11.3: EXTRACTING DATA USING REGULAR EXPRESSIONS

Contributed by Chuck Severance
Clinical Associate Professor (School of Information) at University of Michigan

If we want to extract data from a string in Python we can use the `findall()` method to extract all of the substrings which match a regular expression. Let's use the example of wanting to extract anything that looks like an email address from any line regardless of format. For example, we want to pull the email addresses from each of the following lines:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
          for <source@collab.sakaiproject.org>;
Received: (from apache@localhost)
Author: stephen.marquard@uct.ac.za
```

We don't want to write code for each of the types of lines, splitting and slicing differently for each line. This following program uses `findall()` to find the lines with email addresses in them and extract one or more addresses from each of those lines.

CODE 11.3.1 (PYTHON):

```
%%python3

import re
s = 'A message from csev@umich.edu to cwen@iupui.edu about meeting @2PM'
lst = re.findall('\S+@\S+', s)
print(lst)

# Code: http://www.py4e.com/code3/re05.py
```

run restart

The `findall()` method searches the string in the second argument and returns a list of all of the strings that look like email addresses. We are using a two-character sequence that matches a non-whitespace character (`\S`).

The output of the program would be:

```
['csev@umich.edu', 'cwen@iupui.edu']
```

Translating the regular expression, we are looking for substrings that have at least one non-whitespace character, followed by an at-sign, followed by at least one more non-whitespace character. The "`\S+`" matches as many non-whitespace characters as possible.

The regular expression would match twice (csev@umich.edu and cwen@iupui.edu), but it would not match the string "@2PM" because there are no non-blank characters *before* the at-sign. We can use this regular expression in a program to read all the lines in a file and print out anything that looks like an email address as follows:

```
# Search for lines that have an at sign between characters
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('\S+@\S+', line)
    if len(x) > 0:
        print(x)

# Code: http://www.py4e.com/code3/re06.py
```

We read each line and then extract all the substrings that match our regular expression. Since `findall()` returns a list, we simply check if the number of elements in our returned list is more than zero to print only lines where we found at least one substring that looks like an email address.

If we run the program on `mbox.txt` we get the following output:

```
['wagnermr@iupui.edu']
['cwen@iupui.edu']
['<postmaster@collab.sakaiproject.org>']
['<200801032122.m03LMFo4005148@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
```

Some of our email addresses have incorrect characters like "`<`" or ";" at the beginning or end. Let's declare that we are only interested in the portion of the string that starts and ends with a letter or a number.

To do this, we use another feature of regular expressions. Square brackets are used to indicate a set of multiple acceptable characters we are willing to consider matching. In a sense, the "`\S`" is asking to match the set of "non-whitespace characters". Now we will be a little more explicit in terms of the characters we will match.

Here is our new regular expression:

```
[a-zA-Z0-9]\S*@\S*[a-zA-Z]
```

This is getting a little complicated and you can begin to see why regular expressions are their own little language unto themselves. Translating this regular expression, we are looking for substrings that start with a *single* lowercase letter, uppercase letter, or number "[a-zA-Z0-9]", followed by zero or more non-blank characters ("`\S*`"), followed by an at-sign, followed by zero or more non-blank characters ("`\S*`"), followed by an uppercase or lowercase letter. Note that we switched from "+" to "*" to indicate zero or more non-blank characters since "[a-zA-Z0-9]" is already one non-blank character. Remember that the "*" or "+" applies to the single character immediately to the left of the plus or asterisk.

If we use this expression in our program, our data is much cleaner:

```
# Search for lines that have an at sign between characters
# The characters must be a letter or number
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('[a-zA-Z0-9]\S+@\S+[a-zA-Z]', line)
    if len(x) > 0:
        print(x)

# Code: http://www.py4e.com/code3/re07.py
```

```
...
['wagnermr@iupui.edu']
['cwen@iupui.edu']
['postmaster@collab.sakaiproject.org']
['200801032122.m03LMFo4005148@nakamura.uits.iupui.edu']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
```

Notice that on the "source@collab.sakaiproject.org" lines, our regular expression eliminated two letters at the end of the string (" > ;"). This is because when we append "[a-zA-Z]" to the end of our regular expression, we are demanding that whatever string the regular expression parser finds must end with a letter. So when it sees the " > " after "sakaiproject.org > ;" it simply stops at the last "matching" letter it found (i.e., the "g" was the last good match).

Also note that the output of the program is a Python list that has a string as the single element in the list.