA ABBBBBBA	AB ABABA
parenthesis	1	A(A|B)AAB	AAAAB ABAAB	every other string
		(AB)*A	A ABABABABA	AA ABBA

Give a regular expression that matches moon, moooon, etc. Your expression should match any **even** number of o's except zero (i.e. don't match mn).

Puzzle Solution

Solution to puzzle on previous slide: moo(oo)*n

Regular Expression moo(oo)*n: https://tinyurl.com/reg913m

Give a regex that matches muun, muuuun, moon, moooon, etc. Your expression should match any even number of u's or o's except zero (i.e. don't match mn).

Puzzle Solution

Solution to puzzle on previous slide: `m(uu(uu)*|oo(oo)*)n`

- Note: `m(uu(uu)*)|(oo(oo)*)n` is not correct! OR must be in parentheses!

Order of Operations in Regexes

`m(uu(uu)*|oo(oo)*)n`

- Matches starting with m and ending with n, with either of the following in the middle:
  - `uu(uu)*`
  - `oo(oo)*`

Match examples:
`muun`
`muuuun`
`moon`
`moooon`

Order of Operations in Regexes

`m(uu(uu)*|oo(oo)*)n`

- Matches starting with m and ending with n, with either of the following in the middle:
  - `uu(uu)*`
  - `oo(oo)*`

Match examples:
`muun`
`muuuun`
`moon`
`moooon`

`m(uu(uu)*)|(oo(oo)*)n`

- Matches either of the following
  - `m` followed by `uu(uu)*`
  - m followed by `oo(oo)*`

Match examples:
`muu`
`muuuu`
`oon`
`oooon`

In regexes | comes last.

Expanded Regular Expressions Syntax

Expanded Regex Syntax

operation	example	matches	does not match
any character (except newline)	`.U.U.`	`CUMULUS` `JUGULUM`	`SUCCUBUS` `TUMULTUOUS`
character class	`[A-Za-z][a-z]*`	`word` `Capitalized`	`camelCase` `4illegal`
at least one	`jo+hn`	`john` `jooooohn`	`jhn` `jjohn`
zero or one	`joh?n`	`jon` `john`	`any other string`
repeated exactly {a} times	`j[aeiou]{3}hn`	`jaoehn` `jooohn`	`jhn` `jaeiouhn`
repeated from a to b times: {a,b}	`j[ou]{1,2}hn`	`john` `juohn`	`jhn` `jooohn`

More Regular Expression Examples

regex	matches	does not match
.*SPB.* Zero or more char before and after SPB	RASPBERRY CRISPBREAD	SUBSPACE SUBSPECIES
[0-9]{3}-[0-9]{2}-[0-9]{4} SSN pattern match	231-41-5121 573-57-1821	231415121 57-3571821
[a-z]+@([a-z]+\.)+(edu|com) Email validator	horse@pizza.com horse@pizza.food.com	frank_99@yahoo.com hug@cs

Expanded Regex Puzzle: https://tinyurl.com/reg913w

operation	example	matches	does not match
any character (except newline)	`.U.U.U.`	`CUMULUS` `JUGULUM`	`SUCCUBUS` `TUMULTUOUS`
character class	`[A-Za-z][a-z]*`	`word` `Capitalized`	`camelCase` `4illegal`
at least one	`jo+hn`	`john`	`jhn`
zero or one	`joh?n`	`jon` `john`	`any other string`
repeated exactly {a} times	`j[aeiou]{3}hn`	`jaoehn` `jooohn`	`jhn` `jaeiouhn`
repeated from a to b times: {a,b}	`j[ou]{1,2}hn`	`john` `juohn`	`jhn` `jooohn`

Challenge: Give a regular expression for any lowercase string that has a repeated vowel (i.e. noon, peel, festoon, loop, etc).

Expanded Regex Puzzle Solution

Challenge: Give a regular expression for any lowercase string that has a repeated vowel (i.e. noon, peel, festoon, loop, etc): `[a-z]*(aa|ee|ii|oo|uu)[a-z]*`

More Advanced Regular Expressions Syntax

Limitations of Regular Expressions

Writing regular expressions is like writing a program.

- Need to know the syntax well.
- Can be easier to write than to read.
- Can be difficult to debug.

Regular expressions are terrible at certain types of problems. Examples:

- For parsing a hierarchical structure, such as JSON, use a parser, not a regex!
- Complex features (e.g. valid email address).
- Counting (same number of instances of a and b). (impossible)
- Complex properties (palindromes, balanced parentheses). (impossible)

Email Address Regular Expression (a probably bad idea)

The regular expression for email addresses (for the Perl programming language):

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?: \r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*")*<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*")(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*>(?:(?:\r\n)?[ \t])*)(?:,\s*(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*|(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*")*<(?:(?:\r\n)?[ \t])*(?:@(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*(?:,@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*)*:(?:(?:\r\n)?[ \t])*)?(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*")(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r\\]|\\.)*\](?:(?:\r\n)?[ \t])*))*>(?:(?:\r\n)?[ \t])*))*)?;\s*)
```

From: http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html

Even More Regular Expression Syntax

operation	example	matches	does not match
built-in character classes	`\w+` `\d+`	`fawef` `231231`	`this person` `423 people`
character class negation	`[^a-z]+`	`PEPPERS3982` `17211!↑å`	`porch`
escape character	`cow\.com`	`cow.com`	`cowscom`

Suppose you want to match one of our special characters like: . [ ]

● In these cases, you must "escape" the character using the backslash.
● You can think of the backslash as meaning "take this next character literally".

Create a regular expression that matches the red portion below.

```
169.237.46.168 - - [26/Jan/2014:10:47:58 -0800] "GET
/stat141/Winter04/ HTTP/1.1" 200 2585
"http://anson.ucdavis.edu/courses/"
```

Regular Expressions Puzzle Solution: tinyurl.com/reg913a

Create a regular expression that matches the red portion below: `\[.*\]`

```
169.237.46.168 - - [26/Jan/2014:10:47:58 -0800] "GET
/stat141/Winter04/ HTTP/1.1" 200 2585
"http://anson.ucdavis.edu/courses/"
```

See example in lab file

Even More Regular Expression Features

operation	example	matches	does not match
beginning of line	`^ark`	`ark two` `ark o ark`	`dark`
end of line	`ark$`	`dark` `ark o ark`	`ark two`
lazy version of zero or more *?	`5.*?5`	`5005` `55`	`5005005`

5.*5 would match this!

A few additional common regex features are listed above.

● Won't discuss these in class, but might come up in discussion or hw.
● There are even more out there!

The official guide is good! https://docs.python.org/3/howto/regex.html

Regular Expressions in Python
(and Regex Groups)

re.findall in Python

In Python, `re.findall(pattern, text)` will return a list of all matches.

```python
text = "My social security number is 456-76-4295 bro, or
actually maybe it's 456-67-4295.";
pattern = r"[0-9]{3}-[0-9]{2}-[0-9]{4}"
m = re.findall(pattern, text)
print(m)
```

```
['456-76-4295', '456-67-4295']
```

Regular Expression Groups

Earlier we used parentheses to specify the order of operations.

Parenthesis have another meaning:

- Every set of parentheses specifies a so-called "group".
- Regular expression matchers (e.g. `re.findall`, regex101.com) will return matches organized by groups. In Python, returned as tuples.

```python
s = """Observations: 03:04:53 - Horse awakens.
03:05:14 - Horse goes back to sleep."""
pattern = "(\d\d):(\d\d):(\d\d) - (.*)"
matches = re.findall(pattern, s)
```

```python
[('03', '04', '53', 'Horse awakens.'),
 ('03', '05', '14', 'Horse goes back to sleep.')]
```

Extracting Date Information

With a little more work, we can do something similar and extract day, month, year, hour, minutes, seconds, and time zone all in one regular expression.

- Derivation is left as an exercise for you

```python
import re
pattern = r'\[(\d+)/(\w+)/(\d+):(\d+):(\d+):(\d+) (.+)\]'
day, month, year, hour, minute, second, time_zone = re.findall(pattern, first)[0]
year, month, day, hour, minute, second, time_zone
```

You will also see code that uses `re.search` instead of `re.findall`.

```python
import re
pattern = r'\[(\d+)/(\w+)/(\d+):(\d+):(\d+):(\d+) (.+)\]'
day, month, year, hour, minute, second, time_zone = re.search(pattern, line).groups()
```

Case Studies on Police Data and Restaurant Data

Summary

Today we saw many different string manipulation tools.

- There are many many more!
- With just this basic set of tools, you can do most of what you'll need.

basic python	re	pandas
	re.findall	df.str.findall
str.replace	re.sub	df.str.replace
str.split	re.split	df.str.split
'ab' in str	re.search	df.str.contain
len(str)		df.str.len
str[1:4]		df.str[1:4]