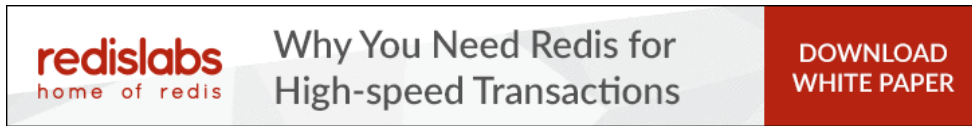


How to access (and edit) variables from a callback function?



I use Boto to access Amazon S3. And for file uploading I can assign a callback function. The problem is that I cannot access the needed variables from that callback function until I make them global. In another hand, if I make them global, they are global for all other Celery tasks, too (until I restart Celery), as the file uploading is executed from a Celery task.

Here is a function that uploads a JSON file with information about video conversion progress.

```
def upload_json():
    global current_frame
    global path_to_progress_file
    global bucket
    json_file = Key(bucket)
    json_file.key = path_to_progress_file
    json_file.set_contents_from_string('{"progress": "%s"}' % current_frame,
    cb=json_upload_callback, num_cb=2, policy="public-read")
```

And here are 2 callback functions for uploading frames generated by ffmpeg during the video conversion and a JSON file with the progress information.

```
# Callback functions that are called by get_contents_to_filename.
# The first argument is representing the number of bytes that have
# been successfully transmitted from S3 and the second is representing
# the total number of bytes that need to be transmitted.
def frame_upload_callback(transmitted, to_transmit):
    if transmitted == to_transmit:
        upload_json()
def json_upload_callback(transmitted, to_transmit):
    global uploading_frame
    if transmitted == to_transmit:
        print "Frame uploading finished"
        uploading_frame = False
```

Theoretically, I could pass the uploading_frame variable to the upload_json function, but it wouldn't get to json_upload_callback as it's executed by Boto.

In fact, I could write something like this.

```
In [1]: def make_function(message):
...:     def function():
...:         print message
...:         return function
...:
In [2]: hello_function = make_function("hello")
In [3]: hello_function
Out[3]: <function function at 0x19f4c08>
In [4]: hello_function()
hello
```

Which, however, doesn't let you edit the value from the function, just lets you read the value.

```
def myfunc():
    stuff = 17
    def lfun(arg):
        print "got arg", arg, "and stuff is", stuff
        return lfun

my_function = myfunc()
my_function("hello")
```

This works.

```
def myfunc():
    stuff = 17
    def lfun(arg):
        print "got arg", arg, "and stuff is", stuff
        stuff += 1
        return lfun

my_function = myfunc()
my_function("hello")
```

And this gives an UnboundLocalError: local variable 'stuff' referenced before assignment.

Thanks.

[python](#) [variables](#) [namespaces](#) [callback](#) [closures](#)

edited Jan 27 '11 at 11:12

asked Jan 27 '11 at 10:46

Your example doesn't show how `uploading_frame` is used, so it is hard to understand why you event want it. Creating a simpler example without any S3-related stuff that more clearly shows the problem would probably help too. – [dkagedal](#) Jan 27 '11 at 10:48

3 Answers

In Python 2.x closures are read-only. You can however use a closure over a mutable value...
i.e.

```
def myfunc():
    stuff = [17] # <----- this is a mutable object
    def lfun(arg):
        print "got arg", arg, "and stuff[0] is", stuff[0]
        stuff[0] += 1
    return lfun

my_function = myfunc()
my_function("hello")
my_function("hello")
```

If you are instead using Python 3.x the keyword `nonlocal` can be used to specify that a variable used in read/write in a closure is not a local but should be captured from the enclosing scope:

```
def myfunc():
    stuff = 17
    def lfun(arg):
        nonlocal stuff
        print "got arg", arg, "and stuff is", stuff
        stuff += 1
    return lfun

my_function = myfunc()
my_function("hello")
my_function("hello")
```

edited Apr 11 '13 at 10:01

answered Jan 27 '11 at 11:35




6502

76k 10 94 177


1 Thank you very much! That works! – [aruseni](#) Jan 27 '11 at 11:44

In Python 3 closures aren't read-only, but you do have to specify modified variables with 'nonlocal'. – [Fred Nurk](#) Jan 27 '11 at 12:16



Love remote work?

Find it on a new kind of career site


stackoverflow
JOBS

Get started

You could create a partial function via `functools.partial`. This is a way to call a function with some variables pre-baked into the call. However, to make that work you'd need to pass a mutable value - eg a list or dict - into the function, rather than just a bool.

```
from functools import partial
def callback(arg1, arg2, arg3):
    arg1[:] = [False]
    print arg1, arg2, arg3

local_var = [True]
partial_func = partial(callback, local_var)

partial_func(2, 1)
print local_var # prints [False]
```

answered Jan 27 '11 at 11:28



[Daniel Roseman](#)

369k 31 472 554

Thank you very much! That works! – [aruseni](#) Jan 27 '11 at 11:45

A simple way to do these things is to use a local function

```
def myfunc():
    stuff = 17
    def lfun(arg):
        print "got arg", arg, "and stuff is", stuff
        stuff += 1
    def register_callback(lfun)
```

This will create a new function every time you call myfunc, and it will be able to use the local "stuff" copy.

answered Jan 27 '11 at 10:53



[dkagedal](#)

447 2 7 12

Please see the update above. – [aruseni](#) Jan 27 '11 at 11:15

Thank you for the answer. – [aruseni](#) Jan 27 '11 at 11:45

1 [This does not work.](#) – [Fred Nurk](#) Jan 27 '11 at 12:18
