11.4: COMBINING SEARCHING AND EXTRACTING

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

If we want to find numbers on lines that start with the string "X-" such as:

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
```

we don't just want any floating-point numbers from any lines. We only want to extract numbers from lines that have the above syntax.

We can construct the following regular expression to select the lines:

```
^X-.*: [0-9.]+
```

Translating this, we are saying, we want lines that start with "X-", followed by zero or more characters (".*"), followed by a colon (":") and then a space. After the space we are looking for one or more characters that are either a digit (0-9) or a period "[0-9.]+". Note that inside the square brackets, the period matches an actual period (i.e., it is not a wildcard between the square brackets).

This is a very tight expression that will pretty much match only the lines we are interested in as follows:

```
# Search for lines that start with 'X' followed by any non
# whitespace characters and ':'
# followed by a space and any number.
# The number can include a decimal.
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^X\S*: [0-9.]+', line):
        print(line)

# Code: http://www.py4e.com/code3/re10.py
```

When we run the program, we see the data nicely filtered to show only the lines we are looking for.

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.6178
X-DSPAM-Probability: 0.0000
```

But now we have to solve the problem of extracting the numbers. While it would be simple enough to use `split`, we can use another feature of regular expressions to both search and parse the line at the same time.

Parentheses are another special character in regular expressions. When you add parentheses to a regular expression, they are ignored when matching the string. But when you are using `findall()`, parentheses indicate that while you want the whole expression to match, you only are interested in extracting a portion of the substring that matches the regular expression.

So we make the following change to our program:

```
# Search for lines that start with 'X' followed by any
# non whitespace characters and ':' followed by a space
# and any number. The number can include a decimal.
# Then print the number if it is greater than zero.
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^X\S*: ([0-9.]+)', line)
    if len(x) > 0:
        print(x)

# Code: http://www.py4e.com/code3/re11.py
```

Instead of calling `search()` , we add parentheses around the part of the regular expression that represents the floating-point number to indicate we only want `findall()` to give us back the floating-point number portion of the matching string.

The output from this program is as follows:

```
['0.8475']
['0.0000']
['0.6178']
['0.0000']
['0.6961']
['0.0000']
..
```

The numbers are still in a list and need to be converted from strings to floating point, but we have used the power of regular expressions to both search and extract the information we found interesting.

As another example of this technique, if you look at the file there are a number of lines of the form:

```
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

If we wanted to extract all of the revision numbers (the integer number at the end of these lines) using the same technique as above, we could write the following program:

```
# Search for lines that start with 'Details: rev='
# followed by numbers and '.'
# Then print the number if it is greater than zero
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^Details:.*rev=([0-9.]+)', line)
    if len(x) > 0:
        print(x)

# Code: http://www.py4e.com/code3/re12.py
```

Translating our regular expression, we are looking for lines that start with "Details:", followed by any number of characters (".*"), followed by "rev=", and then by one or more digits. We want to find lines that match the entire expression but we only want to extract the integer number at the end of the line, so we surround "[0-9]+" with parentheses.

When we run the program, we get the following output:

```
['39772']
['39771']
['39770']
['39769']
...
```

Remember that the "[0-9]+" is "greedy" and it tries to make as large a string of digits as possible before extracting those digits. This "greedy" behavior is why we get all five digits for each number. The regular expression library expands in both directions until it encounters a non-digit, or the beginning or the end of a line.

Now we can use regular expressions to redo an exercise from earlier in the book where we were interested in the time of day of each mail message. We looked for lines of the form:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

and wanted to extract the hour of the day for each line. Previously we did this with two calls to `split` . First the line was split into words and then we pulled out the fifth word and split it again on the colon character to pull out the two characters we were interested in.

While this worked, it actually results in pretty brittle code that is assuming the lines are nicely formatted. If you were to add enough error checking (or a big try/except block) to insure that your program never failed when presented with incorrectly formatted lines, the code would balloon to 10-15 lines of code that was pretty hard to read.

We can do this in a far simpler way with the following regular expression:

```
^From .* [0-9][0-9]:
```

The translation of this regular expression is that we are looking for lines that start with "From " (note the space), followed by any number of characters (".*"), followed by a space, followed by two digits "[0-9][0-9]", followed by a colon character. This is the definition of the kinds of lines we are looking for.

In order to pull out only the hour using `findall()` , we add parentheses around the two digits as follows:

```
^From .* ([0-9][0-9]):
```

This results in the following program:

```
# Search for lines that start with From and a character
# followed by a two digit number between 00 and 99 followed by ':'
# Then print the number if it is greater than zero
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    x = re.findall('^From .* ([0-9][0-9]):', line)
    if len(x) > 0: print(x)

# Code: http://www.py4e.com/code3/re13.py
```

When the program runs, it produces the following output:

```
['09']
['18']
['16']
['15']
...
```