BST281
Lab Session 9

Announcement:

- Problems3 was posted. Deadline is April 7th (This Friday)
- Today's lab focus on
  1. Regular expression

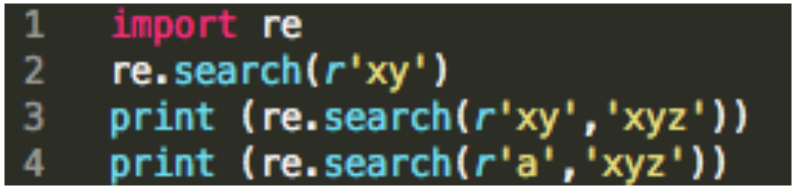
TA: Xue Zou   Email: xuz943@mail.harvard.edu

Regular Expression

- This section will include some guided exercises for learning about working with the re module in Python. We will begin with Python exercises in the interpreter.

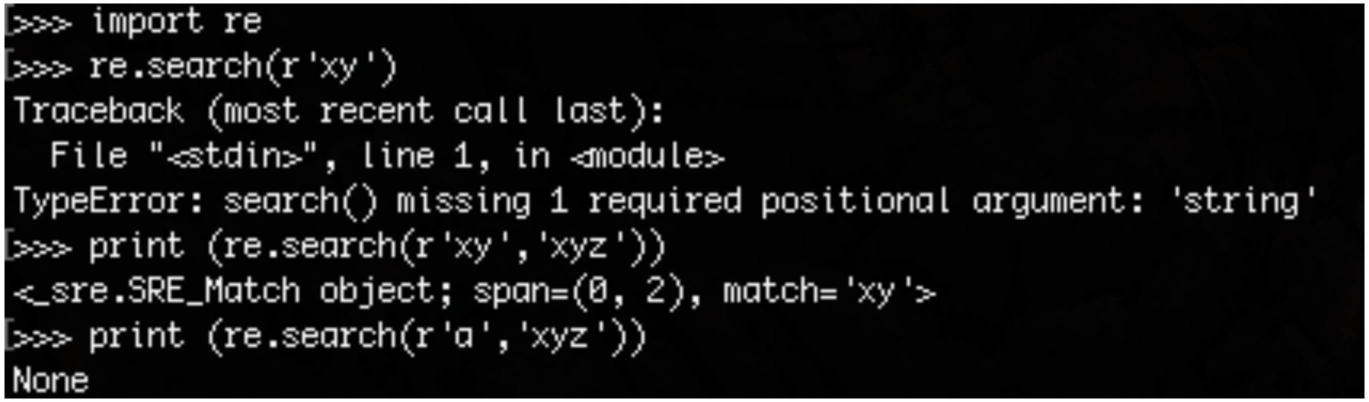
Exercise: re.search

- Type the following into the command line



```
import re
re.search(r'xy')
print (re.search(r'xy','xyz'))
print (re.search(r'a','xyz'))
```

- You should see the following output:



```
>>> import re
>>> re.search(r'xy')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: search() missing 1 required positional argument: 'string'
>>> print (re.search(r'xy','xyz'))
<_sre.SRE_Match object; span=(0, 2), match='xy'>
>>> print (re.search(r'a','xyz'))
None
```

re.search

1. How many and what kind of arguments does `re.search` take?

2. What does `re.search` return?

re.search

1. How many and what kind of arguments does `re.search` take?
   It takes two arguments: the first is a pattern to find, and the second is a string in which to search for the pattern.

2. What does `re.search` return?
   It returns a Match-Object if the pattern is found in the string, and None if it is not.

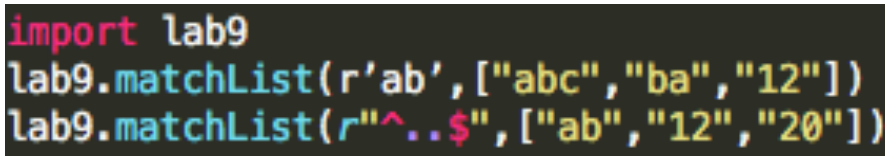
Practicing regular expression

We will write a simple function to tell us whether a pattern matches all, some or none of the ellements of a list. It is easiest to do this in a Python script and then import it into the interpreter. In a script named `lab04_re.py`, enter the following:

lab9.py

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

def matchList(pattern_str,list_astr):
    inList_ab = [bool( re.search( pattern_str, elt_str ) )
                   for elt_str in list_astr]

    numTrue_i = sum( [ int(elt_b) for elt_b in inList_ab ] )

    if   numTrue_i == len(list_astr): return( "all" )
    elif numTrue_i == 0:              return( "none" )

    return("some")
```

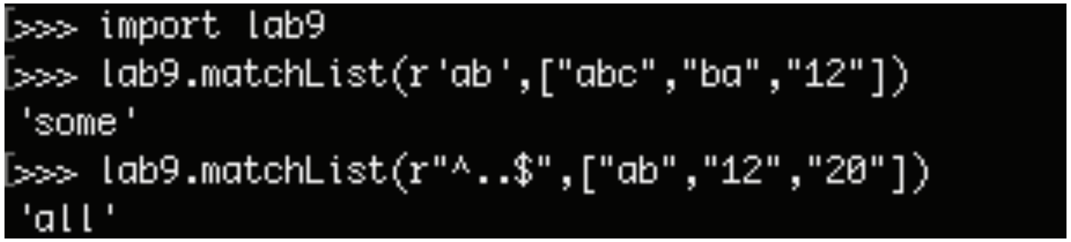
Save the file, then using the command line navigate to the same folder where the script is. Now open the interpreter from that folder and type the following:



```
import lab9
lab9.matchList(r'ab',["abc","ba","12"])
lab9.matchList(r"^..$",["ab","12","20"])
```

Questions:

- You should see the following output:



```
>>> import lab9
>>> lab9.matchList(r'ab',["abc","ba","12"])
'some'
>>> lab9.matchList(r"^..$",["ab","12","20"])
'all'
```

1. In English, what is the function **`matchList`** doing?

2. Why don't we need to use **`import re`** in the interpreter?

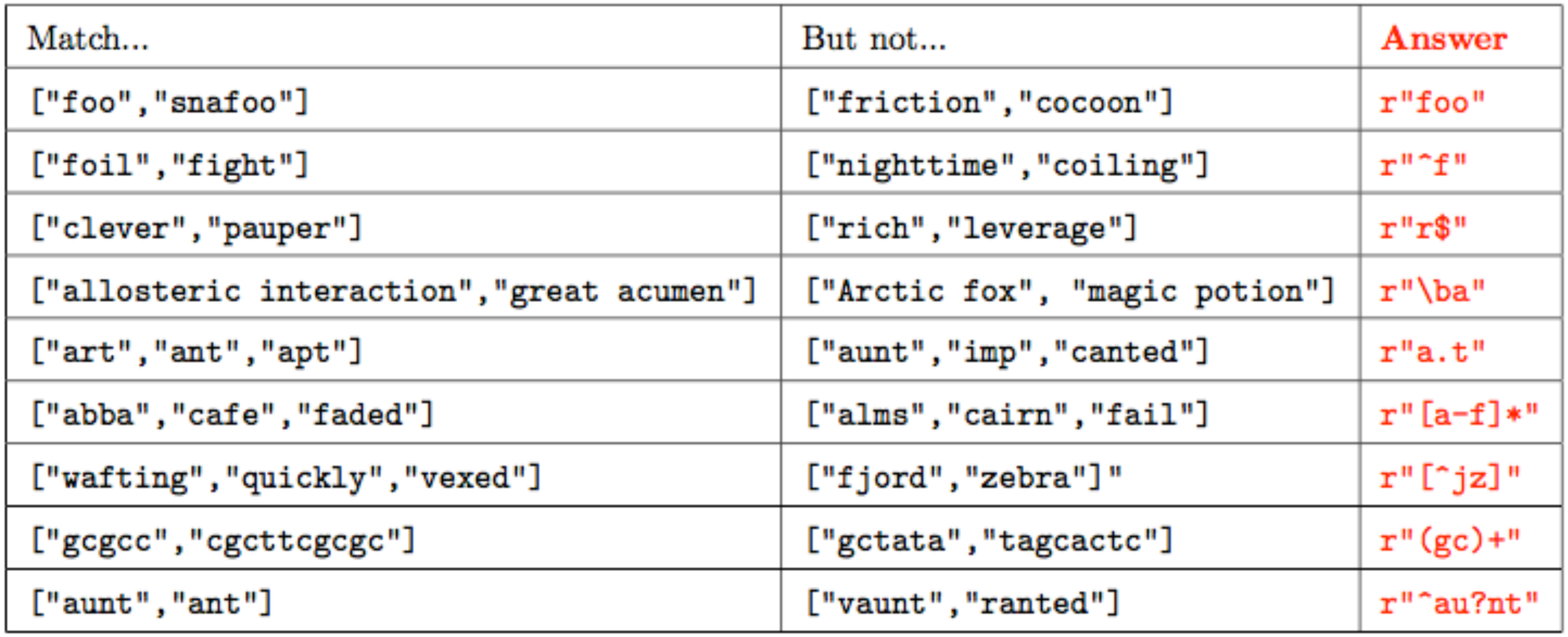
Answer:

1. In English, what is the function `matchList` doing?
   It first creates a list of boolean values which indicate whether the pattern was found in that element of the `list_astr`. Then it calculates the sum of the boolean values for the list (since `True` evaluates to 1 and `False` evaluates to 0). If the sum is the length of `list_astr`, then all the elements had the pattern; if the sum is zero, then none of the elements had the pattern; otherwise, some combination of elements had the pattern.

2. Why don't we need to use `import re` in the interpreter?
   Because our module `lab9` has the line `import re`, so we import the `re` module automatically when we import `lab9`.

More practices

- Using the lab9.matchList function, find regular expressions that distinguish between the following pairs of lists (note that these are rather contrived to force you to use the tools we talked about in class).



Match...	But not...	Answer
["foo","snafoo"]	["friction","cocoon"]	r"foo"
["foil","fight"]	["nighttime","coiling"]	r"^f"
["clever","pauper"]	["rich","leverage"]	r"r$"
["allosteric interaction","great acumen"]	["Arctic fox", "magic potion"]	r"\ba"
["art","ant","apt"]	["aunt","imp","canted"]	r"a.t"
["abba","cafe","faded"]	["alms","cairn","fail"]	r"[a-f]*"
["wafting","quickly","vexed"]	["fjord","zebra"]"	r"[^jz]"
["gcgcc","cgcttcgcgc"]	["gctata","tagcactc"]	r"(gc)+"
["aunt","ant"]	["vaunt","ranted"]	r"^au?nt"

Table 2: Some exercises for writing regular expressions

Capture groups

Now we will try and see what capture groups are all about. Open the interpreter (if it's not open already), and import the `re` module if you haven't already. Then type the following:

```
match = re.search(r'a(\d)b',"a1b")
match.groups()
match.group(0)
match.group(1)
match2 = re.search(r'(a\db)',"a1b")
match2.groups()
match2.group(0)
match2.group(1)
match_rep = re.search(r'a(\d)b',"a1ba2b")
match_rep.groups()
match_rep2 = re.search(r'(a\db)',"a1ba2b")
match_rep2.groups()
match_rep3 = re.search(r'(a\db)+',"a1ba2b")
match_rep3.groups()
match_rep4 = re.search(r'(a\db)*',"a1ba2b")
match_rep4.groups()
match_rep5 = re.search(r'(a\db)(a\db)',"a1ba2b")
match_rep5.groups()
```

You should see the following output:

```
>>> match = re.search(r'a(\d)b',"a1b")
>>> match.groups()
('1',)
>>> match.group(0)
'a1b'
>>> match.group(1)
'1'
>>> match2 = re.search(r'(a\db)',"a1b")
>>> match2.groups()
('a1b',)
>>> match2.group(0)
'a1b'
>>> match2.group(1)
'a1b'
>>> match_rep = re.search(r'a(\d)b',"a1ba2b")
>>> match_rep.groups()
('1',)
>>> match_rep2 = re.search(r'(a\db)',"a1ba2b")
>>> match_rep2.groups()
('a1b',)
>>> match_rep3 = re.search(r'(a\db)+',"a1ba2b")
>>> match_rep3.groups()
('a2b',)
>>> match_rep4 = re.search(r'(a\db)*',"a1ba2b")
>>> match_rep4.groups()
('a2b',)
>>> match_rep5 = re.search(r'(a\db)(a\db)',"a1ba2b")
>>> match_rep5.groups()
('a1b', 'a2b')
>>>
```

Questions:

1. What do the `.groups()` and `.group` functions do?

2. What is the 0th group?

3. Why do you think `match_rep2` returns only the first capture group?

4. Why do the + and * operators (`match_rep3` and `match_rep4`) still only return 1 group?

5. Why do the + and * operators (`match_rep3` and `match_rep4`) return `"a2b"` and not `"a1b"`? (HINT: look at the `re` module documentation for match objects.)

Answer:

1. What do the `.groups()` and `.group` functions do?
   `.groups()` returns a list of the captured groups; `.group` returns the group of the specified index.

2. What is the 0th group?
   The 0th group is always the entire string.

3. Why do you think `match_rep2` returns only the first capture group?
   search stops searching once it finds the desired pattern, so it only "sees" the first group.

4. Why do the + and * operators (`match_rep3` and `match_rep4`) still only return 1 group?
   Because a group is only defined by the parentheses; you will only get back as many groups as you have parenthetic groups.

5. Why do the + and * operators (`match_rep3` and `match_rep4`) return `"a2b"` and not `"a1b"`? (HINT: look at the `re` module documentation for match objects.)
   This is the default behavior: according to `http://docs.python.org/2/library/re.html`, "if a group matches multiple times, only the last match is accessible."