Представьте себе простой калькулятор, который может делить только числа, где мы просим пользователя ввести два числа, а затем распечатать результат.

number\_one = int(input("Please, enter the first number: "))  
number\_two = int(input("Please, enter the second number: "))  
result = number\_one / number\_two  
print("The result of your division is: ", result)

Let's run it:

>>> Please, enter the first number: 5  
>>> Please, enter the second number:  2  
>>> The result of your division is: 2.5

Once again:

>>> Please, enter the first number: 2  
>>> Please, enter the second number: 1  
>>> The result of your division is:  2.0

Now let's try the impossible:

>>> Please, enter the first number: 5  
>>> Please, enter the second number: 0  
Traceback (most recent call last):  
File "<input>", line 1, in <module>  
File "C:/Users/User/.PyCharm2018.2/config/scratches/scratch\_9.py", line 3, in <module>  
    result = number\_one / number\_two  
ZeroDivisionError: division by zero

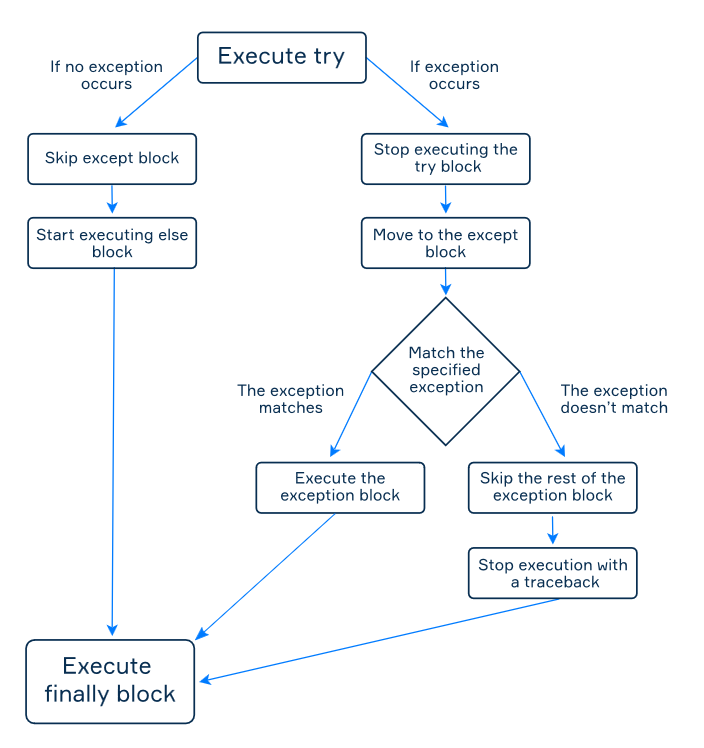
Мы снова сталкиваемся с этой надоедливой трассировкой, и наша программа полностью вылетает! Для предотвращения этого нам нужно использовать **try-except statements** в месте, которое может быть источником ошибок. Вот место, где мы делаем деление - переменный **result**, поэтому давайте закрепим его блоками try-except:

number\_one = int(input("Please, enter the first number: "))  
number\_two = int(input("Please, enter the second number: "))  
try:  
    result = number\_one / number\_two  
except ZeroDivisionError:  
    print("We achieve it thanks to except \*\*\*You can not divide by zero!!")  
else:  
    print("The result of your division is: ", result)  
finally:  
    print("It is done through finally \*\*\*Thanks for using our calculator! Come again!")

**Exception handling keywords**

Here you can see not only **try** and **except** keywords but also **else** and **finally**. The full exception handling block works as follows:

Сначала Python выполняет блок **try**: все, что находится между **try** и **except**.  
Если **no** **exception**, блок **try** успешно выполняется и завершается.  
Если возникает исключение, остальная часть блока **try** пропускается. После этого Python проверяет, соответствует ли тип исключения исключению, указанному после ключевого слова **except**, выполняет блок **except** и продолжает выполнение программы после блока **try**-**except**.  
Если исключение не соответствует исключению, указанному в предложении **except**, оно называется необработанным исключением (**unhandled exception)**, и выполнение вашей программы останавливается с помощью трассировки (**traceback**).  
Блок **else** в этой ситуации выполняется только в том случае, если не было никаких исключений.  
Также может быть ключевое слово **finally**. Предложение **finally** всегда выполняется перед выходом из блока **try-except**, независимо от того, произошло ли исключение или нет.



So let’s try executing our program now!

>>> Please, enter the first number: 5  
>>> Please, enter the second number: 0  
>>> We achieve it thanks to except \*\*\*You can not divide by zero!!  
>>> It is done through finally \*\*\*Thanks for using our calculator! Come again!

Видеть? Теперь наша программа будет работать, даже если пользователь ошибется и захочет разделить на ноль.

**Handling several exceptions**

Но что, если наш пользователь не понимает, что такое «число», и вводит, например, «один»?

>>> Please, enter the first number: 5  
>>> Please, enter the second number: one  
Traceback (most recent call last):  
File "<input>", line 1, in <module>  
File "C:/Users/User/.PyCharm2018.2/config/scratches/scratch\_9.py", line 2, in <module>  
    number\_two = int(input("Please, enter the second number: "))  
ValueError: invalid literal for int() with base 10: 'one'

Oh, Lord! We see those red disappointing lines again! Why? Well, because we specified only a ZeroDivisionError exception in our try-except block. And here we have a ValueError exception, so Python doesn't know how to deal with it in our program.

As you know, the built-in exceptions comprise **a hierarchical structure**, so you can do the following and identify no specific exception:

except:  
    print("An error occurred! Try again.")

Thus you'll catch any exception from the list. But it will also work for KeyboardInterrupt and other useful exceptions and moreover, it's considered to be a bad tone in programming, so it'd be better to use **two or more** except blocks for different exceptions:

except ZeroDivisionError:  
    print("We achieve it thanks to except \*\*\*You can't divide by zero!!")  
except ValueError:  
    print("You can only enter numbers consisting of digits, not text!!")

An except clause also may specify multiple exceptions as a **parenthesized tuple**, for example:

except (ValueError, TypeError):  
    print("You can only enter numbers consisting of digits, not text!!")

Due to the hierarchical structure, one exception can actually catch multiple exceptions. For example:

except ArithmeticError:  
    print("I will also catch FloatingPointError, OverflowError, and ZeroDivisionError")

Иногда может возникнуть ситуация, в которой вы даже не можете предсказать тип исключения в вашем коде. У вас нет другого выбора, кроме как использовать самое общее исключение. Для этой цели вместо использования чистой инструкции except:, например:

except:  
    # do something

you should use except Exception:

except Exception:  
    # do something

except Exception содержит все исключения Python, кроме следующих трех: GeneratorExit, KeyboardInterrupt, SystemExit**.** Таким образом, если вы используете эту структуру, вы все равно сможете завершить свою программу с помощью клавиатуры или команд, которые вызывают SystemExit.

**Summary**

* To deal with exceptions without terminating your program, Python has the **try-except block.**
* There are two more blocks to expand the possibilities to change the behavior of a program: else, which will be executed only if there are no exceptions in try-block, and finally, which will be executed at the end of try-except block whether the exception happened or not.
* All the exceptions comprise a hierarchical structure, i.e. some exceptions also include other exceptions.
* If you want to catch all possible exceptions, you should use the except **Exception** construction.
* Using those means wisely, you can write sustainable and effective code to prevent users’ mistakes and to keep your program running even under unexpected circumstances.