Reentrant Context Managers

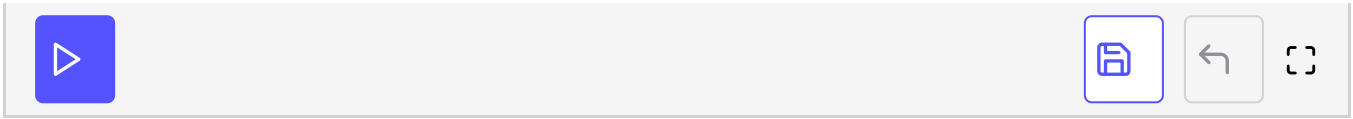
Most context managers that you create will be written such that they can only be used once using a **with** statement. Here's a simple example:

```
from contextlib import contextmanager
@contextmanager
def single():
    print('Yielding')
    yield
    print('Exiting context manager')
context = single()
with context:
    pass

#Yielding
#Exiting context manager

with context:
    pass

#Traceback (most recent call last):
#  File "/usercode/__ed_file.py", line 14, in <module>
# with context
#  File "/usr/local/lib/python3.5/contextlib.py", line 61, in __enter__
#    raise RuntimeError("generator didn't yield") from None
#RuntimeError: generator didn't yield
```



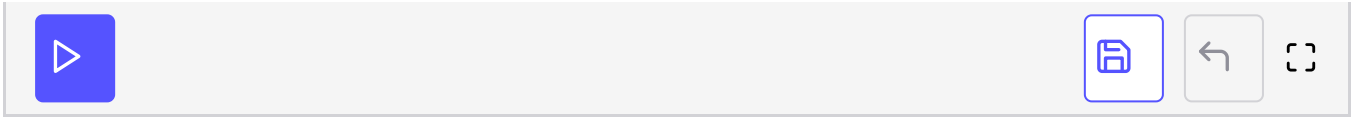
Here we create an instance of our context manager and try running it twice with Python's with statement. The second time it runs, it raises a **RuntimeError**.

But what if we wanted to be able to run the context manager twice? Well we'd need to use one that is "reentrant". Let's use the **redirect_stdout** context manager that we used before!

```
from contextlib import redirect_stdout
from io import StringIO
stream = StringIO()
```

```
write_to_stream = redirect_stdout(stream)
with write_to_stream:
    print('Write something to the stream')
    with write_to_stream:
        print('Write something else to stream')

print(stream.getvalue())
#Write something to the stream
#Write something else to stream
```



Here we create a nested context manager where they both write to **StringIO**, which is an in-memory text stream. The reason this works instead of raising a RuntimeError like before is that redirect_stdout is reentrant and allows us to call it twice. Of course, a real world example would be much more complex with more functions calling each other. Please also note that reentrant context managers are not necessarily thread-safe. Read the documentation before trying to use them in a thread.

Wrapping Up

Context managers are a lot of fun and come in handy all the time. I use them in my automated tests all the time for opening and closing dialogs, for example. Now you should be able to use some of Python's built-in tools to create your own context managers. Be sure to take the time to read the Python documentation on contextlib as there are lots of additional information that is not covered in this chapter. Have fun and happy coding!