Regular Expressions
Chapter 11
In computing, a regular expression, also referred to as “regex” or “regexp”, provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor.

http://en.wikipedia.org/wiki/Regular_expression
Regular Expressions

Really clever “wild card” expressions for matching and parsing strings

http://en.wikipedia.org/wiki/Regular_expression
Regular expression

In computing, a regular expression, also referred to as regex or regexp, provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. A regular expression is written in a formal language that can be interpreted by a regular expression processor, a program that either serves as a parser generator or examines text and identifies parts that match the provided specification.

The following examples illustrate a few specifications that could be expressed in a regular expression:

- The sequence of characters "car" appearing consecutively in any context, such as in "car", "cartoon", or "bicarbonate"
- The sequence of characters "car" occurring in that order with other characters between them, such as in "Icelander" or "chandler"
Understanding Regular Expressions

- Very powerful and quite cryptic
- Fun once you understand them
- Regular expressions are a language unto themselves
- A language of “marker characters” - programming with characters
- It is kind of an “old school” language - compact
WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!

BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!

IT'S HOPELESS!

EVERYBODY STAND BACK.

I KNOW REGULAR EXPRESSIONS.

http://xkcd.com/208/
Regular Expression Quick Guide

^ Matches the beginning of a line
$ Matches the end of the line
. Matches any character
\s Matches whitespace
\S Matches any non-whitespace character
* Repeats a character zero or more times
*? Repeats a character zero or more times (non-greedy)
+ Repeats a character one or more times
+? Repeats a character one or more times (non-greedy)
[aeiou] Matches a single character in the listed set
[^XYZ] Matches a single character not in the listed set
[a-zA-Z0-9] The set of characters can include a range
( Indicates where string extraction is to start
) Indicates where string extraction is to end
The Regular Expression Module

• Before you can use regular expressions in your program, you must import the library using "import re"

• You can use `re.search()` to see if a string matches a regular expression, similar to using the `find()` method for strings

• You can use `re.findall()` extract portions of a string that match your regular expression similar to a combination of `find()` and slicing: `var[5:10]`
Using `re.search()` like `find()`

```python
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line):
        print line
```

```python
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.find('From:') >= 0:
        print line
```
Using `re.search()` like `startswith()`

```python
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.startswith('From: '):
        print line
```

We fine-tune what is matched by adding special characters to the string

```python
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line):
        print line
```
Wild-Card Characters

- The dot character matches any character

- If you add the asterisk character, the character is “any number of times”
Wild-Card Characters

- The **dot** character matches any character

- If you add the **asterisk** character, the character is “any number of times”
Fine-Tuning Your Match

- Depending on how “clean” your data is and the purpose of your application, you may want to narrow your match down a bit.

**X-Sieve:** CMU Sieve 2.3
**X-DSPAM-Result:** Innocent
**X-Plane is behind schedule:** two weeks
Fine-Tuning Your Match

• Depending on how “clean” your data is and the purpose of your application, you may want to narrow your match down a bit.

X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-Plane is behind schedule: two weeks

Match the start of the line
One or more times
Match any non-whitespace character
Matching and Extracting Data

• The `re.search()` returns a True/False depending on whether the string matches the regular expression.

• If we actually want the matching strings to be extracted, we use `re.findall()`.

```python
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+',x)
>>> print y
['2', '19', '42']
```
Matching and Extracting Data

• When we use `re.findall()`, it returns a list of zero or more sub-strings that match the regular expression

```python
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+', x)
>>> print y
['2', '19', '42']
>>> y = re.findall('[AEIOU]+', x)
>>> print y
[]
```
Warning: **Greedy Matching**

- The `repeat` characters (`*` and `+`) push **outward** in both directions (greedy) to match the largest possible string.

```python
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+:', x)
>>> print y
['From: Using the : ']
```

Why not 'From:'?

- First character in the match is an `F`
- Last character in the match is a `:`
- One or more characters

```markdown
One or more characters
```
**Non-Greedy Matching**

- Not all regular expression repeat codes are greedy! If you add a `?` character, the `+` and `*` chill out a bit...

```python
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+?:', x)
>>> print y
['From: ']
```

- First character in the match is an F
- Last character in the match is a :
- One or more characters but not greedy
Fine-Tuning String Extraction

- You can refine the match for `re.findall()` and separately determine which portion of the match is to be extracted by using parentheses.

```python
>>> x = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008

>>> y = re.findall(r'\S+@\S+', x)
>>> print(y)
['stephen.marquard@uct.ac.za']
```

\S+@\S+  

At least one non-whitespace character
FINE-TUNING STRING EXTRACTION

- Parentheses are not part of the match - but they tell where to start and stop what string to extract

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

>>> y = re.findall('\S+@\S+',x)
>>> print y
['stephen.marquard@uct.ac.za']
>>> y = re.findall('^From:.+? (\S+@\S+)',x)
>>> print y
['stephen.marquard@uct.ac.za']
```
Extracting a host name - using find and string slicing

```python
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ', atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```
The Double Split Pattern

- Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

```
words = line.split()
email = words[1]
pieces = email.split('@')
print pieces[1]
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
'uct.ac.za'
```
import re

lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ \]*)', lin)
print y
['uct.ac.za']

Look through the string until you find an at sign
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([^ \])*',lin)
print y
['uct.ac.za']

'@([^ \])*'  
Match non-blank character  
Match many of them
The Regex Version

import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('@([\^ ]*)', lin)
print y

['uct.ac.za']

'@([\^ ]*)'

Extract the non-blank characters
Even Cooler Regex Version

Starting at the beginning of the line, look for the string 'From'

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ ]*)',lin)
print y
['uct.ac.za']
```
Even Cooler Regex Version

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

import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ \s]*)', lin)
print y
['uct.ac.za']

'^From .*@([^ \s]*)'
Even Cooler Regex Version

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ \*])', lin)
print y
['uct.ac.za']
```

'\^From .*@([^ ]*)'
Even Cooler Regex Version

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

import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ \s]*)', lin)
print y
['uct.ac.za']

'^From .*@([^ \s]*)'  
Match non-blank character  
Match many of them
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
y = re.findall('^From .*@([^ ]*)', lin)
print y
['uct.ac.za']
import re
hand = open('mbox-short.txt')
numlist = list()
for line in hand:
    line = line.rstrip()
    stuff = re.findall('^X-DSPAM-Confidence: ([0-9.]+)', line)
    if len(stuff) != 1 :  continue
    num = float(stuff[0])
    numlist.append(num)
print 'Maximum:', max(numlist)

X-DSPAM-Confidence: 0.8475

python ds.py
Maximum: 0.9907
If you want a special regular expression character to just behave *normally* (most of the time) you prefix it with '\'.

```python
>>> import re
>>> x = 'We just received $10.00 for cookies.'
>>> y = re.findall(r'\$[0-9.]+', x)
>>> print(y)
['$10.00']
```
Summary

- Regular expressions are a cryptic but powerful language for matching strings and extracting elements from those strings.
- Regular expressions have special characters that indicate intent.
These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translations here