

## Re: OT: novice regular expression question

**Source:** <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2004-12/4965.html>

---

**From:** It's me (*itsme\_at\_yahoo.com*)

**Date:** 12/30/04

Date: Thu, 30 Dec 2004 19:39:11 GMT

I'll chew on this. Thanks, got to go.

"Steve Holden" <steve@holdenweb.com> wrote in message  
news:ujYAd.64260\$Jk5.22462@lakeread01...

> *It's me wrote:*

>

>> *I am never very good with regular expressions. My head always hurts  
>> whenever I need to use it.*

>>

> *Well, they are a pain to more than just you, and the conventional advice  
> is "even when you are convinced you need to use REs, try and find  
> another way".*

>

>> *I need to read a data file and parse each data record. Each item on the  
>> data record begins with either a string, or a list of strings. I*

searched

>> *around and didn't see any existing Python packages that does that.*

>> *scanf.py, for instance, can do standard items but doesn't know about  
list.*

>> *So, I figure I might have to write a lex engine for it and of course I  
have*

>> *to deal wit RE again.*

>>

> *Well, you haven't yet convinced me that you \*have\* to. Personally, I  
> think you just like trouble :-)*

>

>> *But I run into problem right from the start. To recognize a list, I  
need a*

>> *RE for the string:*

>>

>> *1) begin with [" (left bracket followed by a double quote with zero or  
more*

>> *spaces in between)*

>> *2) followed by any characters until ] but only if that left bracket is  
not*

>> *preceded by the escape character \.*

>>

> *So the pattern is*

>  
> 1. If the line begins with a "[" it should end with a "]"  
>  
> 2. Otherwise, it shouldn't?  
>  
> I'm trying to gently point out that the syntax you want to accept isn't  
> actually very clear. If the format is "Python strings and lists of  
> strings" then you might want to use the Python lexer to parse them, but  
> that's quite an advanced topic. [too advanced for me :-]  
>  
> The problem is matching "up to a right bracket not preceded by a  
> backslash". This seems to require what's technically referred to as a  
> "negative lookbehind assertion" – in other words, a pattern that doesn't  
> match anything, but checks that a specific condition is false or fails.  
>  
> > So, I tried:  
> >  
> > `^\[" "]*[a-zA-Z, ]*(\\|)*[a-zA-Z, \"]*`  
> >  
> > and tested with:  
> >  
> > `["This line\] works"]`  
> >  
> > but it fails with:  
> >  
> > `["This line fails"]`  
> >  
> > I would have thought that:  
> >  
> > `(\\|)*`  
> >  
> > should work because it's zero or more incidence of the pattern `\|`  
> >  
> > Any help is greatly appreciated.  
> >  
> > Sorry for beign OT. I posted this question at the lex group and didn't  
get  
> > any response. I figure may be somebody would know around here.  
>  
> I'd start with baby steps. First of all, make sure that you can match  
> the individual strings. Then use that pattern, parenthesized to turn it  
> into a group, as a component in a more complex pattern.  
>  
> Do you want to treat "this is also \" a string" as an allowable string?  
> In that case you need a pattern that matches 'up to the first quotation  
> mark not preceded by a backslash" as well!  
>  
> Let's try matching a single string first:  
>  
> >>> `s = re.compile(r'(".*?(?!\\)")')`  
> >>> `s.match("s1", "s2").groups()`

comp.lang.python: Re: OT: novice regular expression question

> ("sI",)  
>  
> *Note that I followed the "\*" with a "?" to stop it being greedy, and  
> matching as many characters as it could. OK, does that work when we have  
> escaped quotation marks?*  
>  
> >>> s.match(r"sI\\\"", "s2").groups()  
> ("sI\\\"",)  
>  
> *Apparently so. The negative lookbehind assertion stops a quote from  
> matching when it's preceded by a backslash. Can we match a  
> comma-separated list of such strings?*  
>  
> >>> slpat = r'(".\*?(?!\\)")(?:, (".\*?(?!\\)"))\*'  
> >>> s = re.compile(slpat)  
>  
> *This is a bit trickier: here the second grouping beginning with "(?:" is  
> intended to ensure that only the strings that get matched are included  
> in the groups, not the separators, even though they must be grouped  
> together. The list \*must\* be separated by ", ", but you could alter the  
> pattern to allow zero or more whitespace characters.*  
>  
> >>> s.match(r"sI\\\"", "s2").groups()  
> ("sI\\\"", "s2")  
>  
> *Well, that seems to work. Note that these patterns all ignore bracket  
> characters, so all you need to do now is to surround them with patterns  
> to match the opening and closing brackets, and you're done (I hope).*  
>  
> *Anyway, it'll give you a few ideas to work with.*  
>  
> regards  
> Steve  
> --  
> Steve Holden <http://www.holdenweb.com/>  
> Python Web Programming <http://pydish.holdenweb.com/>  
> Holden Web LLC +1 703 861 4237 +1 800 494 3119