Day 7: Regular Expressions

Suggested reading:

http://docs.python.org/release/2.4.3/lib/module-re.html

Turn In Homework

Homework Review

(necessarily brief today)

Patterns

Examples of Patterns

Dates	`2011-10-25  25 Oct 2011`
Telephone numbers	`608-262-4002`
Filenames	`CS368_2011-3_07_1115T.key`
Hostnames	`chopin.cs.wisc.edu`
Python comment lines	`# This is a comment`
Log file lines	`2011-09-28 09:51:15 startup ...`
List separators	`1, 2 , 3; 4 ,5,6;7`

A **regular expression** is
a **formal** description
of a **pattern**
that **partitions** all **strings**
into **matching** / **non-matching**

Typical Usage

- Test for a match
- Split a string into parts
- Extract part of a match
- Replace part of a match

Matching

x	***self***
^	start
$	end
.	any
\	escape
*	maybe some
+	some
?	maybe
{}	number
[]	class
\d	spec. classes
()	group
|	choice

Most characters match themselves:

letters digits ! @ # % & _ = ; : *etc.*

cat	
cat	*empty string*
a cat	**a**
catalog	**act**
scatter	**cart**
tomcat	**Cat**

# 1	
# 1	**#1**
# 12	**1 1**
### 1a	**###1**
.:# 111	**1#**

x	self
^	***start***
$	end
.	any
\	escape
*	maybe some
+	some
?	maybe
{}	number
[]	class
\d	spec. classes
()	group
|	choice

Matches at start of string; anchors rest of pattern to beginning

^cat	
cat	^cat
catalog	a cat
cathedral	scatter
cat's meow	␣cat
cat toy	tomcat

x	self
^	start
$	***end***
.	any
\	escape
*	maybe some
+	some
?	maybe
{}	number
[]	class
\d	spec. classes
()	group
|	choice

Matches at end of string; anchors rest of pattern to end

cat$	
cat	cat$
bobcat	cats
scat	scatter
\tcat	cat␣
nice cat	cat's

^cat$	
cat	*anything else*

x	self
^	start
$	end
.	***any***
\	escape
*	maybe some
+	some
?	maybe
{}	number
[]	class
\d	spec. classes
()	group
|	choice

Matches any *single* character: dot, whitespace, specials, anything

d.g	
dog	Dog
dig	drag
d.g	edge
mid-game	d. g
add2go	g.d

^d.$	
do	^d.$
di	Do
d!	ad
d2	dog
d␣	d..

x	self
^	start
$	end
.	any
\	***escape***
*	maybe some
+	some
?	maybe
{}	number
[]	class
\d	spec. classes
()	group
|	choice

Makes the following character match itself, no special meaning

1\.0	
1.0	1\.0
131.0.73.1	120
$21.03	1e0
...1.0...	10.1

2\^8	
2^8	2\^8

C:\\	
C:\Documents	c:\...
file:///C:\D	C:foo
C:\\	C:/

x	self
^	start
$	end
.	any
\	escape
*	***maybe some***
+	some
?	maybe
{}	number
[]	class
\d	spec. classes
()	group
|	choice

Match ***preceding element*** 0–*n* times; that is, "maybe some …"

an*y	
any	an*y
canyon	a
botany	an
granny	andy
days	an-y

a.*z	
azimuth	a
dazzle	z
waltz	apples
abuzz	buzz
a.*z	Abuzz

x	self
^	start
$	end
.	any
\	escape
*	maybe some
+	***some***
?	maybe
{}	number
[]	class
\d	spec. classes
()	group
|	choice

Match ***preceding element*** 1–*n* times; that is, "some …"

an+y	
any	an+y
canyon	days
botany	play
granny	Any
tannyl	a+y

a.+z	
dazzle	a
waltz	z
abuzz	azimuth
a2 - z2	apples
a.+z	buzz

x	self
^	start
$	end
.	any
\	escape
*	maybe some
+	some
?	***maybe***
{}	number
[]	class
\d	spec. classes
()	group
|	choice

Match ***preceding element*** 0–1 time; that is, "maybe …" or "optionally …"

an?y

any	an?y
canyon	ann
botany	andy
days	granny

=".+"**

```
var1="hi" var2="Tim"      a=""
var1="" var2="Alain"      quot="
```

=".+?"**

```
var1="hi" var2="Tim"      a=""
var1="" var2="Alain"      quot="
```

x	self
^	start
$	end
.	any
\	escape
*	maybe some
+	some
?	maybe
{}	***number***
[]	class
\d	spec. classes
()	group
|	choice

Match ***preceding element*** a number of times (n = 0, m = ∞ by default)

^a.{3,6}e$

above	ae
ashore	ate
achieve	able
airframe	manager

a.{3}e

above	able
apple	tables
at sea	manager
Yankees	airframe
transcend	a.{3}e

x	self
^	start
$	end
.	any
\	escape
*	maybe some
+	some
?	maybe
{}	number
[]	***class***
\d	spec. classes
()	group
|	choice

Match ***one of*** enclosed chars; most lose special meaning; – is for range

q[aeio]

Iraqi	q[aeio]
qanat	q
qintar	queue
qere	q?

:[0-5][0-9]

11:32 a.m.	1:60
page:08	2:3 ratio

^[^A-Za-z$.]+$

1,234,456	$42.00
\@/	11:32 am

x	self
^	start
$	end
.	any
\	escape
*	maybe some
+	some
?	maybe
{}	number
[]	class
\d	***spec. classes***
()	group
|	choice

Shortcuts for common character classes; use inside or outside of `[]`

\d	digits	**[0-9]**
\D	non-digits	**[^0-9]**
\w	"word" chars	**[a-zA-Z0-9_]**
\W	non-word chars	**[^a-zA-Z0-9_]**
\s	whitespace	**[\t\n\r\f\v]**
\S	non-whitespace	**[^ \t\n\r\f\v]**

^-?\d+$	
42	--1
-1	1a
1234	1e4
0	1.0

x	self
^	start
$	end
.	any
\	escape
*	maybe some
+	some
?	maybe
{}	number
[]	class
\d	spec. classes
()	***group***
|	choice

Groups and saves parts of a match; does not match any chars; can nest

^(pre)?te	
tend	steam
pretend	present
test	a test
pretest	^(pre)?te

(in){2}	
dining	ini
feminine	nine

^(.)(.)\2\1$	
anna	mama
^..^	dad

x	self
^	start
$	end
.	any
\	escape
*	maybe some
+	some
?	maybe
{}	number
[]	class
\d	spec. classes
()	group
|	***choice***

Matches one alternative of a set; applies to group or whole pattern

here|hear	
here	her
hear	haer
there	heer
heart	ear
gatherer	ere

d(og|im|ay)	
dog	dom
dim	diy
day	dig
dime	ayd
Tuesdays	d(og|im|ay)

breathe

Using Regular Expressions

Testing a Match (at Start)

```
re.match(r'regexp', string, flags)
```

- True *if and only if* matches at start of string
- Optional flags change behavior (see docs for all)

```
string = raw_input('Enter string: ')
regexp = raw_input('Enter regexp: ')
if re.match(regexp, string):
    print 'Match!'
```

```
if re.match(r'tim ', name, re.IGNORECASE):
    print 'Are you the instructor?'
```

Testing a Match (Anywhere)

```
re.search(r'regexp', string, flags)
```

- True if matches anywhere in string
- Optional flags change behavior (see docs for all)

```
import re

regexp = raw_input('Enter regexp: ')

wordfile = open('input-07-words.txt')
for line in wordfile:
    if re.search(regexp, line):
        print line.strip()
```

Splitting Strings

Split string using a string separator:

```
string.split(separator)
```

```
>>> 'a,b,c,d , e'.split(',')
['a', 'b', 'c', 'd ', ' e']
```

Split string using a regular expression separator:

```
re.split(separator, string)
```

```
>>> re.split(r'\s*,\s*', 'a,b,c,d , e')
['a', 'b', 'c', 'd', 'e']
```

Extracting Matches

```
matches = re.search(regexp, string, flags)
if matches is not None:
    whole = matches.group(0)
    group = matches.group(N)   # start @ 1
```

- Groups are numbered in order of left parentheses
- The **MatchObject** object has lots more info…

```
dt = raw_input('Date (YYYY-MM-DD)? ')

m = re.match('(\d{4})-(\d\d)-(\d\d)', dt)
if m is not None:
    (year, month, day) = m.group(1, 2, 3)
```

Replacing Matches

```
re.sub(regexp, replacement, string, count)
```

- Replaces ***all*** (or count) matches of ***regexp*** in ***string***
- Replacement can use groups with **\N** syntax

```
>>> s = 'Of France and England, ...'
>>> re.sub(r'and', r'or', s)
'Of France or Englor, ...'
>>> re.sub(r'and', r'or', s, 1)
'Of France or England, ...'

>>> n = '1234.5678'
>>> re.sub(r'\b(\d+)(\.\d+)?', r'\1', n)
'1234'
```

Now *you* can do this:

http://xkcd.com/208/

Last 2 Slides!

More Resources

- [Madcat] ***Mastering Regular Expressions***, Friedl

- Google for patterns
  - Can be very helpful
  - Do you trust what you find?
  - Understand assumptions, limitations, etc.
  - Use as inspiration, not as copy-and-paste solution

Homework

- Analyze a Condor log file
- Like counting word frequencies, except:
  - Only some lines (< 5%) contain our data
  - Only part of the line is interesting to us
  - Must modify info to be counted ***before*** counting it

```
#!/usr/bin/env python

"""Homework for CS 368-2 (2012 Spring)
Assigned on Day 07, 2012-04-10
Written by <Your Name>
"""
```