Search and replace a line in a file in Python

I want to loop over the contents of a text file and do a search and replace on some lines and write the result back to the file. I could first load the whole file in memory and then write it back, but that probably is not the best way to do it.

What is the best way to do this, within the following code?

```
f = open(file)
for line in f:
    if line.contains('foo'):
        newline = line.replace('foo', 'bar')
        # how to write this newline back to the file
```

python file

edited Sep 7 '09 at 10:11
SilentGhost
77.8k 17 150 204

asked Sep 2 '08 at 9:19
pkit
2,209 1 14 19

add comment

12 Answers

I guess something like this should do it. It basically writes the content to a new file and replaces the old file with the new file:

```
from tempfile import mkstemp
from shutil import move
from os import remove, close

def replace(file_path, pattern, subst):
    #Create temp file
    fh, abs_path = mkstemp()
    new_file = open(abs_path,'w')
    old_file = open(file_path)
    for line in old_file:
        new_file.write(line.replace(pattern, subst))
    #close temp file
    new_file.close()
    close(fh)
    old_file.close()
    #Remove original file
    remove(file_path)
    #Move new file
```

```
    move(abs_path, file_path)
```

edited Jan 24 '13 at 22:37

answered Sep 2 '08 at 9:42

Thomas Watnedal
2,365 1 11 21

It doesn't look like the "fh" variable is being used. – Nathan Moinvaziri Jun 29 '12 at 20:42

1 It is used in the close call. – Thomas Watnedal Jul 26 '12 at 12:20

1 Just a minor comment: `file` is shadowing predefined class of the same name. – ezdazuzena Jan 24 '13 at 15:24

@ezdazuzena That's a good point. I've replaced file with file_path – Thomas Watnedal Jan 24 '13 at 22:38

1 This code changes the permissions on the original file. How can I keep the original permissions? – nic Jul 18 '13 at 21:35

add comment

The shortest way would probably be to use the fileinput module. For example, the following adds line numbers to a file, in-place:

```
import fileinput

for line in fileinput.input("test.txt", inplace=True):
    print "%d: %s" % (fileinput.filelineno(), line),
```

What happens here is:

1. The original file is moved to a backup file
2. The standard output is redirected to the original file within the loop
3. Thus any `print` statements write back into the original file

`fileinput` has more bells and whistles. For example, it can be used to automatically operate on all files in `sys.args[1:]`, without your having to iterate over them explicitly. Starting with Python 3.2 it also provides a convenient context manager for use in a `with` statement.

While `fileinput` is great for throwaway scripts, I would be wary of using it in real code because admittedly it's not very readable or familiar. In real (production) code it's worthwhile to spend just a few more lines of code to make the process explicit and thus make the code readable.

There are two options:

1. The file is not overly large, and you can just read it wholly to memory. Then close the file, reopen it in writing mode and write the modified contents back.
2. The file is too large to be stored in memory; you can move it over to a temporary file and open that, reading it line by line, writing back into the original file. Note that this requires twice the storage.

edited Oct 31 '13 at 13:24

answered Nov 14 '08 at 15:47

Eli Bendersky
81.6k 31 184 265

8 I know this only has two lines in it, however I don't think the code is very expressive in itself. Because if you think for a sec, if you didn't know the function, there are very few clues in what is going on. Printing the line number and the line is not the same as writing it ... if you get my gist... – chutsu May 29 '10 at 19:12

3 i agree. how would one use fileinput to write to the file? – jml Jan 24 '11 at 4:50

11 This **DOES** write to the file. It redirects stdout to the file. Have a look at the docs – brice Aug 24 '11 at 16:17

9 The key bit here is the comma at the end of the print statement: it suppresses the print statement adding

another newline (as line already has one). It's not very obvious at all, though (which is why Python 3 changed that syntax, luckily enough). – VPeric Oct 21 '11 at 14:24

2 Please notice this does not work when you provide an opening hook to the file, e.g. when you try to read/write UTF-16 encoded files. – bompf Jul 1 '13 at 12:19

show **3** more comments

Here's another example that was tested, and will match search & replace patterns:

```
import fileinput
import sys

def replaceAll(file,searchExp,replaceExp):
    for line in fileinput.input(file, inplace=1):
        if searchExp in line:
            line = line.replace(searchExp,replaceExp)
        sys.stdout.write(line)
```

Example use:

```
replaceAll("/fooBar.txt","Hello\sWorld!$","Goodbye\sWorld.")
```

edited Apr 27 '11 at 2:25

the Tin Man
67.1k 10 85 131

answered Nov 24 '08 at 19:02

Jason
1,829 17 22

10 The example use provides a regular expression, but neither `searchExp in line` nor `line.replace` are regular expression operations. Surely the example use is wrong. – kojiro Nov 14 '11 at 18:18

add comment

This should work: (inplace editiing)

```
import fileinput

for line in fileinput.input(files, inplace = 1): # Does a list of files, and w
      print line.replace("foo", "bar"),
```

edited Oct 24 '11 at 20:45

danodonovan
4,998 2 21 42

answered Sep 7 '09 at 10:07

Kinlan
7,700 1 24 49

1 I think it should be fileinput.input – DanJ May 25 '11 at 13:59

3 +1. Also if you receive a RuntimeError: input() already active then call the fileinput.close() – geographika Nov 18 '11 at 9:24

Note that `files` should be a string containing the file name, not a file object. – atomh33ls Aug 30 '13 at 10:00

add comment

Based on the answer by Thomas Watnedal. However, this does not answer the line-to-line part of the original question exactly. The function can still replace on a line-to-line basis

This implementation replaces the file contents without using temporary files, as a consequence file permissions remain unchanged.

Also re.sub instead of replace, allows regex replacement instead of plain text replacement only.

Reading the file as a single string instead of line by line allows for multiline match and replacement.

```
import re

def replace(file, pattern, subst):
```

```
def replace(file, pattern, subst):
    # Read contents from file as a single string
    file_handle = open(file, 'r')
    file_string = file_handle.read()
    file_handle.close()

    # Use RE package to allow for replacement (also allowing for (multiline) R
    file_string = (re.sub(pattern, subst, file_string))

    # Write contents to file.
    # Using mode 'w' truncates the file.
    file_handle = open(file, 'w')
    file_handle.write(file_string)
    file_handle.close()
```

edited Nov 30 '12 at 9:22

answered Nov 30 '12 at 8:51

Thijs

51 1 3

add comment

As lassevk suggests, write out the new file as you go, here is some example code:

```
fin = open("a.txt")
fout = open("b.txt", "wt")
for line in fin:
    fout.write( line.replace('foo', 'bar') )
fin.close()
fout.close()
```

answered Sep 2 '08 at 9:42

hamishmcn

3,466 6 24 34

add comment

A more pythonic way would be to use context managers like the code below:

```
from tempfile import mkstemp
from shutil import move
from os import remove
import sys

def replace(source_file_path, pattern, substring):
    fh, target_file_path = mkstemp()
    with open(target_file_path, 'w') as target_file:
        with open(source_file_path, 'r') as source_file:
            for line in source_file:
                target_file.write(line.replace(pattern, substring))
    remove(source_file_path)
    move(target_file_path, source_file_path)
```

You can find the full snippet here.

answered Sep 7 '13 at 18:39

Kiran

414 3 11

add comment

Create a new file, copy lines from the old to the new, and do the replacing before you write the lines to the new file.

answered Sep 2 '08 at 9:24


Lasse V. Karlsen
147k 39 312 494
add comment

Expanding on @Kiran's answer, which I agree is more succinct and Pythonic, this adds codecs to support the reading and writing of UTF-8:

```
import codecs

from tempfile import mkstemp
from shutil import move
from os import remove


def replace(source_file_path, pattern, substring):
    fh, target_file_path = mkstemp()

    with codecs.open(target_file_path, 'w', 'utf-8') as target_file:
        with codecs.open(source_file_path, 'r', 'utf-8') as source_file:
            for line in source_file:
                target_file.write(line.replace(pattern, substring))
    remove(source_file_path)
    move(target_file_path, source_file_path)
```

answered May 2 at 11:15

igniteflow
1,212 2 18 29
add comment

If you're wanting a generic function that replaces *any* text with some other text, this is likely the best way to go, particularly if you're a fan of regex's:

```
import re
def replace( filePath, text, subs, flags=0 ):
    with open( filePath, "r+" ) as file:
        fileContents = file.read()
        textPattern = re.compile( re.escape( text ), flags )
        fileContents = textPattern.sub( subs, fileContents )
        file.seek( 0 )
        file.truncate()
        file.write( fileContents )
```

answered Feb 18 at 14:43

starryknight64
101 12
add comment

Using hamishmcn's answer as a template I was able to search for a line in a file that match my regex and replacing it with empty string.

```
import re

fin = open("in.txt", 'r') # in file
fout = open("out.txt", 'w') # out file
for line in fin:
    p = re.compile('[-][0-9]*[.][0-9]*[,]|[-][0-9]*[,]') # pattern
    newline = p.sub('',line) # replace matching strings with empty string
    print newline
    fout.write(newline)
fin.close()
fout.close()
```

answered Apr 17 at 2:13

selectstar
1

add comment

---

if you remove the indent at the like below, it will search and replace in multiple line. See below for example.

```
def replace(file, pattern, subst):
    #Create temp file
    fh, abs_path = mkstemp()
    print fh, abs_path
    new_file = open(abs_path,'w')
    old_file = open(file)
    for line in old_file:
        new_file.write(line.replace(pattern, subst))
    #close temp file
    new_file.close()
    close(fh)
    old_file.close()
    #Remove original file
    remove(file)
    #Move new file
    move(abs_path, file)
```

edited Apr 8 '13 at 0:41

Rowan Thorpe
20 3

answered Aug 2 '12 at 19:12

loi
1

The formatting of this Python code doesn't look quite right... (I tried to fix, but wasn't sure what was intended) – Andy Hayden Sep 30 '12 at 18:18

add comment

Not the answer you're looking for? Browse other questions tagged python file or ask your own question.