Lesson Plan

1. Why text processing?
2. Opening and reading files
3. Regular Expressions
    - a. What are they?

    <Example 1: Find exact word from address>
    - b. Literal Characters
        - USA
    - c. Meta Characters
        - i. How to extract zip-codes
            1. Quantifiers == {5}
            2. \d{5}
        - ii. How to extract the states?
            1. \w{2} doesn't work
            2. \b\w{2}\b doesn't work
            3. ‘, \w{2} ’ does but it captures the comma too
            4. ‘, (\w{2}) ’ in python would capture the state separately

    <Example 2: Find phone numbers>
    - d. Quantifiers
        - i. \d{3}-\d{3}-\d{4}
    - e. Position
        - i. ^\d{3}-\d{3}-\d{4}
    - f. Character Classes
        - i. \d{3}[\.-]\d{3}[\.-]\d{4}
        - ii. \d{3}[\.\-]\d{3}[\.\-]\d{4}
        - iii. \(*\d{3}\)*[\.\-]\d{3}[\.\-]\d{4}
        - iv. \+?(1-)?\(*\d{3}\)*[\.\-]\d{3}[\.\-]\d{4}

    <Example 4: emails extraction>
        \w+@\w+\.\w+
        \w+@\w+\.\w{3}
        Alternation
            \w+@\w+\.(com|net|edu)
            \w+@\w+\.(ece|cs)\.ucsb\.edu

    <Example 3: HTML extraction>
    - g. <\w*>
    - h. <H\d>
    - i. Capturing groups and Back references
        - i. <(H\d)>.*<\/\1>
        - ii. <(H\d)>(.*)<\/\1>
4. Regex in Python

    <Example 4: Phone Number>
    - a. match(), search(), findall()

    <Example 5: extracting lists>
    - b. split()

    <Example 6: HTML substitution>
    - c. replace: sub()

Why do we want to do text processing?

It's a lot of work to do stuff by hand, this way things can be automated
Data is rarely clean and consistent
Frequently used for scraping webpages
Also can be used for validation!

How do we read data from files?

TXT File

```
with open("test.txt", "r") as f:
        contentList = f.readlines()
```

CSV file

```
import csv
with open("test.csv", "r") as f:
        reader = csv.reader(f, delimiter="\t")
        contentList = list(reader)
```

Notes: r = read, b = binary, w = write
Delimiter examples: tab, space, comma, pipe

How do we extract information from text?

Regular Expressions! (regex) They're awesome like little puzzles
They are not language specific
They are really cryptic writing

Regular expressions are a sequence of characters that defines a search pattern

ctrl f is like doing a regular expression for literal characters
Literal character, like specifically "P" followed by "y" followed by "t" followed by "h"
That's not particularly exciting though

What's cool is:
Meta character (some type of more generalized character like "any vowel" or "any digit")
    Meta characters subtle differences between languages

Quantifiers --number of characters that fit previous meta or literal character

Position --shows where in the string you should be looking

Go through how to detect dates using meta characters

If you want to use a character as a literal character which is typically a meta character put a \ infront of it, like \\ or \[

Single character	quantifiers	position
. any character except new line	* 0 or more	^ start of the string
`\d` Matches any decimal digit; this is equivalent to the class `[0-9]`	+ 1 or more	$ end of the string
`\D` Matches any non-digit character; this is equivalent to the class `[^0-9]`	? 0 or 1	\b word boundary
`\s` Matches any whitespace character; this is equivalent to the class `[ \t\n\r\f\v]`	{2} exactly 2	\B non-word boundary
`\S` Matches any non-whitespace character; this is equivalent to the class `[^\t\n\r\f\v]`	{2,5} between 2 and 5	
`\w` Matches any alphanumeric character; this is equivalent to the class `[a-zA-Z0-9_]`	{2,} 2 or more	
`\W` Matches any non-alphanumeric character; this is equivalent to the class `[^a-zA-Z0-9_]`	{,5} up to 5	

Character Classes

- stuff that appears between square brackets [ ]
- [abc] matches an a or b or c
- [a-ce] matches an a or b or c or e

Alternation

- (net|com) matches net or com

Capturing groups and Back reference

- Using parenthesis can capture groups, so the full match is group 0, the first capture is group 1, the third capture is group 2
- Back reference: You can reference a group one in the regular expression with a \1

Using Regular Expressions in Python

https://docs.python.org/2/howto/regex.html

Regex

- match() you don't have to compile every time so can be more efficient if using the same thing over and over again.
- findall() returns all as a list instead of just the first one

match()	Determine if the RE matches at the beginning of the string.
search()	Scan through a string, looking for any location where this RE matches.
findall()	Find all substrings where the RE matches, and returns them as a list.

```
import re

pattern = '\d{3}-\d{3}-\d{4}'
string0 = '555-867-5309'
string1 = 'Jenny’s phone number is 555-867-5309'

prog = re.compile(pattern)
match = prog.match(string0)
match.group()
match.span()

match = prog.match(string1)
match.group()

match = re.search(pattern, string1)
match.group()
match.span()

matches = re.findall(pattern, string1)
for i in matches:
	print(i)
```

Splitting in Regex

split()	Split the string into a list, splitting it wherever the RE matches

```
import re
food = re.split(r’\sand\s’, ‘oreos and baked beans and spam and soda’)
print food
```

Replace in Regex

sub()	Find all substrings where the RE matches, and replace them with a different string
subn()	Does the same thing as **sub()**, but returns the new string and the number of replacements

```
groceries = re.sub(r'\sAnd\s', ' & ', 'Baked Beans And Spam')
print groceries
```

EXAMPLE IDEAS:

- Phone numbers
- Physical Addresses
- email addresses
- Matching a password
- Matching a URL
- Finding all the Colleges/Universities in a document?
- Matching a HTML tag
- Matching duplicate words
- Removing white space
- Matching an IP address
- Matching Valid dates
- Finding or verifying credit card numbers
- File names (and file name editing)
- Finding dates and reformatting them
- Finding prices