Regular Expressions in Python Tutorial

Michelle Fullwood

PyLadies Meetup Aug 2013

Motivating regular expressions

Scenario: You're evaluating Acme Company's products for a hospital. You're given a text file containing purchase records from Acme. Looking through the text file, however, you see that purchases from other companies are included as well, with no mention of which ones come from which!

```
PCOD    QTY    DEPT      COST
A169    100    Micro     0.58
PDA1    1      Xray      600.00
X280    5      ER        199.99
...
```

Luckily, you know that Acme's product codes consist of one uppercase letter followed by three digits.

Motivating regular expressions

So you get to work with a script:

```
import string
for line in open('purchaserecords.txt','r'):
    if line[0] in string.uppercase and \
       line[1].isdigit() and \
       line[2].isdigit() and \
       line[3].isdigit():
        print line
    else:
        continue
```

It's a bit clunky, but it works.

Motivating regular expressions

The next step in your evaluation is to collate evaluations emailed to you by hospital staff. These were free text, so you need to extract the product codes from within them to know which evaluation refers to which product.

```
'...The gloves(P180) felt sticky...'
'...The X701 vacuum cleaner really sucked!...'
```

You might be able to think of ways to program this, but really...

It's time to bust out regular expressions.

Motivating regular expressions

What we want is a way to simply search for "one uppercase letter followed by three digits". We can do this using (1) a regular expression and (2) the `search` function provided by Python's `re` module:

```
import re

re.search(r'[A-Z]\d{3}', mystring)
```

Just what are regular expressions, anyway?

Regular expressions are strings that describe other sets of strings.

Some simple things we can do with regular expressions:

- Match sets of characters
  - Metacharacters `\w`, `\s`, `\d`
  - Character sets `[A-Z]`, `[AGCT]`, `[^AGCT]`
- Repeat things
  - A specific number of times `^[3,5]`, `?`
  - An unlimited number of times `*`,`+`

Plan for today

We'll learn:

- The pattern language for regular expressions
- The Python `re` functions that allow us to work with regexes

How to practise the code as we go along:

- iPython
  - Install iPython
  - Clone this repo from Github
  - Run `ipython notebook` from the command line
  - A browser window will automatically open
  - Select the only notebook
- Online
  - Enter regexes and strings into `http://www.pythonregex.com/`

Metacharacters

Metacharacters are pre-defined sets of characters.

- . matches ANY character*
- \w matches alphanumeric characters and underscore _
- \d matches digits 0 through 9
- \s matches whitespace characters
  - Spaces
  - Tabs \t
  - Newlines \n\r
  - (Escape sequences \f\v)

→ Exercise 1

Defining sets of characters

- List characters individually
  - `[AGCT]` matches one character A, G, C or T.
  - `[sd]` matches one whitespace character or digit
- Define a range of characters
  - `[A-T]` matches one character between A and T.
  - `[1-7]` matches one digit between 1 and 7.
  - Ranges as defined by ASCII or Unicode tables
  - You can combine ranges: `[a-cA-C]`

→ Exercise 2

Defining where a regular expression applies

We can say a regex has to be at the start or the end of the string, or at word boundaries, with more special characters.

- `^` – beginning of line
- `$` – end of line
- `\b` – word boundary

Examples: → Exercise 3

- `^Hallo$`
- `\bHallo\b`

Escaping characters

→ Exercise 4

Why didn't the regex(es) work? (Discuss.)

- When using characters that also have a special meaning, we have to escape them with a backslash
- `\^ \$ \. \\`
- Exception: within character sets [], metacharacters have their regular meaning.*

Defining complements of a set

Sometimes it's easier to define a set of characters as "everything other than X".

- `\S` – all non-whitespace characters
- `\W` – all non-alphanumeric characters (also excludes underscore)
- `\D` – all non-numeric characters
- `[^A-D]` – all characters other than A, B, C, D

→ Exercise 5