К настоящему времени ваш код стал синтаксически правильным и красивым, он выполняется без проблем. Большой! Но подождите, когда вы продолжите писать программу, в сообщениях от Python появится что-то еще! И программа выполняется только частично. Что здесь происходит? Давайте в этом разберемся.

Некоторые ошибки в вашем коде не помешают запуску программы. Он выйдет из строя только при попытке выполнить «разорванную» строку: строку с ошибкой, называемую **exception**. Таким образом, исключения - это ошибки, обнаруженные во время выполнения (интерпретации) программы, тогда как синтаксические ошибки - это ошибки, обнаруженные во время компиляции программы в байт-код.

Давайте рассмотрим следующий фрагмент кода:

print("I will be the first")  
print("And I will be the second")  
a = 0  
b = 5  
print("Here comes an error!..")  
print(b / a)  
print("I won't be printed")

Действительно, мы можем видеть, что некоторые из отпечатков сработали, а другие (print (), который содержит ошибку, и еще один после нее) нет. Таким образом, весь код до исключения выполняется правильно, а все после - нет.

I will be the first  
and I will be the second  
Here comes an error!..  
Traceback (most recent call last):  
  File  "PATH/TO/YOUR/SCRIPT.py", line 6, in <module>  
    print(b / a)  
ZeroDivisionError: division by zero

Вы уже знаете, что самая важная и информативная часть ошибки в Python - это последняя строка. Он содержит четкое и четкое описание ошибки, с которой столкнулся Python. В этом случае мы видим **ZeroDivisionError**, что довольно информативно, не так ли? На ноль делить нельзя.

Подобно синтаксическим ошибкам, почти все встроенные исключения Python имеют **associated value**, которое указывает подробную причину ошибки. Исключения не являются безоговорочно фатальными: позже вы узнаете, как с ними обращаться.

**Most common exceptions for learners**

Пожалуй, наиболее распространенными исключениями, которые люди видят, пока они все еще изучают Python, являются **NameError**, **TypeError** и **ValueError**.

**NameError** обычно возникает, если вы не определили переменную перед ее использованием или сделали это неправильно.

Помните, что переменные в Python чувствительны к регистру, и их необходимо определить перед использованием.

Рассмотрим этот фрагмент кода:

print(a + b)  
a = 0  
b = 5

Это вызовет следующее исключение:

NameError: name 'a' is not defined

Связанное значение иногда сообщает вам точную проблему, поэтому ее будет очень легко исправить.

Ошибка **TypeError** возникает, когда операция или функция применяется к объекту несоответствующего типа. Связанное значение представляет собой строку, содержащую подробную информацию о несоответствии конкретного типа. Обычно это происходит, когда вы пытаетесь выполнить арифметические вычисления с несколькими неподдерживаемыми типами операндов:

print("15" + 2)

Здесь мы пытаемся суммировать строку и целое число, что снова вызовет исключение:

TypeError: unsupported operand type(s) for +: 'int' and 'str'

**ValueError** может возникнуть, если вы пытаетесь использовать переменную правильного типа, но с неподходящим значением. Например, это тот случай, когда вы пытаетесь создать целое число из строки, которая не имеет целочисленного значения:

print(int("five"))  
  
ValueError: invalid literal for int() with base 10: 'five'

Если у вас возникли проблемы с попыткой понять, в чем заключается ошибка, вы всегда можете скопировать и вставить последнюю строку трассировки и погуглить. Более того, вам настоятельно рекомендуется это сделать, так как 99% проблем, с которыми сталкиваются учащиеся, уже решены на специализированных форумах.