slido

2587070

Join at slido.com
#2587070

ⓘ Click **Present with Slido** or install our Chrome extension to display joining instructions for participants while presenting.

⚠️ Reminder to start the Zoom recording!

2587070

LECTURE 6

Regex

Using string methods and regular expressions (regex) to work with textual data

Data 100, Summer 2025 @ UC Berkeley

Josh Grossman and Michael Xiao

📣 Announcements

2587070

Homework 2A due tomorrow!

Lab 2B due Thursday, July 3rd!

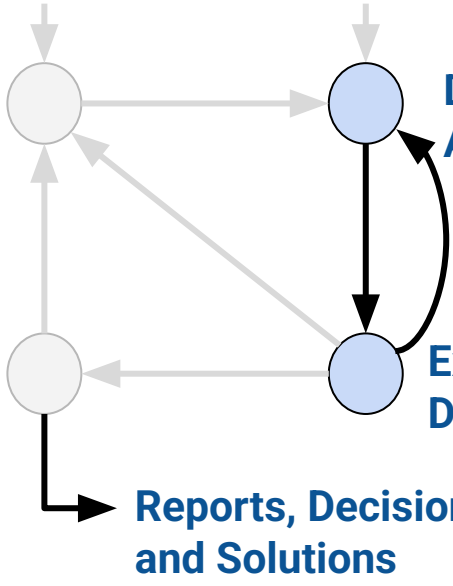
Homework 2B due Monday, July 7th

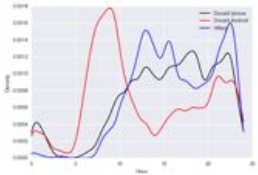
Reminder to make sure your **DSP accommodations are submitted ASAP**

- **By Sunday, July 6th** at the latest
- Very important if you have exam accommodations

This Week

2587070





(Last Lecture)

Data Wrangling
Intro to EDA

(Today)

Working with Text Data
Regular Expressions

(Next)

Visualization
Code for plotting data

2587070

Goals for this Lecture

Lecture 6, Data 100 Summer 2025

Common EDA task: clean text!

- Operate on text data using pandas **`str`** methods
- Apply **regex** to identify patterns in strings

2587070

Why Work With Text?

Lecture 6, Data 100 Summer 2025

- **Standard Text Manipulation Tasks**
- `pandas str` methods
- Why regex?
- Regex basics
- Regex functions

Why Work With Text? Two Common Goals

2587070

1. **Canonicalization**: Convert data 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	State	Population
0	dewitt	IL	16798
1	lacquiparle	MN	8067
2	lewisandclark	MT	55716
3	stjohnthebaptist	LS	43044

Regex Pipeline for Canonicalizing High School Names

2587070

Two datasets needed to be merged based on HS name and location.

Problem: HS names not canonicalized.

For example: "The Bear Preparatory High School" and "Bear Prep"

Solution: Canonicalize with regex! →

```r
simplify_school_name <- function(school_name) {
  # Heuristics for making high school and college names simpler for matching

  school_name %>%
    str_to_lower %>%
    str_replace_all("\\bschool\\b", "") %>%
    str_replace_all("\\bhigh\\b", "") %>%

    # Often high schools can have same simple name as elementary
    # and middle schools, so keep the distinction for now so
    # the simple names are different
    # str_replace_all("\\belem(entary)?\\b", "") %>%

    # H S is an abbv. for high school
    str_replace_all("\\bh\\s?s\\b", "") %>%

    str_replace_all("\\bsenior|charter|college|international|intl\\b", "") %>%
    str_replace_all("\\bacad(emy)?\\b", "") %>%
    str_replace_all("\\btech(nical)?\\b", "") %>%
    str_replace_all("\\bprep(aratory)?\\b", "") %>%
    str_replace_all("\\b(the|of|and|for|at|\\@)\\b", "") %>%

    # st: (mary's) --> st marys
    str_replace_all("[\\'\\:\\)\\(]", "") %>%

    # st. john & mary-joseph --> st john mary joseph
    str_replace_all("[\\.\\-\\/\\&]", " ") %>%

    # removes duplicate whitespace and starting/ending whitespace
    str_squish
}
```

Research Paper: Tomkins et al. (2023)

Why Work With Text? Two Common Goals

1. **Canonicalization**: Convert data 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

join?

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

2. **Extract** information.

Ex Extract dates and times from log files

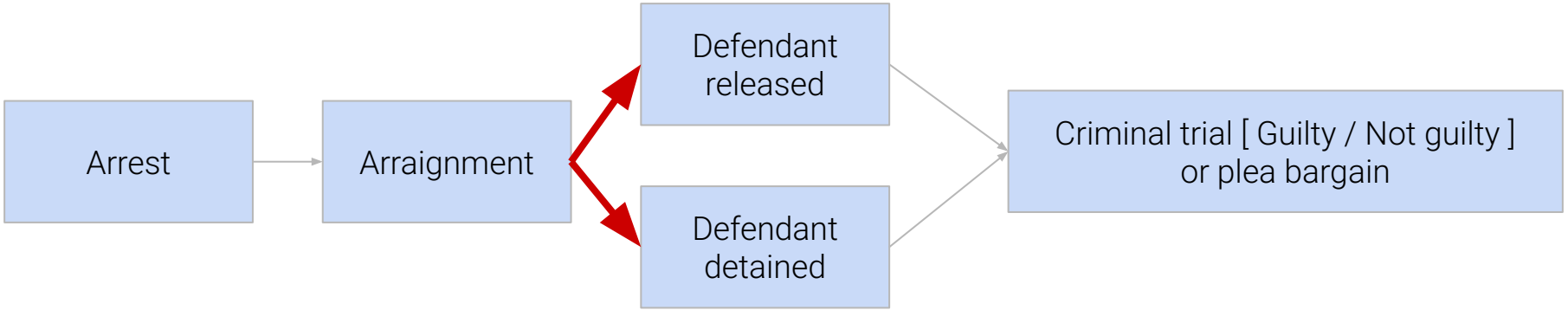
```
169.237.46.168 - -
[26/Jan/2014:10:47:58 -0800] "GET
/stat141/Winter04/ HTTP/1.1" 200 2585
"http://anson.ucdavis.edu/courses/"
```

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

Project with a Federal District Court

2587070

Motivating question: Can we make **pretrial detention** decisions more equitably?



Defendant Info in PDF Bail Reports

2587070

District/Office	Charge(s) (Title, Section, and Description)
[redacted]	21 U.S.C. §841(a)(1),(b)(1)(C)
Judicial Officer The Honorable [redacted] U.S. Magistrate Judge	
Docket Number (Year – Sequence No. – Def. No.) 4:18-cr-00001-XX-1	

DEFENDANT

Name	DOB:	Employer/School	
[redacted]	7/11/1977	UNEMPLOYED	
Address [redacted]		Employer/School Address N/A	
Time At Address 7 Months/7 Years	Time in Community Life	Monthly Income $0	Time with Employer/School N/A

PREBAIL REPORT
(Prepared on December 12, 2018)

Defendant Info in PDF Bail Reports

2587070

03/22/2008 [redacted]	1. No Arrest 2. **Possess Methaqualone**	03/15/2010**: Convicted of Count 2 (Felony);** Sentence: 5 Years Probation 03/12/2010: "Conviction Certified by [redacted] 03/26/2013: Sentence Modified: 5 Years Probation, 28 Days Jail
08/23/2008 [redacted]	**DUI Alcohol/Drugs**	04/22/2009: Subsequent Count of Drive: License Suspended/Etc: Specific Violation – Dismissed/Furtherance of Justice/Plea to Other Charge **Convicted (Misdemeanor);** Sentence: 3 Years Probation, 15 Days Jail

XXX County records provided by XXX reflect that the defendant has **14 Failures to Appear and two prior probation revocations in XXX County.**

Identify and Extract Data Using Patterns in Text

2587070

```
03/22/2008
XXX County, XXX
1. No Arrest
2. Possess Methaqualone
03/15/2010: Convicted of Count 2
(Felony); Sentence: 5 Years
Probation
03/12/2010: "Conviction
Certified
by XXX, Court Clerk, XXX County"
03/26/2013: Sentence Modified: 5
Years Probation, 28 Days Jail
08/23/2008
...
```

```
XXX, XXX
DUI Alcohol/Drugs 04/22/2009:
Subsequent Count of
Drive: License Suspended/Etc:
Specific Violation -
Dismissed/Furtherance of
Justice/Plea to Other Charge
Convicted (Misdemeanor);
Sentence: 3 Years Probation, 15
Days Jail
...
XXX County records provided by
XXX reflect that the defendant
has 14 Failures to Appear
and two prior probation
revocations in XXX County.
...
```


For a more detailed description, see footnote 12 in onlinelibrary.wiley.com/doi/abs/10.1111/jels.12343

2587070

`pandas str` Methods

Lecture 6, Data 100 Summer 2025

- Why work with text?
- **`pandas str` methods**
- Why regex?
- Regex basics
- Regex functions

From String to `str`

2587070

In "base" Python, we have various string operations to work with text data.

Recall:

transformation	`s.lower()` `s.upper()`
split	`s.split(…)`
membership	`'ab' in s`

replacement/ deletion	`s.replace(…)`
substring	`s[1:4]`
length	`len(s)`

Problem: Python assumes we are working with **one string at a time**. Looping can be slow!

`str` Methods

Pandas `str` methods are **vectorized**. No looping; simultaneous computation!

`Series.str.<string_operation>()`

`.str` is a `pandas` accessor, just like `.dt` from the EDA lecture. Accessors provide a wide variety of domain-specific functions.

Apply the function *<string_operation>* to every string in the `Series` → Vectorized!

```
populations["County"]
```

```
0                  DeWitt
1           Lac Qui Parle
2           Lewis & Clark
3    St. John the Baptist
Name: County, dtype: object
```

```
populations["County"].str.lower()
```

```
0                  dewitt
1           lac qui parle
2           lewis & clark
3    st. john the baptist
Name: County, dtype: object
```

`.str` Methods

2587070

Most base Python string operations have a **`pandas str`** equivalent

Operation	Python (single string)	pandas (Series of strings)
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()`

Demo 1: Canonicalization

2587070

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

```
def canonicalize_county(county_series):
 return (county_series
                .str.lower()                  # lowercase
                .str.replace(' ', '')         # remove space
                .str.replace('&', 'and')      # replace &
                .str.replace('.', '')         # remove dot
                .str.replace('county', '')
                .str.replace('parish', '')
           )
```

Demo

lec06.ipynb

Demo

lec06.ipynb

Demo 2: Extracting Date Info (The Long Way)

2587070

```
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, rest2 = rest.split(':')
seconds, time_zone       = rest2.split(' ')
```

Note: While you should understand the code in this part of the demo, regex is a sleeker way to solve the problem above.

Why regex?

Lecture 6, Data 100 Summer 2025

- Why work with text?
- `pandas str` methods
- **Why regex?**
- Regex basics
- Regex functions

String Extraction: An Alternate Approach

2587070

While we can sometimes "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(' ')
```

It often won't work.

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

```
"moon moo mooooooon mon moooon"
```

Circa 2013 meme, "Moon moon"

String Extraction: An Alternate Approach

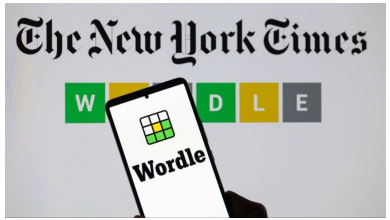
2587070

An alternate approach is to use a **regular expression**.

- Implementation provided in the Python **`re`** library and the pandas **`str`** accessor.
- We can simplify the code in the previous demo with regex:

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

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



Productive mindset to adopt: Think of regex problems like **word puzzles**!

2587070

Regex Basics

Lecture 6, Data 100 Summer 2025

- Why work with text?
- `pandas str` methods
- Why regex?
- **Regex basics**
- Regex functions

Regular Expressions Specify Patterns in Strings

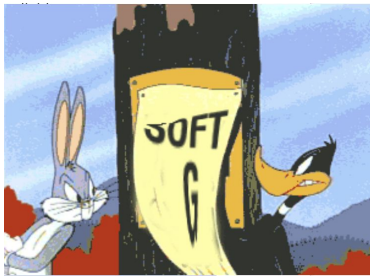
2587070

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.

The language of Social Security Numbers (e.g., 123-45-6789) is described by this regular expression.



"Regex" pronunciation? (as in Re**g**ular)
Check out English Stackexchange discussion

Goals for regex

2587070

The goal of today is NOT to memorize the language of regular expressions!

Instead:

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

Resources for Practicing Regex

2587070

Many resources to experiment with regexes (e.g., regex101.com, regexone.com, …)

For experimenting, we recommend regex101.com. We will use it during today's demos.

- **Important:** Choose the Python "flavor" in the left sidebar. We'll explain the **`r"`** soon!
- Note the reference table in the bottom right.

Regex Basics

2587070

There are four basic operations in regex.

Concatenation – "look for consecutive characters"

`BAAB` matches `BAAB`

`|` – "or"

`BAB|BAAB` matches `BAB` *or* `BAAB`

`*` – "zero or more"

`AB*A` matches `AA`, `ABA`, `ABBA`, ...

`()` – "consider a group"

`(AB)*A` matches `A`, `ABA`, `ABABA`, ...
`A(A|B)AAB` matches `AAAAB` *or* `ABAAB`

*, `(` `)`, and `|` are called **metacharacters** – they represent an operation, rather than a literal text character

Summary So Far

2587070

Operation	Order	Example	Matches	Doesn't match
concatenation (consecutive chars)	3	`AABAAB`	`AABAAB`	every other string
or, |	4	`AA|BAAB`	`AA` `BAAB`	every other string
***** (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`

The regex order of operations. Grouping is evaluated first.

slido

2587070

Which pattern matches moon, moooon, etc? Your expression should match any EVEN number of “o”s, excluding zero (e.g., don’t match mn or mooon).

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

Try it yourself!

2587070

regex101.com/r/8tkQ23/1

moo(oo)*n

Regex Expanded

2587070

Six more regex operations.

newline

. – "look for *any* character other than \n"

`.U.U.U.` matches `CUMULUS, JUGULUM`

[] – "define a character class"

`[A-Za-z]` matches `A, a, B, b...`

+ – "one or more"

`AB+` matches `AB, ABB, ABBB, ...`

? – "zero or one" ("optional")

`AB?` matches `A, AB`

{x} – "repeat exactly x times"

`AB{2}` matches `ABB`

{x, y} – "repeat between x and y times"

`AB{0,2}` matches `A, AB, ABB`

Keep in mind: ***** = **{0,}**, **+** = **{1,}**, and **?** = **{0,1}** = **{,1}**

Character Classes

2587070

[A-Z] – any uppercase letter between A and Z

[0-9] – any digit between 0 and 9

[A-Za-z0-9] – any letter, any digit

Regex built-in classes:

\w is equivalent to [A-Za-z0-9]

\d is equivalent to [0-9]

\s matches space, tab or newline

Use **^** to negate a class = match any character *other* than what follows

[^A-Z] – any character that is *not* an uppercase letter between A and Z

Capitalized shortcuts: **[^A-Za-z0-9] = [^\w] = `\W` [^\d] = `\D` [^\s]=`\S`**

Summary So Far

2587070

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`

What pattern only matches lowercase alphabetic strings that have a repeated vowel (e.g., bazaar, beetroot, radii, oodles, vacuum)?

2-minute stretch!

2587070

How Josh learned regex: regexcrossword.com

Email Address Regular Expression (probably a bad idea)

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

source, StackOverflow discussion

Greediness

2587070

Regex is **greedy** – it looks for the *longest possible* match in a string.

```
<div>.*</div>
```

regex101.com/r/HATiTH/1

"This is an ** example of greediness in ** regular expressions."

Greediness

2587070

Regex is **greedy** – it looks for the *longest possible* match in a string.

```
<div>.*</div>
```

In English:

- "Look for the exact string "
- then, "grab every character except \n…"
- "… until the **FINAL** instance of the string "

"This is an ** **example of greediness in** ** regular expressions."

Greediness

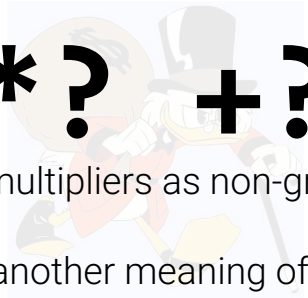
2587070

Regex is **greedy** – it looks for the *longest possible* match in a string.

```
<div>.*?</div>
```

In English:

- "Look for the exact string "
- then, "grab every character except \n…"
- "… until the **FIRST** instance of the string "



? tags multipliers as non-greedy. Docs.

This is another meaning of the **?** modifier!

"This is an ** **example of greediness ** **in regular expressions."

Regex *Even More* Expanded

2587070

The last set!

**** – "read the next character literally"

`a\+b` matches `a+b`

^ – "match the beginning of a string"

`^abc` matches "`abc 123`", not "`123 abc`"

$ – "match the end of a string"

`abc$` matches "`123 abc`", not "`abc 123`"

Be careful: ^ has different behavior inside/outside of character classes!

[^abc] → Match any single character other than a, b, or c

Evan Misshula: Ex-presidents often end up rich. You **start** with **power ^** and **end** with **money $**

Summary So Far

2587070

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

2587070

Which of the following strings matches this regex expression?

^w+\.be?r(oco|ke)I+.*\.(edu|com)$

Which of the following strings matches this regex expression?
^w+\.be?r(oco|ke)l+.*\.(edu|com)$

- www.berkeley.edu
- https://www.berkeley.edu
- www.brocolli.com
- w.berocoley.edu
- www.berkeley.edu/brocolli

Operation	Order	Example	Matches	Doesn't match
concatenation (consecutive chars)	3	AABAAB	AABAAB	every other string
or, |	4	AA|BAAB	AA BAAB	every other string
* (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

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 jooooooohn	jhn jjohn

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
escape character	cow\.com	cow.com	cowscom

Additional Regex Functionality

2587070

Regex101.com is great for learning basic regex *syntax*.

For full functionality of regex (matching, splitting, search and replace, group management, …), see **The Python Regex HOWTO**: docs.python.org/3/howto/regex.html.

Regex is also a sleek way to find+replace in your favorite text editor (even Google Slides!)

Regex Functions

Lecture 6, Data 100 Summer 2025

- Why work with text?
- `pandas str` methods
- Why regex?
- Regex basics
- **Regex functions**

Before We Begin: Raw Strings in Python

2587070

When specifying a pattern, use **raw strings**.

```
pattern = r"[0-9]+"
```

Create by putting **r** before string delimiters:
(**r"..." r'...', r"""..."""**, **r'''...'''**)

Python **and** Regex each use backlash (\) as the **escape character.**

Standard string	Raw string	Matches
"ab*"	r"ab*"	a, ab, abb, …
"\\w+\\s+"	r"\w+\s+"	One or more of [A-Za-z0-9], then one or more spaces
"\\\\section"	r"\\section"	\section

For more information see "The Backslash Plague"

Why we need four backslashes \\\\ to match one backlash

2587070

Suppose we want to match the **literal text '\n'** in a document (i.e, NOT a newline)

print('**\n**') prints a newline

print('**\\n**') prints '\n', which regex would then interpret as the literal character 'n'

print('**\\\n**') prints a literal \, followed by a newline

print('**\\\\n**') prints '\\n', which regex would interpret as the literal string '\n' → Done… 🥲

print(**r'\\n'**) prints '\\n', which regex would interpret as the literal string '\n' → Easier! 😎

Note: All of these examples are in the demo!

Extraction

2587070

`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']
```

A **match** is a substring that matches the provided regex.

Extraction

2587070

`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']
```

A **match** is a substring that matches the provided regex.

`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
```

Extraction with Capture Groups

2587070

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

( ) also specifies a **capture group**.

- Some **`re`** functions extract *only* the text matched by capture groups, if they are specified

```
text = """I will meet you at 08:30:00 pm tomorrow"""
pattern = ".*(\d\d):(\d\d):(\d\d).*"
matches = re.findall(pattern, text)
matches
```

```
[('08', '30', '00')]
```

The capture groups each capture two digits.

Extraction with Capture Groups

2587070

`ser.str.extract(pattern)` docs

Returns a DataFrame of each capture group's **first** match in the string

```
pattern_cg = r"([0-9]{3})-([0-9]{2})-([0-9]{4})"
df["SSN"].str.extract(pattern_cg)
```

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

	0	1	2
0	987	65	4321
1	NaN	NaN	NaN
2	123	45	6789
3	999	99	9999

`ser.str.extractall(pattern)` docs

Returns a multi-indexed DataFrame of **all** matches for each capture group

```
df["SSN"].str.extractall(pattern_cg)
```

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

		0	1	2
match				
0	0	987	65	4321
2	0	123	45	6789
	1	321	45	6789
3	0	999	99	9999

Substitution

2587070

`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)
```

Moo

How it works:

- **`pattern`** matches HTML tags
- Then, sub/replace HTML tags with **`repl=''`** (i.e., empty string)

Substitution

2587070

`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)
```

```
Moo
```

How it works:

- **`pattern`** matches HTML tags
- Then, sub/replace HTML tags with **`repl=''`** (i.e., empty string)

`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, '',
                       regex = True)
```

	Html
0	<td valign="top">Moo</td>
1	Link
2	Bold text

```
0          Moo
1         Link
2    Bold text
Name: Html, dtype: object
```

re.sub and ser.str.replace

2587070

findall → list of matches

extract → DataFrame of matches

sub/replace → Convert matches

Demo

lec06.ipynb

String Function Summary

2587070

Base Python	re	pandas str
`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()`

Limitations of Regular Expressions

2587070

Easier to write than to read.

Regular expressions sometimes jokingly referred to as a "**write only language**". A famous 1997 quote from Jamie Zawinski (co-creator of Firefox's predecessor)

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

Regular expressions are terrible at certain types of problems:

- For parsing a hierarchical structure, such as JSON, use the `json.load()` parser, not regex!
- Parsing real-world HTML/xml (lots of ...<tag>..</tag>..): use `html.parser`.
- Counting (same number of instances of a and b). (impossible)

LLMs are also great at regex tasks! But, sometimes unreliable + computationally expensive.

2587070

LECTURE 6

Regex

Content credit: Acknowledgments