LECTURE 5

Text as Data: Regular Expressions

Using string methods and regular expressions to work with textual data

Text is a very important format of data

- String methods
- Regular expressions
- Word encoding
    - Bag of words
    - TF-IDF
    - Word2Vec (Not covered)
    - BERT (Not covered)

Today's Roadmap

Why Work with Text?

Python String Methods

Regular Expressions (Regex) Basics

Regex Expanded

Convenient Regex

Regex in Python/Pandas (Regex groups)

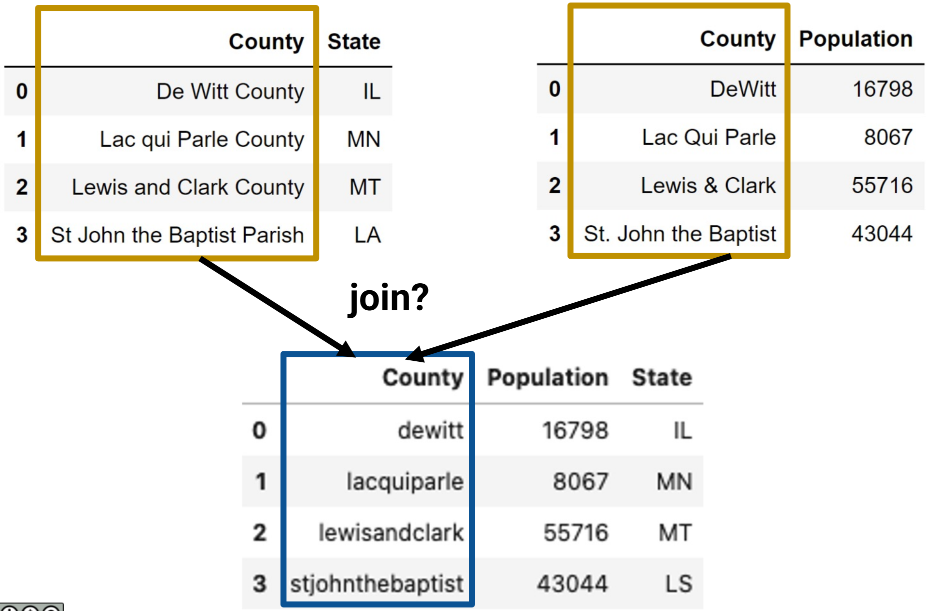
Demo on Restaurant Data

Bonus: Yes, More Regex Syntax

Why work with text? Two Main Goals

1. **Canonicalization**: Convert data that has more than one possible presentation into a standard form.

Ex Join 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

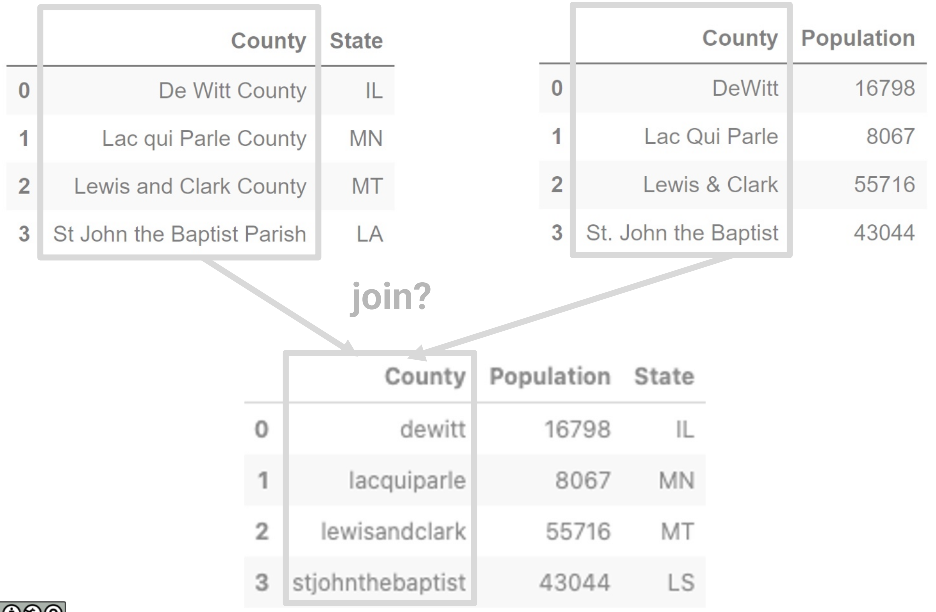
	County	Population
0	DeWitt	16798
1	Lac Qui Parle	8067
2	Lewis & Clark	55716
3	St. John the Baptist	43044

	County	Population	State
0	dewitt	16798	IL
1	lacquiparle	8067	MN
2	lewisandclark	55716	MT
3	stjohnthebaptist	43044	LS

Why work with text? Two Main Goals

1. **Canonicalization**: Convert data that has more than one possible presentation into a standard form.

Ex Join 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

	County	Population	State
0	dewitt	16798	IL
1	lacquiparle	8067	MN
2	lewisandclark	55716	MT
3	stjohnthebaptist	43044	LS

2. **Extract** information into a new feature.

Ex Extract dates and times from log files

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

$\downarrow$

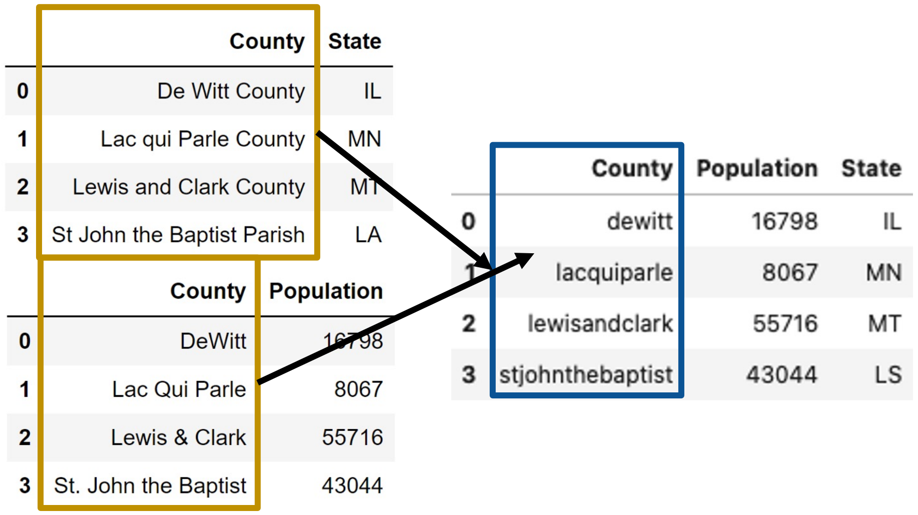
```
day, month, year = "26", "Jan", "2014"
hour, minute, seconds = "10", "47", "58"
```

Python String Methods

Why Work with Text?
Python String Methods
Regular Expressions (Regex) Basics
Regex Expanded
Convenient Regex
Regex in Python/Pandas (Regex groups)
Demo on Restaurant Data
Bonus: Yes, More Regex Syntax

Demo Slides

1. Canonicalization



	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

	County	Population	State
0	dewitt	16798	IL
1	lacquiparle	8067	MN
2	lewisandclark	55716	MT
3	stjohnthebaptist	43044	LS

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

Demo Slides

2. 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/"
```

```
day, month, year = "26", "Jan", "2014"
hour, minute, seconds = "10", "47", "58"
```

One possible solution:

```
pertinent = line.split("[")[1].split(']')[0]
day, month, rest              =
pertinent.split('/')
year, hour, minute, rest = rest.split(':')
seconds, time_zone       = rest.split(' ')
```

Summary: Python String Methods

Canonicalization and **Extraction**

- Parse/replace/split substrings.
- Feels very “hacky,” but messy problems often have messy solutions.

Python string functions:

- Are very **brittle**! Requires maintenance.
- Have **limited flexibility**.

operation	Python	pandas (Series)
transformation	`s.lower()` `s.upper()`	`ser.str.lower()` `ser.str.upper()`
replacement/ deletion	`s.replace(…)`	`ser.str.replace(…)`
split	`s.split(…)`	`ser.str.split(…)`
substring	`s[1:4]`	`ser.str[1:4]`
membership	`'ab' in s`	`ser.str.contains(…)`
length	`len(s)`	`ser.str.len()`

How would you extract all the **`moon`**-like patterns in this string?

```
"moon moo mooooooon mon mooooon"
```

Seem impossible?

String Extraction: An alternate approach

While we can hack together code that uses **`replace/split`**…

```
pertinent = line.split("[")[1].split(']')[0]
day, month, rest = pertinent.split('/')
year, hour, minute, rest = rest.split(':')
seconds, time_zone = rest.split(' ')
```

…An alternate approach is to use a **regular expression**:

- Implementation provided in the Python **`re`** library and the pandas **`str`** accessor.
- We'll spend some time today working up to expressions like this one:

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

Regex Basics

Why Work with Text?
Python String Methods
Regular Expressions (Regex) Basics
Regex Expanded
Convenient Regex
Regex in Python/Pandas (Regex groups)
Demo on Restaurant Data
Bonus: Yes, More Regex Syntax

What Is a Regular Expression?

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

- Example: “The set of all strings of length < 10 that contain `'data'`”

A **regular language** is a formal language that can be described by a **regular expression**.

A **regular expression** (“**regex**”) is a sequence of characters that specifies a search pattern.

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

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

Goals of Today's Lecture

The goal of today is :

1. Understand what regex is capable of.
2. Parse and create regex, **with a reference table**.

$\left.\vphantom{\begin{matrix}a\\b\\c\end{matrix}}\right\}$ high-level

1. Use vocabulary (closure, metacharacter, escape character, groups, etc.) to **describe regex** metacharacters.
2. **Differentiate** between `( )`, `[ ]`, `{ }`
3. Design your own **character classes** with `\d`, `\w`, `\s`, `[ …- … ]`, `^`, etc.
4. Use Python and pandas regex methods.

$\left.\vphantom{\begin{matrix}a\\b\\c\\d\\e\end{matrix}}\right\}$ details; hone with practice

References:

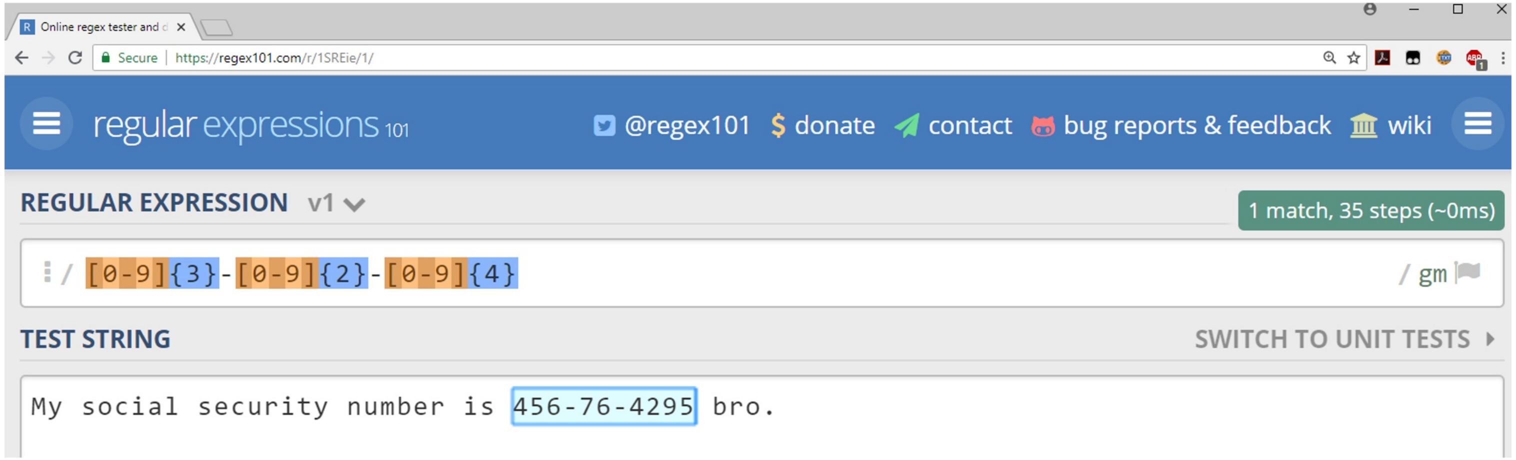
- The official guide is good! https://docs.python.org/3/howto/regex.html

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.

- Example: https://regex101.com/r/1SREie/1



1 match, 35 steps (~0ms)

```
/ [0-9]{3}-[0-9]{2}-[0-9]{4} / gm
```

```
My social security number is 456-76-4295 bro.
```

Basic Regex Syntax

The four basic operations for regular expressions.

You can technically do anything with just these basic four (albeit tediously).

|, *, () are **metacharacters**. They manipulate adjacent characters.

operation	order	example	matches	doesn't 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`
group (parenthesis)	1	`A(A|B)AAB`	`AAAAB` `ABAAB`	every other string
		`(AB)*A`	`A` `ABABABABA`	`AA` `ABBA`

`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!

ABBA

operation	order	example	matches	doesn't 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
group (parenthesis)	1	A(A|B)AAB	AAAAB ABAAB	every other string
		~~(AB)*A~~	A ABABABABA	AA ABBA

Puzzle

Starting off: regex101.com/r/8tkQ23/1 ← Click to test on regex101.com!

operation	order	example	matches	doesn't 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
group (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 os except zero (i.e. don't match mn).

Solution

operation	order	example	matches	doesn't 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`
group (parenthesis)	1	`A(A|B)AAB`	`AAAAB` `ABAAB`	every other string
		`(AB)*A`	`A` `ABABABABA`	`AA` `ABBA`

Answer: `moo(oo)*n`

Puzzle

Order of Operations: regex101.com/r/RAnuqE/1

operation	order	example	matches	doesn't 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
group (parenthesis)	1	`A(A|B)AAB`	AAAAB ABAAB	every other string
		`(AB)*A`	A ABABABABA	AA ABBA

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

Solution

operation	order	example	matches	doesn't 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
group (parenthesis)	1	A(A|B)AAB	AAAAB ABAAB	every other string
		(AB)*A	A ABABABABA	AA ABBA

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

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

Solution

Explanation

✅ `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)*`
- `oo(oo)*` followed by n

Concatenation precedes OR!

Match examples:
```
muu
muuuu
oon
ooooon
```

OR metacharacter `|` comes last in order of operations.

Regex Expanded

Why Work with Text?
Python String Methods
Regular Expressions (Regex) Basics
Regex Expanded
Convenient Regex
Regex in Python/Pandas (Regex groups)
Demo on Restaurant Data
Bonus: Yes, More Regex Syntax

Expanded Regex Syntax

wildcard .
Consider: `.*`

character class:
Match one character in [ ]

Repeat preceding item {…} times

Compare/contrast:
`o*`, `o+`, `o?`

operation	example	matches	doesn't match
any character (except newline)	`.U.U.U.`	CUMULUS JUGULUM	SUCCUBUS TUMULTUOUS
character class	`[A-Za-z][a-z]*`	word Capitalized	camelCase 4illegal
repeated exactly a times: {a}	`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
at least one	`jo+hn`	john jooooohn	jhn jjohn
zero or one	`joh?n`	jon john	any other string

Expanded Regex Syntax

wildcard .
Consider: `.*`

character class:
Match one character in [ ]

Repeat preceding item {…} times

Compare/contrast:
`o*`, `o+`, `o?`

operation	example	matches	doesn't match
any character (except newline)	`.U.U.U.`	`CUMULUS` `JUGULUM`	`SUCCUBUS` `TUMULTUOUS`
character class	`[A-Za-z][a-z]*`	`word` `Capitalized`	`camelCase` `4illegal`
repeated exactly a times: {a}	`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`
at least one	`jo+hn`	`john` `joooooohn`	`jhn` `jjohn`
zero or one	`joh?n`	`jon` `john`	any other string

Expanded Regex examples

wildcard .
Consider: .*

character class:
Match one character in [ ]

Repeat preceding item {…} times

Compare/contrast:
o*, o+, o?

matches	does not match
`.*SPB.*`	
RASPBERRY SPBOO	SUBSPACE SUBSPECIES
`[0-9]{3}-[0-9]{2}-[0-9]{4}`	
231-41-5121 573-57-1821	231415121 57-3571821
`[a-z]+@([a-z]+\.)+(edu|com)`	
horse@pizza.com horse@pizza.food.com	frank_99@yahoo.com hug@cs

\ **escapes** the next character, so **\.** matches a period.

Puzzle

Expanded Regex Puzzle 1: regex101.com/r/g0tP0I/1

operation	example	matches	doesn't match
any character (except newline)	`.U.U.U.`	`CUMULUS` `JUGULUM`	`SUCCUBUS` `TUMULTUOUS`
character class	`[A-Za-z][a-z]*`	`word` `Capitalized`	`camelCase` `4illegal`
repeated exactly a times: {a}	`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`
at least one	`jo+hn`	`john` `jooooohn`	`jhn` `jjohn`
zero or one	`joh?n`	`jon` `john`	any other string

Give a regular expression for any lowercase string that has a repeated vowel (**noon, peel, festoon, loop**, **oodles**, etc).

Solution

Expanded Regex Puzzle 1

operation	example	matches	doesn't match
any character (except newline)	`.U.U.U.`	`CUMULUS` `JUGULUM`	`SUCCUBUS` `TUMULTUOUS`
character class	`[A-Za-z][a-z]*`	`word` `Capitalized`	`camelCase` `4illegal`
repeated exactly a times: {a}	`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`
at least one	`jo+hn`	`john` `joooooohn`	`jhn` `jjohn`
zero or one	`joh?n`	`jon` `john`	any other string

Answer: `[a-z]*(aa|ee|ii|oo|uu)[a-z]*`

Puzzle

Expanded Regex Puzzle 2: https://regex101.com/r/aKyP2r/3

operation	example	matches	doesn't match
any character (except newline)	`.U.U.U.`	CUMULUS JUGULUM	SUCCUBUS TUMULTUOUS
character class	`[A-Za-z][a-z]*`	word Capitalized	camelCase 4illegal
repeated exactly a times: {a}	`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
at least one	`jo+hn`	john joooooohn	jhn jjohn
zero or one	`joh?n`	jon john	any other string

Give a regular expression for any string that contains **both a lowercase letter and a number**.

Solution

Expanded Regex Puzzle 2

operation	example	matches	doesn't match
any character (except newline)	`.U.U.U.`	`CUMULUS` `JUGULUM`	`SUCCUBUS` `TUMULTUOUS`
character class	`[A-Za-z][a-z]*`	`word` `Capitalized`	`camelCase` `4illegal`
repeated exactly a times: {a}	`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`
at least one	`jo+hn`	`john` `jooooohn`	`jhn` `jjohn`
zero or one	`joh?n`	`jon` `john`	any other string

Answer: `(.*[0-9].*[a-z].*)|(.*[a-z].*[0-9].*)`

Interlude

https://alf.nu/RegexGolf

Interlude

Email Address Regular Expression (probably a 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])*)*\<(?:(?:\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])*)(?:\.(?:(?:\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])*))*\>(?:(?:\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])*))*@(?:(?:\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])*)*\<(?:(?:\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])*)(?:\.(?:(?:\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])*))*\>(?:(?:\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])*)*\<(?:(?:\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])*)(?:\.(?:(?:\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])*))*\>(?:(?:\r\n)?[ \t])*))*)?;\s*)
```

source, StackOverflow discussion

Convenient Regex

Why Work with Text?
Python String Methods
Regular Expressions (Regex) Basics
Regex Expanded
Convenient Regex
Regex in Python/Pandas (Regex groups)
Demo on Restaurant Data
Bonus: Yes, More Regex Syntax

Convenient Regex Syntax

\w	[A-Za-z0-9_]
\d	[0-9]
\s	whitespace
+	at least one

[^...] negates entire character class

"take this next character literally"

operation	example	matches	doesn't match

built-in **character classes**	`\w+`
	`\d+`
	`\s+`
		`Fawef_03`
		`231231`
		whitespace
			`this person`
			`423 people`
			non-whitespace

character class **negation**	`[^a-z]+`	`PEPPERS3982`
		`17211!↑å`
			`porch`
			`CLAmS`

escape character	`cow\.com`	`cow.com`	`cowscom`

Puzzle

Back to Our Log File, Part 1: https://regex101.com/r/bJ9vUn/1

operation	example	matches	doesn't match

built-in **character classes**	`\w+`
	`\d+`
	`\s+`
		`Fawef_03`
		`231231`
		whitespace
			`this person`
			`423 people`
			non-whitespace

character class **negation**	`[^a-z]+`	`PEPPERS3982`
		`17211!↑å`
			`porch`
			`CLAmS`

escape character	`cow\.com`	`cow.com`	`cowscom`

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

Give a regular expression that matches the gold portion above.

Solution

Back to Our Log File, Part 1

operation	example	matches	doesn't match

built-in **character classes**	`\w+`
	`\d+`
	`\s+`
		`Fawef_03`
		`231231`
		whitespace
			`this person`
			`423 people`
			non-whitespace

character class **negation**	`[^a-z]+`	`PEPPERS3982`
		`17211!↑å`
			`porch`
			`CLAmS`

escape character	`cow\.com`	`cow.com`	`cowscom`

Answer: `\[.*\]`

Even More Regular Expression Features

A few additional common regex features are listed above.

- Won't discuss these in lecture, but **might come up** in discussion or hw.
- There are even more features out there!

operation	example	matches	doesn't 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`

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

hell
Greedy: h.+l matches hell
Lazy: h.+?l matches hel.

<em>Hello World</em>
Greedy: <.+> will match <em>Hello World</em>
Lazy: <.+?> will match <em> and </em>

Regex in Python and Pandas (Regex groups)

Why Work with Text?
Python String Methods
Regular Expressions (Regex) Basics
Regex Expanded
Convenient Regex
Regex in Python and Pandas (regex groups)
Demo on Restaurant Data
Bonus: Yes, More Regex Syntax

Canonicalization: Python

`re.sub(pattern, repl, text)` docs

Returns text with all instances of **`pattern`** replaced by **`repl`**.

```
text = ‘<div><td
valign="top">Moo</td></div>’
pattern = r"<[^>]+>"
re.sub(pattern, '', text) # returns Moo
```

```
Moo
```

Canonicalization: Pandas

`re.sub(pattern, repl, text)` docs

Returns text with all instances of `pattern` replaced by `repl`.

```
text = ‘<div><td valign="top">Moo</td></div>’
pattern = r"<[^>]+>"
re.sub(pattern, '', text) # returns Moo
```

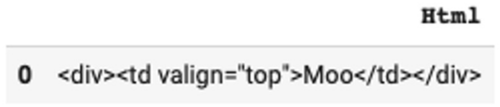
```
Moo
```

`pattern` is a **raw string**. `r"..."`

`ser.str.replace(pattern, repl, regex=True )` docs

Returns Series with all instances of `pattern` in Series `ser` replaced by `repl`.

```
df["Html"].str.replace(pattern, '')
```



	Html
0	<div><td valign="top">Moo</td></div>

```
0            Moo
Name: Html, dtype: object
```

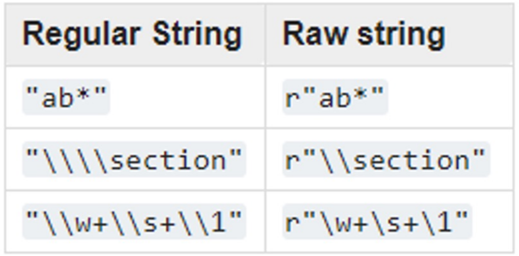
Sidenote: Raw Strings in Python

Note: When specifying a pattern, we strongly suggest using **raw strings**.

- A raw string is created using `r""` or `r''` instead of just `""` or `''`.

```
pattern = r"<[^>]+>"
```

- The exact reason is a bit tedious.
  - Rough idea: Regular expressions and Python strings both use \ as an escape character.
  - Using non-raw strings leads to uglier regular expressions.



Regular String	Raw string
`"ab*"`	`r"ab*"`
`"\\\\section"`	`r"\\section"`
`"\\w+\\s+\\1"`	`r"\w+\s+\1"`

For more information see “The Backslash Plague” under
https://docs.python.org/3/howto/regex.html#the-backslash-plague

Extraction

`re.findall(pattern, text)`

docs

Return a list of all matches to **`pattern`**.

```
text = "My social security number is 123-45-
6789 bro, or actually maybe it’s 321-45-
6789.";
pattern = r"[0-9]{3}-[0-9]{2}-[0-9]{4}"
re.findall(pattern, text)
```

```
['123-45-6789', '321-45-6789']
```

Extraction

`re.findall(pattern, text)`

docs

Return a list of all matches to `pattern`.

```
text = "My social security number is 123-45-
6789 bro, or actually maybe it’s 321-45-
6789.";
pattern = r"[0-9]{3}-[0-9]{2}-[0-9]{4}"
re.findall(pattern, text)
```

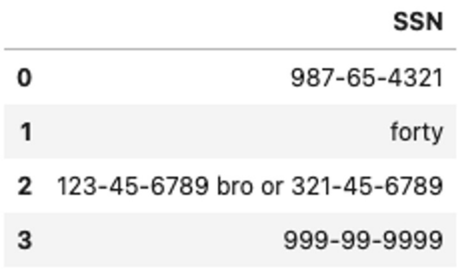
```
['123-45-6789', '321-45-6789']
```

`ser.str.findall(pattern)`

docs

Returns a Series of lists

```
df["SSN"].str.findall(pattern)
```



	SSN
0	987-65-4321
1	forty
2	123-45-6789 bro or 321-45-6789
3	999-99-9999

```
0                [987-65-4321]
1                           []
2   [123-45-6789, 321-45-6789]
3                [999-99-9999]
Name: SSN, dtype: object
```

Regular Expression Capture Groups

Earlier we used parentheses to specify the **order of operations**.

Parenthesis have **another meaning**:

- Every set of parentheses specifies a **match/capture group**.
- In Python, matches are returned as tuples of groups.

```
text = """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, text)
```

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

There's more than one way to regex, e.g. `(\d\d)` vs `(\d{2})`

Back to Our Log File, Part 2

With this notion of groups, let’s come back to the regex presented without explanation earlier.

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

operation	example	matches	doesn’t match

built-in **character classes**	`\w+`
	`\d+`
	`\s+`
		`fawef`
		`231231`
		whitespace
			`this person`
			`423 people`
			non-whitespace

at least one	`jo+hn`	`john` `joooooohn`	`jhn` `jjohn`
escape character	`cow\.com`	`cow.com`	`cowscom`
any character (except newline)	`.U.U.U.`	`CUMULUS` `JUGULUM`	`SUCCUBUS` `TUMULTUOUS`

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

```
('26', 'Jan', '2014', '10',
      '47', '58', '-0800')
```

Finding regex groups

`re.findall(pattern, text)` docs

(Python) Return a list of all matches to `pattern`.

`ser.str.findall(pattern)` docs

Returns a Series of lists of all matches.

`ser.str.extract(pattern)` docs

Returns a DataFrame of first match,
one group per column

`ser.str.extractall(pattern)` docs

Returns a DataFrame of all matches,
one group per column, one row per match

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.

Some people, when confronted with a problem, think 'I know, I'll use regular expressions.'
Now they have two problems.

Jamie Zawinski (Source)

Regular expressions sometimes jokingly referred to as a "**write only language**".

Regular expressions are terrible at certain types of problems:

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

However, regular expressions are decent at **wrangling text data**.

Demo on Restaurant Data

Why Work with Text?
Python String Methods
Regular Expressions (Regex) Basics
Regex Expanded
Convenient Regex
Regex in Python/Pandas (Regex groups)
Demo on Restaurant Data
Bonus: Yes, More Regex Syntax

String function summary

Today we saw many, many different string manipulation tools (highlighted).

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

Python String	re	pandas Series

`s.lower()`
`s.upper()`
		`ser.str.lower()`
		`ser.str.upper()`

`s.replace(…)`	**`re.sub(…)`**	**`ser.str.replace(…)`**
`s.split(…)`	`re.split(…)`	`ser.str.split(…)`
`s[1:4]`		**`ser.str[1:4]`**

	`re.findall(…)`	**`ser.str.findall(…)`**
		`ser.str.extractall(…)`
		`ser.str.extract(…)`

`'ab' in s`	`re.search(…)`	`ser.str.contains(…)`
`len(s)`		`ser.str.len()`
`s.strip()`		`ser.str.strip()`

Bonus: Yes, More Regex Syntax

Why Work with Text?

Python String Methods

Regular Expressions (Regex) Basics

Regex Expanded

Convenient Regex

Regex in Python/Pandas (Regex groups)

Demo on Restaurant Data

Bonus: Yes, More Regex Syntax

Optional (but Handy) Regex Concepts

These regex features aren't going to be on an exam, but they are useful:

- **Lookaround**: match "good" if it's not preceded by "not": `(?<!not )good`
- **Backreferences**: match HTML tags of the same name: `<(\w+)>.*</\1>`
- **Named groups**: match a vowel as a named group:
  `(?P<vowel>[aeiou])`
- **Free Space**: Allow free space and comments in a pattern.

```
# Match a 20th or 21st century date in yyyy-mm-dd format
(19|20)\d\d                 # year (group 1)
[- /.]                      # separator
(0[1-9]|1[012])             # month (group 2)
[- /.]                      # separator
(0[1-9]|[12][0-9]|3[01])    # day (group 3)
```

BONUS MATERIAL

Of these concepts, **named groups** is the most useful for **extraction**.

Exercise

1. Which strings contain a match for the following regular expression, `"1+1$"`? The character `"␣"` represents a single space.

   ◯ A. `What␣is␣1+1` ◯ B. `Make␣a␣wish␣at␣11:11` ◯ C. `111␣Ways␣to␣Succeed`

B

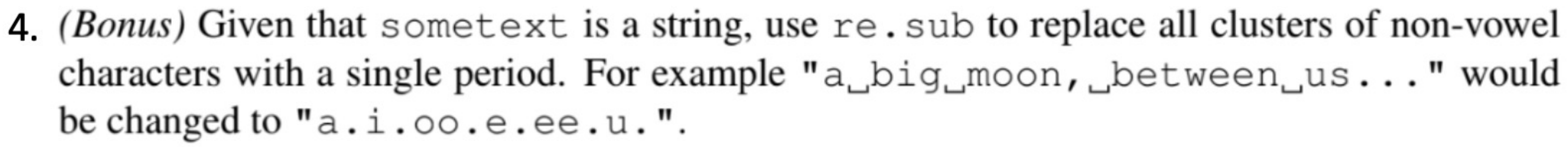
2. Write a regular expression that matches strings (including the empty string) that only contain lowercase letters and numbers.

^[a-z0-9]*$

3. Given `sometext = "I've␣got␣10␣eggs,␣20␣gooses,␣and␣30␣giants."`, use `re.findall` to extract all the items and quantities from the string. The result should look like **`['10 eggs', '20 gooses', '30 giants']`**. You may assume that a space separates quantity and type, and that each item ends in s.

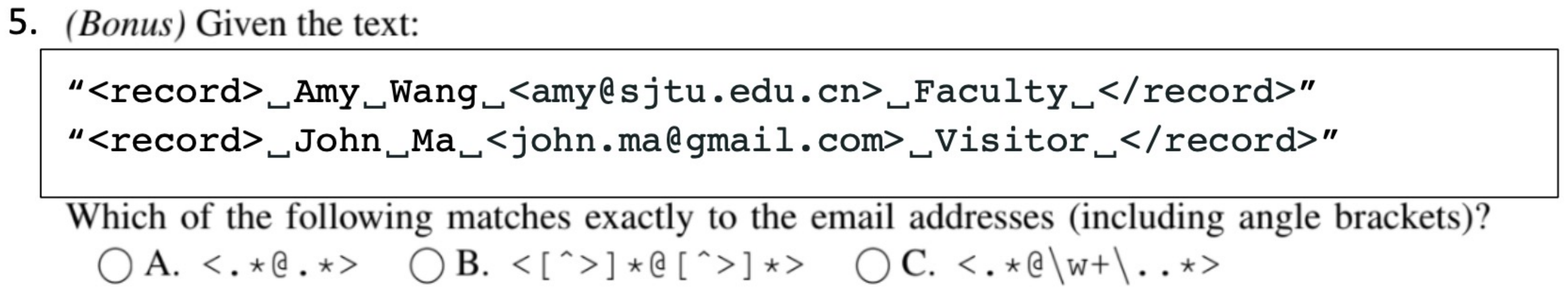
[0-9]
re.findall(r"\d\d\s\w+s",sometext)

Exercise



4. *(Bonus)* Given that `sometext` is a string, use `re.sub` to replace all clusters of non-vowel characters with a single period. For example `"a␣big␣moon,␣between␣us..."` would be changed to `"a.i.oo.e.ee.u."`.

re.sub(r"[^aeiou]+",".",sometext)



5. *(Bonus)* Given the text:

```
"<record>␣Amy␣Wang␣<amy@sjtu.edu.cn>␣Faculty␣</record>"
"<record>␣John␣Ma␣<john.ma@gmail.com>␣Visitor␣</record>"
```

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

◯ A. `<.*@.*>` ◯ B. `<[^>]*@[^>]*>` ◯ C. `<.*@\w+\..*>`

(B)