python: change sys.stdout print to custom print function

Ask Question

asked 5 years, 3 months ago

viewed 4,296 times

active 5 years, 3 months ago

4

4

Im trying to understand how to create a custom print function. (using python 2.7)

```
import sys
class CustomPrint():
    def __init__(self):
        self.old_stdout=sys.stdout #save stdout

    def write(self, text):
        sys.stdout = self.old_stdout #restore normal stdout and print
        print 'custom Print--->' + text
        sys.stdout= self # make stdout use CustomPrint on next 'print'
                         # this is the line that trigers the problem
                         # how to avoid this??


myPrint = CustomPrint()
sys.stdout = myPrint
print 'why you make 2 lines??...'
```

The code above prints this to console:

```
>>>
custom Print--->why you make 2 lines??...
custom Print--->

>>>
```

and i want to print only one line:

```
>>>
1custom Print--->why you make 2 lines??...
```

Linked

220 Redirect stdout to a file in Python?

14 Set a Read-Only Attribute in Python?

0 Overriden print function in python does not work inside execFIle call

Related

3534 Calling an external command in Python

4466 What are metaclasses in Python?

2571 What is the difference between @staticmethod and @classmethod in Python?

2157 Finding the index of an item given a list containing it in Python

2613 Difference between append vs. extend list methods in Python

4278 Does Python have a ternary conditional operator?

2465 Accessing the index in 'for' loops?

```
>>>
```

But cant figure out how to make this custom print work , i understand that there's some kind of recursion that triggers the second output to the console (i use self.write , to assign stdout to self.write himself !)

how can i make this work ? or is my approach just completely wrong...

python operator-overloading stdout

share improve this question

asked Feb 20 '13 at 17:45

andsoa

320 ● 5 ● 11

related: Redirect stdout to a file in Python? – jfs Jan 27 '15 at 11:27

add a comment

3 Answers

active oldest votes

2

It's not recursion. What happens is your `write` function is called twice, once with the text you expect, second time with just `'\n'`. Try this:

```
import sys
class CustomPrint():
    def __init__(self):
        self.old_stdout=sys.stdout

    def write(self, text):
        text = text.rstrip()
        if len(text) == 0: return
        self.old_stdout.write('custom Print--->' + text + '\n')
```

What I do in the above code is I add the new line character to the text passed in the first call, and make sure the second call made by the print statement, the one meant to print new line, doesn't print anything.

2357 How to make a chain of function decorators?

2044 Difference between __str__ and __repr__?

2741 Does Python have a string 'contains' substring method?

Hot Network Questions

- Where is the bloodiest square mile on Earth?
- Why does array[idx++]+="a" increase idx once in Java 8 but twice in Java 9 and 10?
- Decoding the Kaadi system
- How to prove that Z operator rotates points on Bloch sphere about Z axis through 180°?
- Time travel novel with dragons and an old truck
- Is there a German equivalent for "self defeating"?
- How can I play the devil's advocate in politics without being attacked?
- What is the logic behind forbidding LGBT conversion therapies?
- Notation for an interval when you don't know which bound is greater
- Is this solution to nonlinear first order ODE correct?
- Why is best subset selection not favored in comparison to lasso?
- Why aren't Americans simply called "Americans"?
- In a post-fossil fuel economy, what could natural gas pipelines be used for?
- Can the original WotC-published SRD RTF files be found anywhere?
- How can I make this logo look more obviously like an icing bag and crown?

Now try to comment out the first two lines and see what happens:

```
    def write(self, text):
        #text = text.rstrip()
        #if len(text) == 0: return
        self.old_stdout.write('custom Print--->' + text + '\n')
```

share improve this answer

edited Feb 20 '13 at 18:08

answered Feb 20 '13 at 18:00

piokuc

16.4k ● 6 ● 41 ● 73

Thanks , its working like it should :) didn't know that \n was added on a second pass ! sys.stderr.write(":".join("{0:b}".format(ord(c)) for c in text)) even added this line just to make sure and your right ^^ – andsoa Feb 20 '13 at 18:22

1 there's a small problem when using comma separated print: 'print var1, var2' Apparently the comma calls print for each 'var' and suppresses the addition of '\n' on the previous 'var' ! This can be solved by adding a tmp var on the customPrint class , that stores text , and only prints when text ends with '\n' – andsoa Feb 21 '13 at 13:24

@andsoa Ok, thanks for sharing this – piokuc Feb 21 '13 at 13:39

add a comment

3

One solution may be to use a context manager if it's localised.

```
#!/usr/bin/env python
from contextlib import contextmanager
################################################################################
@contextmanager
def no_stdout():
    import sys
    old_stdout = sys.stdout
```

```
    class CustomPrint():
        def __init__(self, stdout):
            self.old_stdout = stdout

        def write(self, text):
            if len(text.rstrip()):
                self.old_stdout.write('custom Print--->'+ text)

    sys.stdout = CustomPrint(old_stdout)

    try:
        yield
    finally:
        sys.stdout = old_stdout
################################################################################
print "BEFORE"
with no_stdout():
    print "WHY HELLO!\n"
    print "DING DONG!\n"

print "AFTER"
```

The above produces:

```
BEFORE
custom Print--->WHY HELLO!
custom Print--->DING DONG!
AFTER
```

The code would need tidying up esp. around what the class should do WRT setting stdout back to what it was.

share improve this answer

answered Feb 20 '13 at 18:12

sotapme

3,499 ● 2 ● 12 ● 17

thanks for sharing this elegant solution ! – andsoa Feb 21 '13 at 13:14

add a comment

1

How about doing `from __future__ import print_function`. This way you will use Python3 print function instead of print statement from Python2. Then you can redefine the print function:

```
def print(*args, **kwargs):
    __builtins__.print("Custom--->", *args, **kwargs)
```

There is a catch however, you will have to start using print function.

share improve this answer

answered Feb 20 '13 at 17:52

Fenikso

6,431 ● 1 ● 26 ● 56

That would work , but my case i need to change the print output of some classes written with python 2.7 print :/ – andsoa Feb 20 '13 at 18:04

add a comment

Your Answer

Sign up or log in

Sign up using Google

Sign up using Facebook

Sign up using Email and Password

Post as a guest

Name

Email

required, but never shown

Post Your Answer

By clicking "Post Your Answer", you acknowledge that you have read our updated terms of service, privacy policy and cookie policy, and that your continued use of the website is subject to these policies.

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