Theory: Exceptions

9 minutes 6 / 9 problems solved

Start practicing

3009 users solved this topic. Latest completion was 5 minutes ago.

By now, your code has become syntactically correct and beautiful, it is being executed without any troubles. Great! But wait, when you continue writing the program, something else appears in the messages from Python! And the program is only partially executed. What is going on here? Let's figure this out.

Some errors in your code will not prevent the program from running. It will only crash while trying to execute a “broken” line: a line with an error called an **exception**. Thus, exceptions are the errors detected during the program **execution** (interpretation), whereas syntax errors are those detected during **compiling** the program into byte-code.

Let's examine the following piece of code:

```
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")
```

Indeed, we can see that some of the **print**s worked, while others (`print()` which contains the error, and yet another one after it) didn't. Thus, all the code **before** the exception is executed properly, and everything **after** is not.

```
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
```

You already know that the most important and informative part of an error in Python is the last line. It contains a clear and distinct description of the error Python has encountered. In this case, we can see **ZeroDivisionError**, which is quite informative, right? One can't divide by zero.

Similar to syntax errors, nearly all the built-in Python exceptions have an **associated value** that indicates the detailed cause of the error. Exceptions are not unconditionally fatal: later you will learn how to handle them.

§1. Most common exceptions for learners

Perhaps, the most common exceptions people see while they are still learning Python are `NameError`, `TypeError` and `ValueError`.

`NameError` is usually raised when you haven't defined your variable before using it or you have done it incorrectly.

Remember that variables are **case-sensitive** in Python and they've got to be defined **before usage**.

Consider this piece of code:

```
print(a + b)
a = 0
b = 5
```

It will cause the following exception:

```
NameError: name 'a' is not defined
```

Current topic:

✓ Exceptions Stage 1 3★ ···

Topic depends on:

✓ Variables Stage 1 17★ ···

✓ Errors Stage 1 3★ ···

Topic is required for:

Float special values ···

Built-in exceptions ···

✓ Exception handling 3★ ···

Processing requests Stage 1 ···

✓ Hashable ···

Table of contents:

The associated value sometimes tells you the exact problem, so it's going to be really easy to fix it.

`TypeError` is raised when an operation or function is applied to an object of **inappropriate type**. The associated value is a string giving details about the particular type mismatch. A common case for it is when you're trying to perform arithmetic calculations on several unsupported operand types:

```
print("15" + 2)
```

Here we're trying to sum up a string and an integer, which will cause an exception again:

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

`ValueError` can be raised if you're trying to use a variable of the correct type, but with inappropriate **value**. For example, it's the case when you are trying to make an integer of a string, which has no integer value:

```
print(int("five"))

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

If you have any troubles trying to understand what the error is, you can always copy-paste the last line of a traceback and google it. Moreover, you're strongly encouraged to do so, as 99% of troubles that the learners face have already been solved on specialized forums.

§2. Conclusion

We've learned the exception basics, so now you are familiar with this concept and some of the most common exceptions in Python. Here's a recap:

- A program will not be compiled and executed if there are **syntax errors** in it.
- On the other hand, **exceptions** don't prevent a program from being compiled and run, but as soon as the line with an exception is being executed, the program crashes.
- There are certain tools to handle exceptions and even to raise them on your own, and soon you will learn more about it.

Report a typo

217 users liked this theory. 1 didn't like it. **What about you?**

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion