Introduction to Computation for the Humanities and Social Sciences

CS 3

Chris Tanner

Lecture 14

Regular Ex*

* Not a Taylor Swift song

Lecture 14

- While Loops
- Regular Expressions

While Loops

- We've talked about iteration. Namely, via `for-loops`
- `for-loop's` iterate for a pre-defined number of times (e.g., `for i in range(10):` or `for name in names:`)
- Alternatively, we could iterate forever!! (or `while` a certain condition remains true… which is more reasonable).
- Introducing, `while` loops

While Loops

```
while (this boolean statement is True):
    do these lines of code
    and this one too
    followed by this one
```

While Loops

```
1    input_file = open("test.csv", 'r')
2    while age < 35:
3        cols = input_file.readline().split(",")
4        age = int(cols[3])
5        print("age is: " + str(age))
```

Lecture 14

- While Loops
- Regular Expressions

Cleaning Texts

- Throughout the textual analysis section of the course, we have provided **cleaned** texts

- A **cleaned** input is one that has been taken from its original raw form, and has been converted to a form that is easy for processing.

- For example, often we want to remove or replace special characters from the original text to simplify the grouping of words or sentences

Finding Common Patterns

- Often in textual analysis, you are interested in finding words or phrases that match a **pattern** (e.g., a bunch of letters together followed by a comma)

- If the pattern is found, then you often want to either <u>replace</u> that pattern (e.g., remove the comma) and/or <u>return</u> the pattern that was matched

Finding all Matched Patterns — Manually!

- How would we extract the hashtags from this tweet?

Finding all Matched Patterns — Manually!

```
tweet = "RT @jleicole For #WHD2013, I ran 5.312 @CharityMiles to help @Girl
current_hashtag = ""
hashtags = []
is_in_hashtag = False
for i in range(len(tweet)):
    if tweet[i] == " ": # found a space, so we've possible ended a hashtag
        if current_hashtag != "":
            hashtags.append(current_hashtag)
            current_hashtag = ""
        is_in_hashtag = False
    else:
        if tweet[i] == "#": # the start of a hashtag
            is_in_hashtag = True
        if is_in_hashtag == True:
            current_hashtag += tweet[i]
if current_hashtag != "":
    hashtags.append(current_hashtag)
```

REGULAR EXPRESSIONS TO THE RESCUE!

We can import Python's Regular Expression library via:

```
import re
```

Finding all Matched Patterns — with Regex!

findall() returns a list of all substrings that match the pattern

Finding all Matched Patterns — with Regex!


```
tweet = “RT @jleicole For #WHD2013, I ran …“
pattern = "#[^, ]+"
hashtags = re.findall(pattern, tweet)
```

Replacing Text

sentence = "Ms. Smith, are you okay?!? Please talk to me! Oh dear ..."

Imagine we want to replace all end-punctuation with a period, so that our text looks like:

sentence = “Ms. Smith, are you okay. Please talk to me. Oh dear.”

Replacing Text

sentence = "Ms. Smith, are you okay?!? Please talk to me! Oh dear ..."

This would normally be annoyingly tedious to write code for.

```
pattern = "[?!]+|\s*\.+"
sentence = re.sub(pattern, '.', sentence)
```

Replacing Text with Regex!

```
re.sub(pattern, replacement, text)
```

sub() replaces all matches in text with the replacement text

Regex Patterns

Patterns work by matching on:

- specific characters (e.g., ‘z’) or
- large categories of characters (e.g., all lowercased letters or all digits)

WORKED EXAMPLE:

“Code didn't work, no idea why…”

Specific Characters

```
text = “Code didn't work, no idea why…”
pattern = ‘a’
re.findall(pattern, text)
```

“Code didn't work, no idea why…”

Range of Specific Characters

```
text = “Code didn't work, no idea why…”
pattern = ‘[aeiouy]’
re.findall(pattern, text)
```

“Code didn't work, no idea why…”

```
['o', 'e', 'i', 'o', 'o', 'i', 'e', 'a', 'y']
```

The [ ] brackets denote “any of these characters”

Range of Specific Characters

```
text = “Code didn't work, no idea why…”
pattern = ‘[a-z]’
re.findall(pattern, text)
```

“Code didn't work, no idea why…”

```
[‘o', 'd', 'e', 'd', 'i', 'd', 'n', 't',
'w', 'o', 'r', 'k', 'n', 'o', 'i', 'd',
'e', 'a', 'w', 'h', 'y']
```

Range of Specific Characters

```
text = “Code didn't work, no idea why…”
pattern = ‘[a-zA-Z]’
re.findall(pattern, text)
```

“Code didn't work, no idea why…”

```
['C', 'o', 'd', 'e', 'd', 'i', 'd',
'n', 't', 'w', 'o', 'r', 'k', 'n',
'o', 'i', 'd', 'e', 'a', 'w', 'h',
'y']
```

Regex Patterns

Repeated Characters

```
text = “Code didn't work, no idea why…”
pattern = ‘[a-zA-Z]+’
re.findall(pattern, text)
```

“Code didn't work, no idea why…”

```
['Code', 'didn', 't', 'work', 'no', 'idea', 'why']
```

The + sign means 1 or more occurrences

Regex Patterns

Repeated Characters

```
text = “Code didn't work, no idea why…”
pattern = ‘[a-zA-Z]*’
re.findall(pattern, text)
```

“Code didn't work, no idea why…”

```
['Code', '', 'didn', '', 't', '', 'work', '', '', 'no',
'', 'idea', '', 'why', '', '', '', '']
```

The * sign means 0 or more occurrences

Repeated Characters

Instead of matching on 0 or more or 1 or more occurrences, you can also specify an exact number of occurrences N with {N}

Regex Patterns

N number of occurrences

```
text = “555-123-1234, 33-555-123-1234”
pattern = ‘\d{3}-\d{3}-\d{4}’
re.findall(pattern, text)
```

“555-123-1234, 33-555-123-1234”

\d{3} means exactly 3 single-digits in a row

```
['555-123-1234', '555-123-1234']
```

Regex Patterns

N number of occurrences

```
text = “555-123-1234, 33-555-123-1234”
pattern = ‘\d{1,3}-\d{3}-\d{3}-\d{4}’
re.findall(pattern, text)
```

What do you think this matches?

Regex Patterns

N number of occurrences

```
text = “555-123-1234, 33-555-123-1234”
pattern = ‘\d{1,3}-\d{3}-\d{3}-\d{4}’
re.findall(pattern, text)
```

“555-123-1234, 33-555-123-1234”

Special Characters

- \w - Any alphanumeric character and underscore, equivalent to [a-zA-Z0-9_]
- \s - Matches any whitespace (spaces, tabs, line breaks)
- \d - Matches any digit character, equivalent to [0-9]

Special Characters

Regular Expression Character Classes	
[ab-d]	One character of: a, b, c, d
[^ab-d]	One character except: a, b, c, d
[\b]	Backspace character
\d	One digit
\D	One non-digit
\s	One whitespace
\S	One non-whitespace
\w	One word character
\W	One non-word character

Regex Patterns

Regex Cheat Sheet

https://www.debuggex.com/cheatsheet/regex/python

Also, try out regular expressions in real-time:

https://pythex.org/