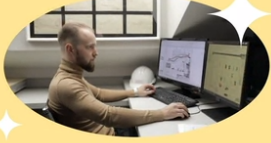


# Менеджеры контекста многократного использования

AMS<sup>3</sup> AUTODESK

Распродажа онлайн-курсов от «AMS<sup>3</sup>»



autocad-specialist.ru

Большая распродажа курсов от «AMS<sup>3</sup>» – Скидки до 60%!

%

Узнать больше

РЕКЛАМА

[Стандартная библиотека Python3.](#) / [Модуль contextlib в Python, создания менеджеров контекста](#)  
/ Менеджеры контекста многократного использования

[Одноразовые менеджеры контекста](#), а также [менеджеры контекста с повторным входом](#) поддерживают многократное использование, но они не будут работать или не будут работать правильно, если конкретный экземпляр менеджера контекста уже использовался в [операторе with](#).

Примером [многократно используемого менеджера контекста](#), но не менеджера с повторным входом (т. к. они также могут использоваться повторно), является [класс contextlib.ExitStack\(\)](#). Этот класс вызывает все зарегистрированные в настоящее время обратные вызовы, оставляя оператор with, независимо от того, где были добавлены эти обратные вызовы:

```
from contextlib import ExitStack

stack = ExitStack()

with stack:
    stack.callback(print, "Обратный вызов: из первого контекста")
    print("Покидаем первый контекст")

# Покидаем первый контекст
# Обратный вызов: из первого контекста

with stack:
    stack.callback(print, "Обратный вызов: из второго контекста")
    print("Покидаем второй контекст")

# Покидаем второй контекст
# Обратный вызов: из второго контекста

# пробуем использовать как менеджер
# контекста с повторным входом
with stack:
    stack.callback(print, "Обратный вызов: из внешнего контекста")
    with stack:
        stack.callback(print, "Обратный вызов: из внутреннего контекста")
        print("Покидаем внутренний контекст")
    print("Покидаем внешний контекст")

# Покидаем внутренний контекст
# Обратный вызов: из внутреннего контекста
# Обратный вызов: из внешнего контекста
# Покидаем внешний контекст
```

Как видно из выходных данных примера, повторное использование одного объекта stack в нескольких операторах работает правильно, но попытка вложения их приведет к очистке stack в конце самого внутреннего оператора with, что вряд ли будет желательным поведением.

Использование отдельных экземпляров ExitStack вместо повторного использования одного экземпляра позволяет избежать этой проблемы:

https://docs-python.ru/standart-library/modul-contextlib-python/mnogorazovye-kontekstnye-menedzhery/

1/3

```
from contextlib import ExitStack

with ExitStack() as outer_stack:
    outer_stack.callback(print, "Обратный вызов: из внешнего контекста")
    with ExitStack() as inner_stack:
        inner_stack.callback(print, "Обратный вызов: из внутреннего контекста")
        print("Покидаем внутренний контекст")
    print("Покидаем внешний контекст")

# Покидаем внутренний контекст
# Обратный вызов: из внутреннего контекста
# Покидаем внешний контекст
# Обратный вызов: из внешнего контекста
```

Содержание раздела:

- [ОБЗОРНАЯ СТРАНИЦА РАЗДЕЛА](#)
- [Ограничение обычных менеджеров контекста](#)
- [Контекстные менеджеры с повторным входом](#)
- [Многоразовые контекстные менеджеры](#)
- [Переменное количество контекстных менеджеров](#)
- [Менеджера контекста в качестве декоратора функции](#)
- [Замена try-finally менеджером контекста](#)
- [Перехват исключений из методов enter](#)
- [Очистка ресурсов при неудачном завершении with](#)
- [Декоратор @contextmanager модуля contextlib](#)
- [Декоратор @asynccontextmanager модуля contextlib](#)
- [Функция closing\(\) модуля contextlib](#)
- [Функция nullcontext\(\) модуля contextlib](#)
- [Функция suppress\(\) модуля contextlib](#)
- [Функции redirect std\\*\(\) модуля contextlib](#)
- [Класс ContextDecorator\(\) модуля contextlib](#)
- [Класс ExitStack\(\) модуля contextlib](#)
- [Класс AsyncExitStack\(\) модуля contextlib](#)
- [Функция aclosing\(\) модуля contextlib](#)
- [Функция AsyncContextDecorator\(\) модуля contextlib](#)
- [Контекстный менеджер chdir\(\) модуля contextlib](#)

ХОЧУ ПОМОЧЬ  
ПРОЕКТУ

