RegExpr:Review & Wrapup; List "Comprehensions"

Lecture 13b

Larry Ruzzo

(w/ thanks to Mary Kuhner for some slides)

Simple RegExpr Testing

```
>>> import re
>>> str1 = 'what foot or hand fell fastest'
>>> re.findall(r'f[a-z]*', str1)
['foot', 'fell', 'fastest']

>>> str2 = "I lack e's successor"
>>> re.findall(r'f[a-z]*',str2)
[]
```

Definitely recommend trying this with examples to follow, & more

Returns list of all matching substrings. (You don't need compile, etc.)

RegExpr Syntax

They're strings

Most punctuation is special; needs to be escaped (e.g., "\." instead of ".") to get non-special behavior

So, "raw" string literals (r'C:\new\.txt') are generally recommended for regexprs

- Unless you double your backslashes judiciously

RegExpr Semantics, I
Characters

RexExprs are patterns; they “match” sequences of characters

Letters, digits (& escaped punctuation like ‘\.’) match only themselves, just once

```
 r’TATAAT’      ‘ACGTTATAATGGTATAAT’
```

RegExpr Semantics, 2
Character Groups

Character groups [abc], [a-zA-Z], [^0-9] also match single characters, any of the characters in the group.

‘.’ similar – matches any letter (except newline)

Built-in shortcuts, e.g. `\s` ≡ `[ \n\t\r\f\v]`

```
r’T[AG]T[^GC].T’‘ACGTTGTAATGGTATnCT’
```

RegExpr Semantics, 3: Concatenation, Or, Grouping

You can group subexpressions with parens

If R, S are RegExprs, then

- RS matches the *concatenation* of strings matched by R, S individually
- R | S matches the *union*–either R or S

$\texttt{r'TAT(A.|.A)T'\ '}\underline{\texttt{TATCAT}}\texttt{G}\underline{\texttt{TATACT}}\texttt{CC}\overbrace{\texttt{TATCCT}}^{?}\texttt{'}$

RegExpr Semantics, 4 Repetition

If R is a RegExpr, then

R*	matches 0 or more consecutive strings (independently) matching R
R+	1 or more
R{n}	exactly n
R{m,n}	any number between m and n, inclusive
R?	0 or 1

Beware precedence (* > concat > |)

$$\texttt{r'TAT(A.|.A)*T'}\ \texttt{'}\underline{\texttt{TATCAT}}\texttt{G}\underline{\texttt{TATAC}}\overbrace{\underline{\texttt{TAT}}\texttt{CAC}}^{?}\underline{\texttt{TATT}}\texttt{'}$$

RegExprs in Python

By default

- Case sensitive, line-oriented (\n treated specially)
- Matching is generally "greedy"
  - Finds longest version of earliest starting match
  - Next "findall()" match will *not* overlap

$$\texttt{r".+\textbackslash.py"}\quad\texttt{"}\underbrace{\texttt{Two files: hw3.py and upper.py}}\texttt{."}$$

$$\texttt{r"\textbackslash w+\textbackslash.py"}\quad\texttt{"Two files: }\underbrace{\texttt{hw3.py}}\texttt{ and }\underbrace{\texttt{UPPER.py}}\texttt{."}$$

Python Mechanics

`re.match(pat, str)`
matches only at front of string

`re.search(pat,str)`
matches anywhere in string

Return "match" objects

`re.findall(pat,str)`
finds all (nonoverlapping) matches

Return lists of strings

Many others (split, substitute,...)

“Match” Objects

Retain info about exactly where the pattern matched, and how.

Of special note, if your pattern contains parenthesized groups, you can see what, if anything, matched each group, within the context of the overall match.

```
str= 'My birthdate is 09/03/1988'
pat = r'[bB]irth.* (\d{2})/(\d{2})/(\d{4})'
match = re.match(pat,str)
match.groups()
('09', '03', '1988')
```

“digit” ≡ [0-9]

Many more options; see Python docs...

Pattern Objects & "Compile"

Compile: assemble, e.g. a report, from various sources

```
mypat = re.compile(pattern[,flags])
```

Preprocess the pattern to make pattern matching fast. Useful if your code will do repeated searches with the same pattern. (Optional flags can modify defaults, e.g. case-sensitive matching, etc.)

Then use:

```
mypat.{match,search,findall,...}(string)
```

Exercise 1

Suppose "filenames" are upper or lower case letters or digits, starting with a letter, followed by a period (".") followed by a 3 character extension (again alphanumeric). Scan a list of lines or a file, and print all "filenames" in it, with*out* their extensions. Hint: use paren groups.

List “Comprehensions”

A common pattern–build one list from another:

```
list1 = [2,4,6]
list2 = []
for i in list1:
   list2.append(i**2)

print list2
[4, 16, 36]
```

Shorthand:

```
 list2 = [ i**2 for i in list1 ]
```

List “Comprehensions”, 2

More general form:

```
[expr(var) for var in iterable if cond(var)]
```

list, file, etc.

Example:

```
list2 = [i**2 for i in range(1,7) if i%2==0]
print list2
[4, 16, 36]
```

List “Comprehensions”, 3

Sometimes convenient, but never necessary

Can also iterate over multiple things, make nested lists, etc. (But can get rather hard to read, so restraint is also important,)

Practice

1. Make a list of powers of 2: $[2^0, 2^1, ..., 2^{10}]$
2. Make a list of lines from a file, each stripped of leading and trailing whitespace
3. Make a list of lists, each inner list being the list of uppercase equivalents of words from one line of a file

Solution 1

```
import sys
import re
filename = sys.argv[1]
filehandle = open(filename,"r")
filecontents = filehandle.read()
myrule = re.compile(
   r"([a-zA-Z][a-zA-Z0-9]*)\.[a-zA-Z0-9]{3}")
#Finds skidoo.bar amidst 23skidoo.barber; ok?
match = myrule.findall(filecontents)
print match
```

Practice Solutions

Solution 1:

```
[2**i for i in range(11)]
[1,2,4,8,16,32,64,128,256,512,1024]
```

Solution 2:

```
[line.strip() for line in file('mat.py')]
['a=[1,3,7]', 'mat=[]', 'for i in a:', ...]
```

Solution 3:

```
[line.split() for line in file('mat.py')]
[['a=[1,3,7]'], ['mat=[]'], ['for', 'i',
'in', 'a:'],...]
```