

Introduction to Data Science – Week 9

NETA LIVNEH

COURSE 55793

2019-2020

Until Now Recap

Midterm!

And before that:

- Reading and writing files
- Data types: int, float, string, list, tuple, dictionary, Boolean, set
- Control flows:: ifs, loops, functions
- Functions



This Week

Regular Expressions

Reading from an archive file

Midterm takeaways



Regular Expressions

Regular Expressions is a powerful method to search, replace, parse text, pattern extraction and validation

In data science, it is mostly used in the preprocessing stage to clean the text and extract features

Mostly, regular expressions makes the code more readable than the use of string functions like index, find, count, replace and split

Usually, it is also more efficient than python built in methods for searching in text



An Example: Parsing Phone Number

We need to deal with these kind of inputs

- 054-1234567
- 054 1234567
- 054-123-4567
- 02-1234567
- 02 - 1234567
- +972-541234567
- +972541234567
- 00972541234567
- +972-54-1234567



Country **area** **number**



How to use Regex

1. Import the regex module with `import re`
2. Create a Regex object with the `re.compile()` function (Remember to use a raw string)
3. Pass the string you want to search into the Regex object's `search()` method. This returns a Match object
4. Call the Match object's `group()` method to return a string of the actual matched text



Match vs. search vs. findall

The `match()` function checks for a match only at the beginning of the string (by default)

The `search()` function checks for a match anywhere in the string

The `findall()` function finds all possible matches in the entire sequence and returns them as a list of strings. Each returned string represents one match. If there is more than 1 group to match then it would return a list of tuples.

```
>>> import re
>>> phoneNumRegex = re.compile(r'(\d{3})-(\d{7})')
>>> print('Using search:', phoneNumRegex.search('My number is 054-6630664.').group())
Using search: 054-6630664
>>> print('Using match:', phoneNumRegex.match('My number is 054-6630664.'))
Using match: None
>>> print('Using findall:', phoneNumRegex.findall('My number is 054-6630664.'))
Using findall: [('054', '6630664')]
```



Search and Replace

```
result = re.sub('Gwendolen', '', earnest) # Delete pattern Gwendolen.  
result = re.sub('Gwendolen', 'Honey', earnest) # Replace pattern Gwendolen -> Honey.  
result = re.sub(r'\s+', ' ', earnest) # Eliminate duplicate whitespaces.  
result = re.sub(r'\s*[[\w\.\-!\s\']]+?\s*', '', earnest) # Delete play notes.
```

```
In [7]: print(earnest)  
Lady Bracknell. In the carriage, Gwendolen! [Gwendolen goes to the door. She and Jack blow kisses  
to each other behind Lady Bracknell's back. Lady Bracknell looks vaguely about as if she could not u  
nderstand what the noise was. Finally turns round.] Gwendolen, the carriage!  
Gwendolen. Yes, mamma. [Goes out, looking back at Jack.]  
Lady Bracknell. [Sitting down.] You can take a seat, Mr. Worthing.  
[Looks in her pocket for note-book and pencil.]  
Jack. Thank you, Lady Bracknell, I prefer standing.  
  
In [8]: print(re.sub(r'\s*[[\w\.\-!\s\']]+?\s*', '', earnest))  
Lady Bracknell. In the carriage, Gwendolen! Gwendolen, the carriage!  
Gwendolen. Yes, mamma.  
Lady Bracknell. You can take a seat, Mr. Worthing.  
  
Jack. Thank you, Lady Bracknell, I prefer standing.
```



Summarizing Regex Symbols

`?` matches zero or one of the preceding group.

`*` matches zero or more of the preceding group.

`+` matches one or more of the preceding group.

`{n,m}` matches at least `n` and at most `m` of the preceding group. The `{n}` matches exactly `n` of the preceding group.

`^` matches the start of the string

`$` matches the end of a string

`.` matches any character, except newline characters

`\d`, `\w`, and `\s` match a digit, word, or space character, respectively

Most character classes can be *negated* by capitalizing

`\D`, `\W`, and `\S` match anything except a digit, word, or space character, respectively.

`\b` matches a word boundary. `\B` matches a non word boundary

`(a|b|c)` matches either `a` or `b` or `c`. This is equivalent to `[abc]`.

`[^abc]` matches any character that isn't between the brackets.

`{n,m}?` or `*?` or `+?` performs a **nongreedy** match of the preceding group

(match) in general is a *remembered group*. You can get the value of what matched by using the `groups()` or `group()` methods



Regular Expressions functions

module/attribute	explanation
<code>re.compile(pattern)</code>	Compile a regular expression pattern into a regular expression object
<code>re.match(pattern, string)</code>	If zero or more characters at the beginning of <i>string</i> match the regular expression pattern, return a corresponding match object. Return <code>None</code> if the string doesn't match the pattern
<code>re.search(pattern, string)</code>	Scan through <i>string</i> looking for the first location where the regular expression pattern produces a match, and return a corresponding match object. Return <code>None</code> if no position in the string matches the pattern
<code>Re.findall(pattern, string)</code>	Return all non-overlapping matches of <i>pattern</i> in <i>string</i> , as a list of strings. If one or more groups are present in the pattern, return a list of groups
<code>re.sub(pattern, repl, string, count=0)</code>	Replaces all occurrences of the RE <i>pattern</i> in <i>string</i> with <i>repl</i> , substituting all occurrences unless <i>count</i> provided. It returns modified string.
<code>Match.group(num=0)</code>	Returns one or more subgroups of the match. 0 returns the whole match

`re.match(pattern, string)` is equivalent to `prog.match(string)`, where `prog = re.compile(pattern)`



Many Regex engines online

The screenshot displays the regex101.com website interface. The top navigation bar includes the site logo, social media links, and utility links like 'donate', 'contact', 'bug reports & feedback', and 'wiki'. The left sidebar contains sections for 'SAVE & SHARE', 'FLAVOR' (with options for PCRE, ECMAScript, Python, and Golang), 'TOOLS' (with a 'Code Generator' link), and a 'SPONSOR' section for 'Hotjar'. The main content area is divided into three horizontal panels: 'REGULAR EXPRESSION' (with a 'no match' status), 'TEST STRING' (containing the text 'regex101.com'), and 'SUBSTITUTION'. The right sidebar features an 'EXPLANATION' section, a 'MATCH INFORMATION' section, and a 'QUICK REFERENCE' section with a search bar and a list of token types (All Tokens, Common Tokens, General Tokens, Anchors).



*Some people, when confronted with a problem,
think "I know, I'll use regular expressions."*

Now they have two problems.

-- Jamie Zawinski, in comp.emacs.xemacs



Reading Compressed files

There are many compression algorithms so there are different Python implementations for each compression type. To name a few

- gzip – for gzip format files
- bz2 – support for bzip2 compression
- lzma - Compression using the LZMA algorithm
- zipfile – Works with zip archives

Why use python instead of opening the archive file?

- Works directly from archive - important when working with large files
- Can be a pain when you have many archive files
- Can read files line by line
- Can read sequentially
- Process several files in parallel



Reading from Archive: Zip

```
import datetime
import zipfile

def print_info(archive_name):

    if not zipfile.is_zipfile(archive_name): # Checks if the file is a Zipfile
        return ('not a zipfile')
    zf = zipfile.ZipFile(archive_name, 'r') # Read the Zipfile.

    for info in zf.infolist(): # Goes over the info object for every member of the archive.
        print(info.filename)
        print('\tComment:\t', info.comment)
        print('\tModified:\t', datetime.datetime(*info.date_time))
        print('\tSystem:\t\t', info.create_system, '(0 = Windows, 3 = Unix)')
        print('\tZIP version:\t', info.create_version)
        print('\tCompressed:\t', info.compress_size, 'bytes')
        print('\tUncompressed:\t', info.file_size, 'bytes', '\n')
```

```
title_tokens.txt
Comment:
Modified: 2016-07-21 10:28:52
System: 0 (0 = Windows, 3 = Unix)
ZIP version: 31
Compressed: 4602953576 bytes
Uncompressed: 13847328867 bytes
```

Process finished with exit code 0



Reading from Archive: Zip

```
import os, zipfile

def average_word_count(archive_name):
    with zipfile.ZipFile(archive_name) as zf:
        for filename in zf.namelist():
            if filename.endswith('.txt'):
                with zf.open(filename, 'r') as open_file: # opening file in archive to read line by line.
                                                            # To read the whole file use zf.read(filename)

                    count = 0
                    words = 0
                    for line in open_file:
                        count += 1
                        words += len(line.split())
                if count > 0:
                    print('file {0:s} has {1:d} lines with an average words count of {2:6.2f}'.format(repr(filename),
                                                                                                  count, words/count))
                else:
                    print('file {!r} is empty'.format(filename))
```

