# **Распространение исключений (propagation exceptions)**

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

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

На предыдущих занятиях мы с вами в целом рассмотрели работу блоков try / except / finally / else на примере относительно простых программ. В реальности, программы куда сложнее, содержат вызовы различных функций и даже могут использовать многопоточную реализацию. Давайте посмотрим, как будут выглядеть сообщения об ошибках при использовании функций.

Для этого объявим вначале функцию:

**def** func1():     **return** 1/0

и вызовем ее также после третьего print():

**print**("Я к вам пишу – чего же боле?")**print**("Что я могу еще сказать?")**print**("Теперь, я знаю, в вашей воле") func1()**print**("Меня презреньем наказать.")**print**("Но вы, к моей несчастной доле")**print**("Хоть каплю жалости храня,")**print**("Вы не оставите меня.")

При попытке выполнить эту программу, Python выдаст следующие строчки:

Traceback (most recent call last):  
  File "D:/Python/Projects/p\_course/ex1.py", line 7, in <module>  
    func1()  
  File "D:/Python/Projects/p\_course/ex1.py", line 2, in func1  
    return 1/0  
ZeroDivisionError: division by zero

Что примечательного в этом сообщении? Мы здесь видим указание об ошибке типа ZeroDivisionError для строчки 2 и строчки 7. В строчке 7 происходит вызов функции, которая выдает исключение, а в строчке 2 записано непосредственное деление на ноль. Почему строчки отображаются именно в таком порядке? Здесь нам нужно вспомнить, что при вызове функций формируется стек их вызова. И в нашем случае, этот стек можно представить, следующим образом:

Далее, возникшее исключение на уровне func1, последовательно распространяется по всему стеку вызова, доходя до верхнего уровня main. Именно этот стек распространения исключения и выдает интерпретатор языка Python.

Если, к примеру, добавить в программу еще один вложенный вызов функции:

**def** func2():     **return** 1/0   **def** func1():     **return** func2()

То в консоли появятся уже три строчки об ошибке, так как стек вызова стал длиннее:

Traceback (most recent call last):  
  File "D:/Python/Projects/p\_course/ex1.py", line 10, in <module>  
    func1()  
  File "D:/Python/Projects/p\_course/ex1.py", line 5, in func1  
    return func2()  
  File "D:/Python/Projects/p\_course/ex1.py", line 2, in func2  
    return 1/0  
ZeroDivisionError: division by zero

Все это пример того, как исключение, зародившееся на одном из уровней стека вызова, постепенно поднимается на самый верх. Это называется распространением исключений. По-английски:

propagation exceptions

Причем, обработать (перехватывать) исключение можно на любом уровне этого стека. Например, сделаем это на самом верхнем. Поместим вызов функции в блок try/except:

**try**:     func1()**except**:     **print**("Error for func1")

Теперь, все исключения для func1, которые дойдут до верхнего уровня, будут обработаны и программа выполнится в полном объеме.

Но, мы также можем обрабатывать исключения и на более глубоких уровнях, например, непосредственно в функции func2() при выполнении деления на ноль:

**def** func2():     **try**:         **return** 1/0     **except**:         **return** "-- деление на ноль --"

А на вершине стека, в глобальной области, просто вызовем функцию func1:

**print**("Я к вам пишу – чего же боле?")**print**("Что я могу еще сказать?")**print**("Теперь, я знаю, в вашей воле")**print**(func1())**print**("Меня презреньем наказать.")**print**("Но вы, к моей несчастной доле")**print**("Хоть каплю жалости храня,")**print**("Вы не оставите меня.")

При выполнении программы также не будет возникать никаких ошибок, так как деление на ноль обрабатывается непосредственно в функции func2 и исключение уже не всплывает на более высокие уровни.

То есть, мы можем обрабатывать исключения на разных уровнях стека вызова, что очень удобно. Этот механизм обработки исключений позволяет программистам писать независимый, модульный, красивый код. В критических функциях достаточно генерировать исключения, а их обработку выполнять на другом, более глобальном уровне. Например, создается класс для печати данных на принтере. Тогда все ошибки, связанные с принтером (нет бумаги, нет подключения, не тот режим печати и т.п.) можно обрабатывать единым образом на верхнем, глобальном уровне. А нижние уровни только сигнализируют о проблемах и не более того. В результате, получается разделение ролей: функции нижних уровней сосредоточены исключительно на обработке данных, а функции верхних – на формировании сервисной информации для пользователя.

Вот так, благодаря механизму распространения и обработки исключений можно создавать гибкий и безопасный программный код.

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