Theory: Exception handling

19 minutes 7 / 9 problems solved

Imagine a simple calculator, which can only divide numbers, where we ask a user for input of two numbers and then print the result.

```
while True:
   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 4, in <module>
    result = number_one / number_two
ZeroDivisionError: division by zero
```

Again we encounter this annoying traceback which crashes our program completely! For preventing this, we need to use **try-except statements** in the place, which can be a possible source of errors. Here is the place where we make the division — the variable `result`, so let's brace it with try-except blocks:

```
 while True:
    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!")
```

§1. 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:

- First, Python executes the `try` block: everything between `try` and `except`.
- If there is **no exception**, the try block is successfully executed and finished.

Current topic:

✓ Exception handling 3★

Topic depends on:

✓ While loop 11★

✓ Exceptions 3★ Stage 1

✗ Built-in exceptions

Topic is required for:

Assert statement

Testing user input

Unit testing in Python

How to read a traceback

User-defined exceptions

Queries and filters

Table of contents:

↑ Exception handling

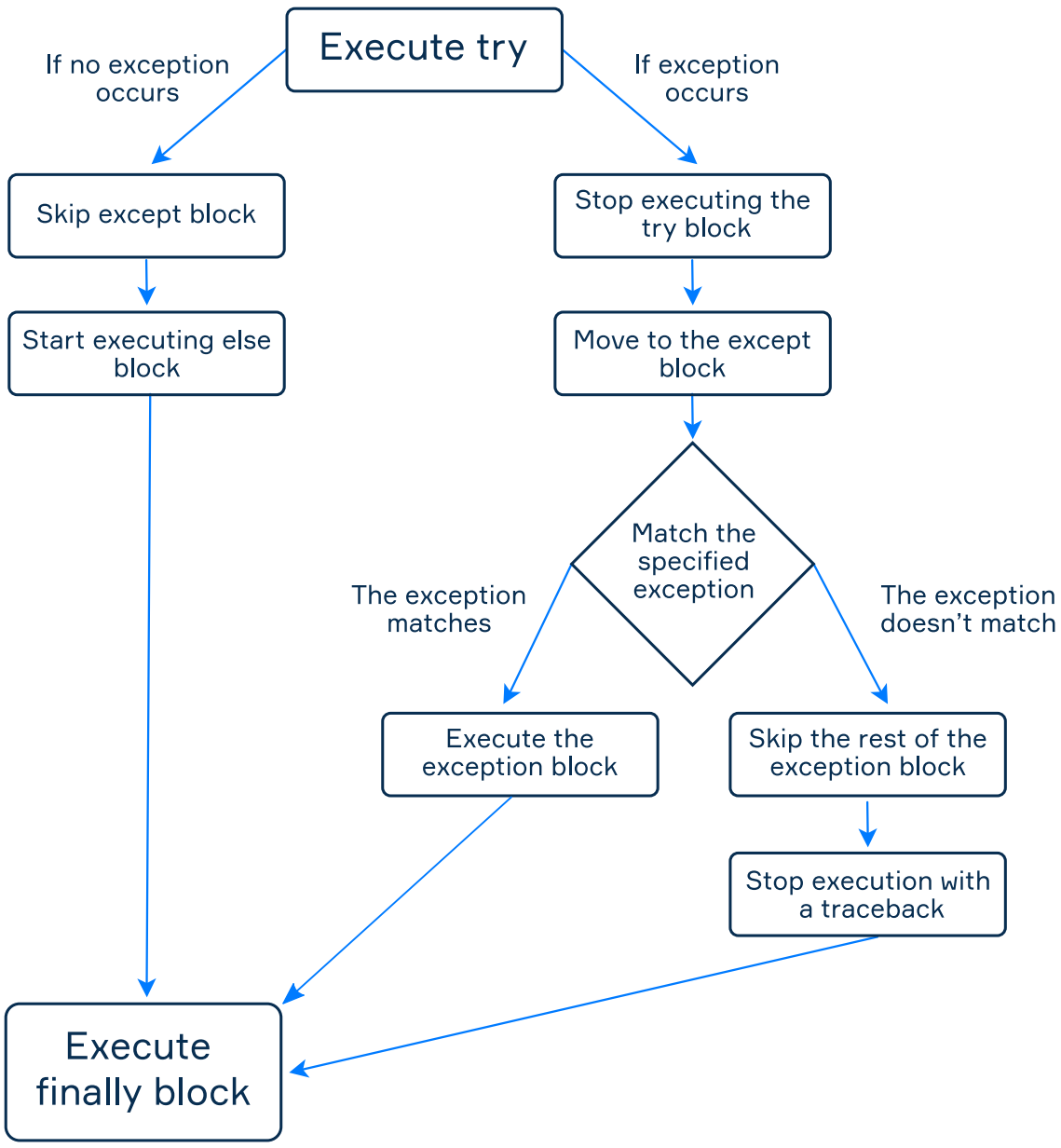
§1. Exception handling keywords

§2. Handling several exceptions

§3. Conclusion

Feedback & Comments

- If an exception occurs, the rest of the try block **is skipped**. After that, Python checks if the type of exception matches the exception specified after the `except` keyword, it executes the except block and continues executing the program **after** the try-except block.
- If an exception doesn't match the exception named in the except clause, it is called an *unhandled exception* and execution of your program stops with a **traceback**.
- The **else-block** in this situation is only executed if there occurred **no exceptions**.
- There can also be a `finally` keyword. A `finally` clause is **always** executed before leaving the try-except block, whether an exception has occurred or not.



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!
```

See? Now our program will be working even if the user makes a mistake and wants to divide by zero.

§2. Handling several exceptions

But what if our user doesn't understand what the "number" is and enters, for example, "one"?

```
>>> 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 3, in <mod
ule>
    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 ZeroDivisionEr
ror")
```

Sometimes there can be a situation in which you can't even predict the type of exception in your code. You have no other choice, but to use the most general exception. For that purpose, instead of using pure `except:` statement, like this:

```
except:
    # do something
```

you should use `except Exception`:

```
except Exception:
    # do something
```

`except Exception` contains all Python exceptions but the following three: `GeneratorExit`, `KeyboardInterrupt`, `SystemExit`. So if you use this structure, you will still be able to finish your program by the means of keyboard or commands, which cause `SystemExit`.

§3. Conclusion

- 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.

Report a typo

210 users liked this theory. **5** didn't like it. **What about you?**

Start practicing

This content was created over 1 year ago and updated about 21 hours ago. Share your feedback below in comments to help us improve it!

Comments (9) Hints (1) Useful links (2) Show discussion