Turn a string into a valid filename?

Asked 12 years, 10 months ago Active 3 months ago Viewed 208k times

352

82

I have a string that I want to use as a filename, so I want to remove all characters that wouldn't be allowed in filenames, using Python.

I'd rather be strict than otherwise, so let's say I want to retain only letters, digits, and a small set of other characters like `"_-.() "`. What's the most elegant solution?

The filename needs to be valid on multiple operating systems (Windows, Linux and Mac OS) - it's an MP3 file in my library with the song title as the filename, and is shared and backed up between 3 machines.

python filenames slug sanitize

Share Improve this question Follow

edited Nov 28 '16 at 2:18

martineau
104k 22 144 256

asked Nov 17 '08 at 9:02

Sophie Gage
4,542 4 22 21

25 Shouldn't this be built into the os.path module? – endolith Mar 10 '09 at 13:59

3 Perhaps, although her use case would require a single path that's safe across *all* platforms, not just the current one, which is something os.path isn't designed to handle. – javawizard Jun 3 '13 at 21:37

5 To expand on the above comment: the current design of `os.path` actually loads a different library depending on the os (see the second note in the documentation). So if a quoting function was implemented in `os.path` it could only quote the string for POSIX-safety when running on a POSIX system or for windows-safety when running on windows. The resulting filename would not necessarily be valid across both windows and POSIX, which is what the question asks for. – dshepherd Mar 10 '15 at 10:58

It's easy enough to use the `path` functions for a different OS. For example, on unix, use `import ntpath; ntpath.abspath("a.txt")` to get the absolute path of a file on a (hypothetical) Windows file system. Or use `posixpath` for posix systems (linux, Mac OS) – cowlinator Jul 23 at 9:17

26 Answers

Active | Oldest | Votes

```
import unicodedata
import re

def slugify(value, allow_unicode=False):
    """
    Taken from
https://github.com/django/django/blob/master/django/utils/text.py
    Convert to ASCII if 'allow_unicode' is False. Convert spaces or repeated
    dashes to single dashes. Remove characters that aren't alphanumerics,
    underscores, or hyphens. Convert to lowercase. Also strip leading and
    trailing whitespace, dashes, and underscores.
    """
    value = str(value)
    if allow_unicode:
        value = unicodedata.normalize('NFKC', value)
    else:
        value = unicodedata.normalize('NFKD', value).encode('ascii',
'ignore').decode('ascii')
    value = re.sub(r'[^\w\s-]', '', value.lower())
    return re.sub(r'[-\s]+', '-', value).strip('-_')
```

And the older version:

```
def slugify(value):
    """
    Normalizes string, converts to lowercase, removes non-alpha characters,
    and converts spaces to hyphens.
    """
    import unicodedata
    value = unicodedata.normalize('NFKD', value).encode('ascii', 'ignore')
    value = unicode(re.sub('[^\w\s-]', '', value).strip().lower())
    value = unicode(re.sub('[-\s]+', '-', value))
    # ...
    return value
```

There's more, but I left it out, since it doesn't address slugification, but escaping.

Share Improve this answer Follow

edited Jan 4 at 10:36

Roelant
3,396 1 19 51

answered Nov 17 '08 at 12:23

S.Lott
364k 75 490 762

11 The last line should be: value = unicode(re.sub('[-\s]+', '-', value)) – Joseph Turian Oct 8 '10 at 6:49

1 Thanks - I could be missing something, but I'm getting: "normalize() argument 2 must be unicode, not str" – Alex Cook Jan 12 '12 at 19:21

"normalize() argument 2". Means the `value` . If the value must be Unicode, then, you have to be sure that

133

You can use list comprehension together with the string methods.

```
>>> s
'foo-bar#baz?qux@127/\\9]'
>>> "".join(x for x in s if x.isalnum())
'foobarbazqux1279'
```

Share Improve this answer Follow

edited Oct 29 '12 at 9:59

John Mee

45k 31 135 174

answered Nov 17 '08 at 9:12

lutz

4 Note that you can omit the square brackets. In this case a generator expression is passed to join, which saves the step of creating an otherwise unused list. – Oben Sonne Oct 17 '11 at 11:25

45 +1 Loved this. Slight modification I've done: "".join([x if x.isalnum() else "_" for x in s]) -- would yield a result where invalid items are _, like they're blanked. Maybe tha thelps someone else. – Eddie Parker Dec 6 '12 at 22:44

18 This solution is great! I made a slight modification though: `filename = "".join(i for i in s if i not in "\/:*?<>|")` – Alex Krycek Jun 23 '13 at 23:17

3 Unfortunately it doesn't even allow spaces and dots, but I like the idea. – tiktak Jul 11 '13 at 10:40

18 @tiktak: to (also) allow spaces, dots and underscores you can go for `"".join( x for x in s if (x.isalnum() or x in "._- "))` – hardmooth May 9 '16 at 6:32

107

This whitelist approach (ie, allowing only the chars present in valid_chars) will work if there aren't limits on the formatting of the files or combination of valid chars that are illegal (like ".."), for example, what you say would allow a filename named " . txt" which I think is not valid on Windows. As this is the most simple approach I'd try to remove whitespace from the valid_chars and prepend a known valid string in case of error, any other approach will have to know about what is allowed where to cope with Windows file naming limitations and thus be a lot more complex.

```
>>> import string
>>> valid_chars = "-_.() %s%s" % (string.ascii_letters, string.digits)
>>> valid_chars
'-_.() abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
>>> filename = "This Is a (valid) - filename%$&$ .txt"
>>> ''.join(c for c in filename if c in valid_chars)
```

8 `valid_chars = frozenset(valid_chars)` wouldn't hurt. It is 1.5 times faster if applied to allchars. – jfs Nov 17 '08 at 11:14

2 Warning: This maps two different strings to the same string >>> import string >>> valid_chars = "-.() %s%s" % (string.ascii_letters, string.digits) >>> valid_chars '-.() abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789' >>> filename = "a.com/hello/world" >>> ''.join(c for c in filename if c in valid_chars) 'a.comhelloworld' >>> filename = "a.com/helloworld" >>> ''.join(c for c in filename if c in valid_chars) 'a.comhelloworld' >>> – robert king May 16 '12 at 0:53

3 Not to mention that naming a file `"CON"` on Windows will get you into trouble... – Nathan Osman Jul 5 '12 at 4:09

2 A slight rearrangement makes specifying a substitute character straightforward. First the original functionality: ''.join(c if c in valid_chars else '' for c in filename) or with a substituted character or string for every invalid character: ''.join(c if c in valid_chars else '.' for c in filename) – PeterVermont Feb 10 '14 at 19:17

Minor point,".txt" is a valid filename on Windows, though I suspect there are lot more ".gitignore" files out there. – Pat Corwin Sep 7 '20 at 4:57

102

What is the reason to use the strings as file names? If human readability is not a factor I would go with base64 module which can produce file system safe strings. It won't be readable but you won't have to deal with collisions and it is reversible.

```
import base64
file_name_string = base64.urlsafe_b64encode(your_string)
```

Update: Changed based on Matthew comment.

Share Improve this answer Follow

edited Apr 13 '09 at 16:48

answered Nov 17 '08 at 9:12

Igal Serban
10.2k 3 32 36

64 Warning! base64 encoding by default includes the "/" character as valid output which isn't valid in filenames on a lot of systems. Instead use base64.urlsafe_b64encode(your_string) – Matthew Apr 12 '09 at 1:33

1 This should *absolutely* be regarded as the ideal answer for webservers with any internal user-named content. Even if the administrator needs to go find something, you can easily write a script to transform all queries to the same form. – codetaku Oct 18 '13 at 20:49

21 Actually human readability is almost always a factor, even if only for debugging purposes. – static_rtti Jun

43

Just to further complicate things, you are not guaranteed to get a valid filename just by removing invalid characters. Since allowed characters differ on different filenames, a conservative approach could end up turning a valid name into an invalid one. You may want to add special handling for the cases where:

- The string is all invalid characters (leaving you with an empty string)
- You end up with a string with a special meaning, eg "." or ".."
- On windows, certain device names are reserved. For instance, you can't create a file named "nul", "nul.txt" (or nul.anything in fact) The reserved names are:

  CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9

You can probably work around these issues by prepending some string to the filenames that can never result in one of these cases, and stripping invalid characters.

Share Improve this answer Follow

answered Nov 17 '08 at 9:57

Brian
110k 28 105 109

36

There is a nice project on Github called python-slugify:

Install:

```
pip install python-slugify
```

Then use:

```
>>> from slugify import slugify
>>> txt = "This\ is/ a%#$ test ---"
>>> slugify(txt)
'this-is-a-test'
```

Share Improve this answer Follow

answered Apr 29 '15 at 11:19

Shoham
5,682 8 37 40

a valid filename.

36

The most recent and updated version is found in utils/text.py, and defines "get_valid_filename", which is as follows:

```
def get_valid_filename(s):
    s = str(s).strip().replace(' ', '_')
    return re.sub(r'(?u)[^-\w.]', '', s)
```

(See https://github.com/django/django/blob/master/django/utils/text.py)

Share Improve this answer Follow

answered Oct 18 '17 at 0:24

cowlinator
4,542 4 28 42

4 for the lazy already on django: `django.utils.text import get_valid_filename` – theannouncer Mar 2 '18 at 19:59

2 In case you are unfamiliar with regex, `re.sub(r'(?u)[^-\w.]', '', s)` removes all characters which are not letters, not numbers (0-9), not the underscore ('_'), not the dash ('-'), and not the period ('.'). "Letters" here includes all unicode letters, such as 漢語. – cowlinator May 4 '18 at 21:41

4 You may want to also check for length: Filenames are limited to 255 characters (or, you know, 32; depending on the FS) – Matthias Winkelmann Nov 13 '18 at 12:50

`return re.sub(r'(?u)[^-\w.]', '_', s)` for better readability – spiralmoon Jul 5 '20 at 18:21

21

This is the solution I ultimately used:

```
import unicodedata

validFilenameChars = "-_.() %s%s" % (string.ascii_letters, string.digits)

def removeDisallowedFilenameChars(filename):
    cleanedFilename = unicodedata.normalize('NFKD', filename).encode('ASCII',
'ignore')
    return ''.join(c for c in cleanedFilename if c in validFilenameChars)
```

The unicodedata.normalize call replaces accented characters with the unaccented equivalent, which is better than simply stripping them out. After that all disallowed characters are removed.

My solution doesn't prepend a known string to avoid possible disallowed filenames, because I know they can't occur given my particular filename format. A more general solution would need

8 camel case .. ahh – demented hedgehog Oct 17 '16 at 3:54

Could this be edited / updated to work with Python 3.6 ? – Wavesailor Mar 2 '18 at 15:23

16

In one line:

```
valid_file_name = re.sub('[^\w_.)( -]', '', any_string)
```

you can also put '_' character to make it more readable (in case of replacing slashs, for example)

Share Improve this answer Follow

answered Aug 4 '16 at 11:29

mnach

452 3 9

15

Keep in mind, there are actually no restrictions on filenames on Unix systems other than

- It may not contain \0
- It may not contain /

Everything else is fair game.

```
$ touch "
> even multiline
> haha
> ^[[31m red ^[[0m
> evil"
$ ls -la
-rw-r--r--       0 Nov 17 23:39 ?even multiline?haha??[31m red ?[0m?evil
$ ls -lab
-rw-r--r--       0 Nov 17 23:39 \neven\ multiline\nhaha\n\033[31m\ red\ \033[0m\nevil
$ perl -e 'for my $i ( glob(q{./*even*}) ){ print $i; } '
./
even multiline
haha
 red
evil
```

Yes, i just stored ANSI Colour Codes in a file name and had them take effect.

For entertainment, put a BEL character in a directory name and watch the fun that ensues when

20:18

1 @cowlinator that clarification was added 10 hours after my answer was posted :) Check the OP's edit log. – Kent Fredric May 16 '19 at 1:08

7

You could use the re.sub() method to replace anything not "filelike". But in effect, every character could be valid; so there are no prebuilt functions (I believe), to get it done.

```
import re

str = "File!name?.txt"
f = open(os.path.join("/tmp", re.sub('[^-a-zA-Z0-9_.() ]+', '', str))
```

Would result in a filehandle to /tmp/filename.txt.

Share Improve this answer Follow

edited Jul 1 '15 at 9:35

Cees Timmerman
14k 9 83 112

answered Nov 17 '08 at 9:10

gx.
393 1 8

5 You need the dash to go first in the group matcher so it doesn't appear as a range. re.sub('[^-a-zA-Z0-9_.()]+', '', str) – phord Nov 16 '10 at 0:09

7

```
>>> import string
>>> safechars = bytearray(('_-.()' + string.digits +
string.ascii_letters).encode())
>>> allchars = bytearray(range(0x100))
>>> deletechars = bytearray(set(allchars) - set(safechars))
>>> filename = u'#ab\xa0c.$%.txt'
>>> safe_filename = filename.encode('ascii', 'ignore').translate(None,
deletechars).decode()
>>> safe_filename
'abc..txt'
```

It doesn't handle empty strings, special filenames ('nul', 'con', etc).

Share Improve this answer Follow

edited Jun 22 '17 at 11:26

answered Nov 17 '08 at 10:15

jfs
356k 157 889 1545

I'm also worried about the blacklist. Granted, it's a blacklist that's based off a whitelist, but still. It seems

less... safe. How do you know that "allchars" is actually complete? – isaaclw Apr 24 '12 at 18:23

@isaaclw: '.translate()' accepts 256-char string as a translation table (byte-to-byte translation). '.maketrans()' creates such string. All values are covered; it is a pure whitelist approach – jfs Apr 25 '12 at 7:48

What about the filename '.' (a single dot). That would not work on Unixes as the present directory is using that name. – Finn Årup Nielsen Feb 4 '15 at 16:26

6

Another issue that the other comments haven't addressed yet is the empty string, which is obviously not a valid filename. You can also end up with an empty string from stripping too many characters.

What with the Windows reserved filenames and issues with dots, the safest answer to the question “how do I normalise a valid filename from arbitrary user input?” is “don't even bother try”: if you can find any other way to avoid it (eg. using integer primary keys from a database as filenames), do that.

If you must, and you really need to allow spaces and ‘.’ for file extensions as part of the name, try something like:

```
import re
badchars= re.compile(r'[^A-Za-z0-9_. ]+|^\.|\.$|^ | $|^$')
badnames= re.compile(r'(aux|com[1-9]|con|lpt[1-9]|prn)(\.|$)')

def makeName(s):
    name= badchars.sub('_', s)
    if badnames.match(name):
        name= '_'+name
    return name
```

Even this can't be guaranteed right especially on unexpected OSs — for example RISC OS hates spaces and uses ‘.’ as a directory separator.

Share Improve this answer Follow

answered Nov 17 '08 at 13:24

bobince
504k 102 630 810

6

Though you have to be careful. It is not clearly said in your intro, if you are looking only at latine language. Some words can become meaningless or another meaning if you sanitize them with ascii characters only.

"下北沢" your system might end up doing "---" which is doomed to fail after a while and not very helpful. So if you deal with only files I would encourage to either call them a generic chain that you control or to keep the characters as it is. For URIs, about the same.

Share Improve this answer Follow

answered Mar 11 '09 at 10:44

karlcow
6,809 4 34 69

6

Why not just wrap the "osopen" with a try/except and let the underlying OS sort out whether the file is valid?

This seems like much less work and is valid no matter which OS you use.

Share Improve this answer Follow

edited May 30 '12 at 1:46

dplante
2,387 3 20 26

answered Nov 17 '08 at 11:24

James Anderson
26.5k 7 47 76

5 Does it valid the name though? I mean, if the OS is not happy, then you still need to do something, right? – jeromej Feb 27 '14 at 1:33

1 In some cases, the OS/Language may silently munge your filename into an alternative form, but when you do a directory listing, you'll get a different name out. And this can lead to a "when I write the file its there, but when I look for the file its called something else" problem. (I'm talking about behaviour I've heard about on VAX ...) – Kent Fredric Apr 5 '16 at 10:22

Moreover, "The filename needs to be valid on multiple operating systems", which you can't detect with an osopen running on one machine. – LarsH Jun 4 '19 at 12:15

5

I liked the python-slugify approach here but it was stripping dots also away which was not desired. So I optimized it for uploading a clean filename to s3 this way:

```
pip install python-slugify
```

Example code:

```
s = 'Very / Unsafe / file\nname hähä \n\r .txt'
clean_basename = slugify(os.path.splitext(s)[0])
clean_extension = slugify(os.path.splitext(s)[1][1:])
```

```
>>> clean_filename
'very-unsafe-file-name-haha.txt'
```

This is so failsafe, it works with filenames without extension and it even works for only unsafe characters file names (result is `none` here).

Share Improve this answer Follow edited Oct 5 '17 at 16:51 answered Oct 5 '17 at 16:36

therealmarv
3,572 4 21 41

1 I like this, don't reinvent the wheel, don't import the whole Django framework if you don't need it, don't directly paste the code if you are not going to maintain it in the future, and generated string tries to matches similar letters to safe ones, so new string is easier to read. – vicenteherrera Mar 23 '20 at 12:15

1 To use underscore instead of dash: name=slugify(s, separator='_') – vicenteherrera Mar 23 '20 at 12:19

5

I realise there are many answers but they mostly rely on regular expressions or external modules, so I'd like to throw in my own answer. A pure python function, no external module needed, no regular expression used. My approach is not to clean invalid chars, but to only allow valid ones.

```
def normalizefilename(fn):
    validchars = "-_.() "
    out = ""
    for c in fn:
      if str.isalpha(c) or str.isdigit(c) or (c in validchars):
        out += c
      else:
        out += "_"
    return out
```

if you like, you can add your own valid chars to the `validchars` variable at the beginning, such as your national letters that don't exist in English alphabet. This is something you may or may not want: some file systems that don't run on UTF-8 might still have problems with non-ASCII chars.

This function is to test for a single file name validity, so it will replace path separators with _ considering them invalid chars. If you want to add that, it is trivial to modify the `if` to include os path separator.

Share Improve this answer Follow answered Mar 11 '19 at 12:21


```
import unicodedata
```

```
validFilenameChars = "-_.() %s%s" % (string.ascii_letters, string.digits)
def removeDisallowedFilenameChars(filename):
    cleanedFilename = unicodedata.normalize('NFKD', filename).encode('ASCII',
'ignore')
    return ''.join(chr(c) for c in cleanedFilename if chr(c) in
validFilenameChars)
```

Share Improve this answer Follow

edited Sep 4 '19 at 7:02
Mark Warburton
391 7 16

answered Apr 22 '19 at 4:48
Jean-Robin Tremblay
109 1 2

Could you explain your answer in details? – Serenity Apr 22 '19 at 5:07

Its the same answer accepted by Sophie Gage. But it has been modified to work on python 3.6 – Jean-Robin Tremblay Apr 22 '19 at 5:23

4

If you don't mind installing a package, this should be useful:
https://pypi.org/project/pathvalidate/

From https://pypi.org/project/pathvalidate/#sanitize-a-filename:

```
from pathvalidate import sanitize_filename

fname = "fi:l*e/p\"a?t>h|.t<xt"
print(f"{fname} -> {sanitize_filename(fname)}\n")
fname = "\0_a*b:c<d>e%f/(g)h+i_0.txt"
print(f"{fname} -> {sanitize_filename(fname)}\n")
```

Output

```
fi:l*e/p"a?t>h|.t<xt -> filepath.txt
_a*b:c<d>e%f/(g)h+i_0.txt -> _abcde%f(g)h+i_0.txt
```

Share Improve this answer Follow

answered Mar 25 '20 at 19:34
Stavros
43 8

```
> filename = "This is a väryì' Strange File-Nömé.jpeg"
> pattern = re.compile(r'[^-a-zA-Z0-9.]+')
> slugify(filename,regex_pattern=pattern)
'this-is-a-varyi-strange-file-nome.jpeg'
```

Note that the regex pattern was copied from the

`ALLOWED_CHARS_PATTERN_WITH_UPPERCASE`

global variable within the `slugify.py` file of the python-slugify package and extended with "."

Keep in mind that special characters like `.()` must be escaped with `\.`

If you want to preserve uppercase letters use the `lowercase=False` argument.

```
> filename = "This is a väryì' Strange File-Nömé.jpeg"
> pattern = re.compile(r'[^-a-zA-Z0-9.]+')
> slugify(filename,regex_pattern=pattern, lowercase=False)
'This-is-a-varyi-Strange-File-Nome.jpeg'
```

This worked using Python 3.8.4 and python-slugify 4.0.1

Share Improve this answer Follow

answered Mar 26 at 8:15

Alex

31 4

1

Most of these solutions don't work.

'/hello/world' -> 'helloworld'

'/helloworld'/ -> 'helloworld'

This isn't what you want generally, say you are saving the html for each link, you're going to overwrite the html for a different webpage.

I pickle a dict such as:

```
{'helloworld':
    (
    {'/hello/world': 'helloworld', '/helloworld/': 'helloworld1'},
    2)
```

Share Improve this answer Follow

answered May 16 '12 at 1:04

robert king
14.7k 8 88 108

note, if using helloworld1, you also need to check helloworld1 isn't in use and so on.. – robert king Nov 25 '13 at 21:21

1

Not exactly what OP was asking for but this is what I use because I need unique and reversible conversions:

```
# p3 code
def safePath (url):
    return ''.join(map(lambda ch: chr(ch) if ch in safePath.chars else '%%%02x'
% ch, url.encode('utf-8')))
safePath.chars = set(map(lambda x: ord(x),
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+-_ .'))
```

Result is "somewhat" readable, at least from a sysadmin point of view.

Share Improve this answer Follow

answered Sep 12 '14 at 12:19

makeroo
471 7 9

A wrapper for this with no spaces in files names: `def safe_filename(filename): return safePath(filename.strip().replace(' ','_'))` – SpeedCoder5 Jun 20 '18 at 17:23

0

I'm sure this isn't a great answer, since it modifies the string it's looping over, but it seems to work alright:

```
import string
for chr in your_string:
 if chr == ' ':
   your_string = your_string.replace(' ', '_')
 elif chr not in string.ascii_letters or chr not in string.digits:
    your_string = your_string.replace(chr, '')
```

Share Improve this answer Follow

answered May 5 '12 at 3:56

TankorSmash

UPDATE

0

All links broken beyond repair in this 6 year old answer.

Also, I also wouldn't do it this way anymore, just `base64` encode or drop unsafe chars. Python 3 example:

```
import re
t = re.compile("[a-zA-Z0-9.,_-]")
unsafe = "abcðéåß®Δ°´©¬ñ√ƒµ©ΔʃØ"
safe = [ch for ch in unsafe if t.match(ch)]
# => 'abc'
```

With `base64` you can encode and decode, so you can retrieve the original filename again.

But depending on the use case you might be better off generating a random filename and storing the metadata in separate file or DB.

```
from random import choice
from string import ascii_lowercase, ascii_uppercase, digits
allowed_chr = ascii_lowercase + ascii_uppercase + digits

safe = ''.join([choice(allowed_chr) for _ in range(16)])
# => 'CYQ4JDKE9JfcRzAZ'
```

ORIGINAL LINKROTTEN ANSWER:

The `bobcat` project contains a python module that does just this.

It's not completely robust, see this post and this reply.

So, as noted: `base64` encoding is probably a better idea if readability doesn't matter.

- Docs https://svn.origo.ethz.ch/bobcat/src-doc/safefilename-module.html
- Source https://svn.origo.ethz.ch/bobcat/trunk/src/bobcatlib/safefilename.py

Share Improve this answer Follow

edited Dec 28 '15 at 14:30

answered Jul 10 '09 at 10:19

wires
4,618 2 33 31

All links dead. Man, do something. – Shiva Dec 26 '15 at 10:18

converts txt to a valid Windows/*nix filename with printable characters

```
only.

    args:
        txt: The str to convert.
        chr_set: 'normal', 'universal', or 'inclusive'.
            'universal':    '
-.0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
            'normal':       Every printable character exept those disallowed on
Windows/*nix.
            'extended':     All 'normal' characters plus the extended character
ASCII codes 128-255
    """

    FILLER = '-'

    # Step 1: Remove excluded characters.
    if chr_set == 'universal':
        # Lookups in a set are O(n) vs O(n * x) for a str.
        printables = set('
-.0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz')
    else:
        if chr_set == 'normal':
            max_chr = 127
        elif chr_set == 'extended':
            max_chr = 256
        else:
            raise ValueError(f'The chr_set argument may be normal, extended or
universal; not {chr_set=}')
        EXCLUDED_CHRS = set(r'<>:"/\|?*')               # Illegal characters in
Windows filenames.
        EXCLUDED_CHRS.update(chr(127))                  # DEL (non-printable).
        printables = set(chr(x)
                         for x in range(32, max_chr)
                         if chr(x) not in EXCLUDED_CHRS)
    result = ''.join(x if x in printables else FILLER   # Allow printable
characters only.
                     for x in txt)

    # Step 2: Device names, '.', and '..' are invalid filenames in Windows.
    DEVICE_NAMES = 'CON,PRN,AUX,NUL,COM1,COM2,COM3,COM4,' \
                   'COM5,COM6,COM7,COM8,COM9,LPT1,LPT2,' \
                   'LPT3,LPT4,LPT5,LPT6,LPT7,LPT8,LPT9,' \
                   'CONIN$,CONOUT$,..,.'.split()        # This list is an O(n)
operation.
    if result in DEVICE_NAMES:
        result = f'-{result}-'

    # Step 3: Maximum length of filename is 255 bytes in Windows and Linux
(other *nix flavors may allow longer names).
    result = result[:255]

    # Step 4: Windows does not allow filenames to end with '.' or ' ' or begin
with ' '.
    result = re.sub(r'^[. ]', FILLER, result)
    result = re.sub(r' $', FILLER, result)
```


Share Improve this answer Follow edited Dec 18 '20 at 16:26 answered Dec 18 '20 at 16:18

ChaimG
4,898 4 27 41

0

Yet another answer for Windows specific paths, using simple replacement and no funky modules:

```
import re

def check_for_illegal_char(input_str):
    # remove illegal characters for Windows file names/paths
    # (illegal filenames are a superset (41) of the illegal path names (36))
    # this is according to windows blacklist obtained with Powershell
    # from: https://stackoverflow.com/questions/1976007/what-characters-are-
forbidden-in-windows-and-linux-directory-names/44750843#44750843
    #
    # PS> $enc = [system.Text.Encoding]::UTF8
    # PS> $FileNameInvalidChars = [System.IO.Path]::GetInvalidFileNameChars()
    # PS> $FileNameInvalidChars | foreach { $enc.GetBytes($_) } | Out-File -
FilePath InvalidFileCharCodes.txt

    illegal =
'\u0022\u003c\u003e\u007c\u0000\u0001\u0002\u0003\u0004\u0005\u0006\u0007\u0008'
 + \

'\u0009\u000a\u000b\u000c\u000d\u000e\u000f\u0010\u0011\u0012\u0013\u0014\u0015'
 + \

'\u0016\u0017\u0018\u0019\u001a\u001b\u001c\u001d\u001e\u001f\u003a\u002a\u003f\u0

    output_str, _ = re.subn('['+illegal+']','_', input_str)
    output_str = output_str.replace('\\','_')   # backslash cannot be handled
by regex
    output_str = output_str.replace('..','_')   # double dots are illegal too,
or at least a bad idea
    output_str = output_str[:-1] if output_str[-1] == '.' else output_str #
can't have end of line '.'

    if output_str != input_str:
        print(f"The name '{input_str}' had invalid characters, "
              f"name was modified to '{output_str}'")

    return output_str
```

When tested with `check_for_illegal_char('fas\u0003\u0004good\\..asd.')`, I get:

```
The name 'fas♥♦good\..asd.' had invalid characters, name was modified to
'fas__good__asd'
```