

# More Regular Expressions

Lecture 12b

Larry Ruzzo

(w/ thanks to Mary Kuhner for many slides)

# grep

Not part of Python (predates it by 20 years)

A useful utility in its own right, & a quick way to test some reg exp basics (but syntax is slightly different)

Just run it from the command line

```
% grep "re\.compile" *.py
```

# Strings Again

'abc'

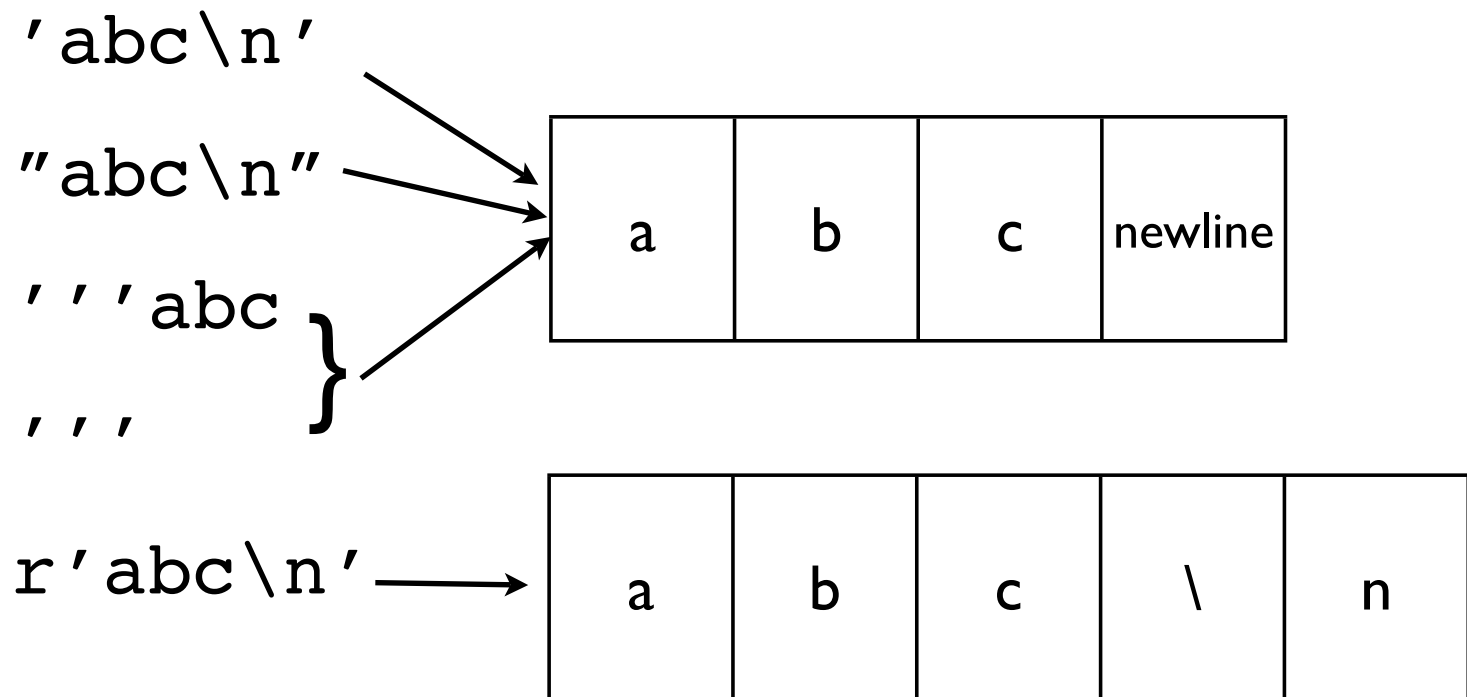
"abc"

'''abc'''

r'abc'

a	b	c
---	---	---

# Strings Again



# Why so many?

' vs " lets you put the other kind inside

' ' ' lets you run across many lines

all 3 let you include “invisible” characters

r'...' (raw strings) can't do invisible stuff, but avoid problems with backslash

`open('C:\new\text.dat')` vs

`open('C:\\new\\text.dat')` vs

`open(r'C:\new\text.dat')`

## Regular Expressions

---

- Regular expressions (regex) are a text-matching tool embedded in Python
- They are useful in creating string searches and string modifications
- You can always use regular Python instead, but regexps are often much easier
- Documentation: <http://docs.python.org/library/re.html>

## Basics of regexp construction

---

- Letters and numbers match themselves
- Normally case sensitive
- Watch out for punctuation—most of it has special meanings!

## Matching one of several alternatives

---

- Square brackets mean that any of the listed characters will do
- `[ab]` means either "a" or "b"
- You can also give a range:
- `[a-d]` means "a" "b" "c" or "d"
- Negation: caret means "not"

`[^a-d]`      # anything but a, b, c or d

## Wild cards

---

- "." means "any character"
- If you really mean "." you must use a backslash
- WARNING:
  - backslash is special in Python strings
  - It's special again in regexps
  - This means you need too many backslashes
  - We will use "raw strings" instead
  - Raw strings look like `r"ATCGGC"`

## Using . and backslash

---

- To match file names like "hw3.pdf" and "hw5.txt":

`hw.\....`

## Zero or more copies

---

- The asterisk repeats the previous character 0 or more times
- "ca\*t" matches "ct", "cat", "caat", "caaat" etc.
- The plus sign repeats the previous character 1 or more times
- "ca+t" matches "cat", "caat" etc. but not "ct"

## Repeats

---

- Braces are a more detailed way to indicate repeats
- $A\{1,3\}$  means at least one and no more than three A's
- $A\{4,4\}$  means exactly four A's

# simple testing

```
>>> import re
```

```
>>> string = 'what foot or hand fell fastest'
```

```
>>> re.findall(r'[a-z]*', string)  
['foot', 'fell', 'fastest']
```

## Practice problem 1

---

- Write a regexp that will match any string that starts with "hum" and ends with "001" with any number of characters, including none, in between
- (Hint: consider both "." and "\*")

## Practice problem 2

---

- Write a regexp that will match any Python (.py) file.
- There must be at least one character before the "."
- ".py" is not a legal Python file name
- (Imagine the problems if you imported it!)

## Using the regexp

---

First, compile it:

```
import re
myrule = re.compile(r".+\.py")
print myrule
<_sre.SRE_Pattern object at 0xb7e3e5c0>
```

The result of compile is a Pattern object which represents your regexp

## Using the regexp

---

Next, use it:

```
mymatch = myrule.search(myDNA)
print mymatch
None
mymatch = myrule.search(someotherDNA)
print mymatch
<_sre.SRE_Match object at 0xb7df9170>
```

The result of match is a Match object which represents the result.

## **All of these objects! What can they do?**

Functions offered by a Pattern object:

- `match()`—does it match the beginning of my string? Returns `None` or a match object
- `search()`—does it match anywhere in my string? Returns `None` or a match object
- `findall()`—does it match anywhere in my string? Returns a list of strings (or an empty list)
- Note that `findall()` does NOT return a Match object!

## **All of these objects! What can they do?**

---

Functions offered by a Match object:

- `group()`—return the string that matched
  - `group()`—the whole string
  - `group(1)`—the substring matching 1st parenthesized sub-pattern
  - `group(1,3)`—tuple of substrings matching 1st and 3rd parenthesized sub-patterns
- `start()`—return the starting position of the match
- `end()`—return the ending position of the match
- `span()`—return (start,end) as a tuple

## A practical example

---

Does this string contain a legal Python filename?

```
import re
myrule = re.compile(r".+\.py")
mystring = "This contains two files, hw3.py and uppercase.py."
mymatch = myrule.search(mystring)
print mymatch.group()
This contains two files, hw3.py and uppercase.py
# not what I expected! Why?
```

## Matching is greedy

---

- My regexp matches "hw3.py"
- Unfortunately it also matches "This contains two files, hw3.py"
- And it even matches "This contains two files, hw3.py and uppercase.py"
- Python will choose the longest match
- I could break my file into words first
- Or I could specify that no spaces are allowed in my match

## A practical example

---

Does this string contain a legal Python filename?

```
import re
myrule = re.compile(r"[^ ]+\.py")
mystring = "This contains two files, hw3.py and uppercase.py."
mymatch = myrule.search(mystring)
print mymatch.group()
hw3.py
allmymatches = myrule.findall(mystring)
print allmymatches
['hw3.py', 'uppercase.py']
```

## Practice problem 3

---

- Create a regexp which detects legal Microsoft Word file names
- The file name must end with ".doc" or ".DOC"
- There must be at least one character before the dot.
- We will assume there are no spaces in the names
- Print out a list of all the legal file names you find
- Test it on testre.txt (on the web site)

## Practice problem 4

---

- Create a regexp which detects legal Microsoft Word file names that do not contain any numerals (0 through 9)
- Print out the start location of the first such filename you encounter
- Test it on testre.txt

## Practice problem

---

- Create a regexp which detects legal Microsoft Word file names that do not contain any numerals (0 through 9)
- Print out the “base name”, i.e., the file name after stripping of the .doc extension, of each such filename you encounter. Hint: use parenthesized sub patterns.
- Test it on testre.txt

## Practice problem 1 solution

---

Write a regexp that will match any string that starts with "hum" and ends with "001" with any number of characters, including none, in between

J

## Practice problem 2 solution

---

Write a regexp that will match any Python (.py) file.

```
myrule = re.compile(r".+\.py")

# if you want to find filenames embedded in a bigger
# string, better is:
myrule = re.compile(r"[^ ]+\.py")
# this version does not allow whitespace in file names
```

## Practice problem 3 solution

---

Create a regexp which detects legal Microsoft Word file names, and use it to make a list of them

```
import sys
import re
filename = sys.argv[1]
filehandle = open(filename,"r")
filecontents = filehandle.read()
myrule = re.compile(r"[^ ]+\.[dD][oO][cC]")
matchlist = myrule.findall(filecontents)
print matchlist
```

## Practice problem 4 solution

---

Create a regexp which detects legal Microsoft Word file names which do not contain any numerals, and print the location of the first such filename you encounter

```
import sys
import re
filename = sys.argv[1]
filehandle = open(filename,"r")
filecontents = filehandle.read()
myrule = re.compile(r"[^ 0-9]+\.[dD][oO][cC]")
match = myrule.search(filecontents)
print match.start()
```

## Regular expressions summary

---

- The `re` module lets us use regular expressions
- These are fast ways to search for complicated strings
- They are not essential to using Python, but are very useful
- File format conversion uses them a lot
- Compiling a regexp produces a `Pattern` object which can then be used to search
- Searching produces a `Match` object which can then be asked for information about the match