# **Обработка исключений. Блоки finally и else**

Курс по Python ООП: <https://stepik.org/a/116336>

[Смотреть материал на видео](https://www.youtube.com/watch?v=3DmzotEptvM&list=PLA0M1Bcd0w8zPwP7t-FgwONhZOHt9rz9E)

Мы продолжаем тему исключений. На прошлом занятии мы с вами познакомились с основами работы блоков try / except, а также увидели, как правильно использовать иерархию классов исключений для их корректной обработки. Однако простой конструкции try / except часто бывает недостаточно для корректной обработки возникающих ошибок. Еще на практике дополнительно прописывают необязательные блоки else и finally. Давайте посмотрим, как они работают и для чего нужны.

Итак, блок try поддерживает необязательный блок else, который выполняется при штатном выполнении кода внутри блока try, то есть, когда не произошло никаких ошибок. Например, его можно записать так:

**try**:     x, y = map(int, input().split())     res = x / y **except** ZeroDivisionError **as** z:     **print**(z)**except** ValueError **as** z:     **print**(z) **else**:     **print**("Исключений не произошло")

Теперь, при запуске программы, вводя корректные числа, мы увидим сообщение «Исключений не произошло». Если же возникает какое-либо исключение, то этот блок выполняться не будет.

Вторым необязательным блоком является блок finally, который, наоборот, выполняется всегда после блока try, вне зависимости произошла ошибка или нет:

**finally**:     **print**("Блок finally выполняется всегда")

Теперь, при запуске программы, мы будем видеть это сообщение вне зависимости от возникновения возможных ошибок. И здесь часто возникает вопрос: зачем нужен этот блок, если он выполняется всегда после try? Мы с таким же успехом можем записать этот print сразу после этого блока и, вроде бы, все будет работать также? В действительности, нет. Смотрите, если мы, например, уберем блок except с исключением ValueError, запустим программу и введем нечисловые значения, то, конечно, возникнет необработанное исключение, но при этом, блок finally все равно выполнился! Этого не произошло бы, если просто записать print после try.

Но все же, в каких ситуация нам может понадобиться использовать блок finally? Классический пример – это работа с файлами. Вначале, для открытия файла, нам следует прописать функцию open() в блоке try:

**try**:     f = open("myfile.txt")     f.write("hello")**except** FileNotFoundError **as** z:     **print**(z)**except**:     **print**("Другая ошибка")

Здесь ошибка произойдет после открытия файла в строчке f.write(), поэтому файл остается открытым. Это нехорошо. Мы знаем, что любой файл нужно закрывать даже при возникновении ошибок. Как раз здесь может пригодиться блок finally:

**try**:     f = open("myfile.txt")     f.write("hello")**except** FileNotFoundError **as** z:     **print**(z)**except**:     **print**("Другая ошибка")**finally**:     **if** f:         f.close()         **print**("Файл закрыт")

Запустим программу и увидим, что файл закрывается (даже если убрать второй блок except).

Конечно, конкретно при работе с файлами такую громоздкую конструкцию заменяют более компактной с использованием файлового менеджера:

**try**:     **with** open("myfile.txt") **as** f:         f.write("hello")**except** FileNotFoundError **as** z:     **print**(z)**except**:     **print**("Другая ошибка")

Который автоматически закрывает файл. Я привел первый вариант исключительно в учебных целях, чтобы вы увидели для чего в принципе можно использовать блок finally.

Следующая особенность работы блока finally связана с обработкой исключений внутри функций. Предположим, в функции вводятся два целых числа и возвращаются в виде кортежа:

**def** get\_values():     **try**:         x, y = map(int, input().split())         **return** x, y     **except** ValueError **as** v:         **print**(v)         **return** 0, 0     **finally**:         **print**("finally выполняется до return")     x, y = get\_values()**print**(x, y)

Здесь может возникнуть исключение ValueError, если были введены не целые числа. В этом случае возвращаются нули. Также прописан блок finally с выводом сообщения «finally выполняется до return», чтобы мы убедились, что этот блок действительно выполняется до оператора return.

Запускаем программу, вводим два значения и вне зависимости от ошибок срабатывает блок finally до оператора return. Вот этот момент также нужно знать при реализации этого блока.

## **Вложенные блоки try / except**

Наконец, блоки try / except можно вкладывать один в другой, например, так:

**try**:     x, y = map(int, input().split())     **try**:         res = x / y     **except** ZeroDivisionError:         **print**("Деление на ноль")**except** ValueError **as** z:     **print**("Ошибка ValueError")

Или, можно внутренний блок try вынести в функцию:

**def** div(a, b):     **try**:         **return** x / y     **except** ZeroDivisionError:         **return** "Деление на ноль"

А, затем, вызвать в первом блоке try:

**try**:     x, y = map(int, input().split())     res = div(x, y)**except** ValueError **as** z:     **print**("Ошибка ValueError")     **print**(res)

Все эти комбинации, конечно же, допустимы и используются на практике. На следующем занятии мы продолжим эту тему и детальнее рассмотрим механизм возникновения и обработки исключений.

Курс по Python ООП: <https://stepik.org/a/116336>