How to implement a verbose REGEX in Python

Asked 9 years, 1 month ago Active 9 years, 1 month ago Viewed 7k times

14

6

I am trying to use a verbose regular expression in Python (2.7). If it matters I am just trying to make it easier to go back and more clearly understand the expression sometime in the future. Because I am new I first created a compact expression to make sure I was getting what I wanted.

Here is the compact expression:

```
test_verbose_item_pattern = re.compile('\n{1}\b?I[tT][eE][mM]\s+\d{1,2}\.?\(?[a-e]?
\)?.*[^0-9]\n{1}')
```

It works as expected

Here is the Verbose expression

```
verbose_item_pattern = re.compile("""
\n{1}       #begin with a new line allow only one new line character
\b?       #allow for a word boundary the ? allows 0 or 1 word boundaries \nITEM or \n
ITEM
I        # the first word on the line must begin with a capital I
[tT][eE][mM]  #then we need one character from each of the three sets this allows for
unknown case
\s+       # one or more white spaces this does allow for another \n not sure if I
should change it
\d{1,2}    # require one or two digits
\.?        # there could be 0 or 1 periods after the digits 1. or 1
\(?        # there might be 0 or 1 instance of an open paren
[a-e]?      # there could be 0 or 1 instance of a letter in the range a-e
\)?         # there could be 0 or 1 instance of a closing paren
.*          #any number of unknown characters so we can have words and punctuation
[^0-9]     # by its placement I am hoping that I am stating that I do not want to allow
strings that end with a number and then \n
\n{1}     #I want to cut it off at the next newline character
""",re.VERBOSE)
```

The problem is that when I run the verbose pattern I get an exception

```
Traceback (most recent call last):
File "C:/Users/Dropbox/directEDGAR-Code-Examples/NewItemIdentifier.py", line 17, in
```

figure it out. I did take my verbose expressions and compact it

me as the verbose.

asked Dec 13 '12 at 2:02

PyNEwbie
4,716 3 34 83

You need to escape your backslashes -- they aren't making it into the regex engine as they're converted during the normal Python string processing phase. It only worked incidentally in the compressed version because of this. Also, look up "raw string literals" :-)
– Cameron Dec 13 '12 at 2:07

@Cameron thanks I assumed that because I was using a triple quote I did not have to escape the slashes. I did as you said and now I get an unexpected end of pattern plus I saw examples where they did not escape the backslashes or use an r in front of the string
– PyNEwbie Dec 13 '12 at 2:25

2 Answers

Active Oldest Votes

18

- It is a good habit to use raw string literals when defining regex patterns. A lot of regex patterns use backslashes, and using a raw string literal will allow you to write single backslashes instead of having to worry about whether or not Python will interpret your backslash to have a different meaning (and having to use two backslashes in those cases).
- `\b?` is not valid regex. This is saying 0-or-1 word boundaries. But either you have a word boundary or you don't. If you have a word boundary, then you have 1 word boundary. If you don't have a word boundary then you have 0 word boundaries. So `\b?` would (if it were valid regex) be always true.
- Regex makes a distinction between the end of a string and the end of a line. (A string may consist of multiple lines.)
  - `\A` matches only the start of a string.
  - `\Z` matches only the end of a string.
  - `$` matches the end of a string, and also end of a line in re.MULTILINE mode.
  - `^` matches the start of a string, and also start of a line in re.MULTILINE mode.

```
import re
verbose_item_pattern = re.compile(r"""
    $            # end of line boundary
    \s{1,2}      # 1-or-2 whitespace character, including the newline
    I            # a capital I
    [tT][eE][mM] # one character from each of the three sets this allows for unknown
case
    \s+          # 1-or-more whitespaces INCLUDING newline
    \d{1,2}      # 1-or-2 digits
    [.]?         # 0-or-1 literal .
    \(?          # 0-or-1 literal open paren
    [a-e]?       # 0-or-1 letter in the range a-e
    \)?          # 0-or-1 closing paren
    .*           # any number of unknown characters so we can have words and
```

edited Dec 13 '12 at 3:03

answered Dec 13 '12 at 2:41
unutbu

753k 158 1660 1592

thanks for your answer. It helped and I have learned some useful things. I did not understand the $, now I do and I do not understand the MULTILINE flag the regex does not work without it (I am getting no matches) so it seems like something useful to poke at I appreciate your time – PyNEwbie Dec 13 '12 at 2:58

2

As say in the comment you should escape your backslash or use raw string even with triple quote.

```
verbose_item_pattern = re.compile(r"""
...
```

Share Improve this answer Follow

answered Dec 13 '12 at 2:57

Ghislain Hivon

131 5