Theory: How to read a traceback

20 minutes 0 / 5 problems solved

"Anyone who has never made a mistake has never tried anything new." — Albert Einstein.

A programmer, like any other person, should be very careful to avoid mistakes. In real life, you may not understand exactly where you did something wrong, but in programming, there is a **traceback** that will point you to a mistake! It is very important to learn to read it. In this topic, we will not dwell on specific exceptions but will try to convince you instead that traceback is very important, and you should try to get all the necessary information from it.

§1. Traceback insides

You've already learned to read a simple traceback and know that the last line is usually the most useful. Sometimes a traceback can contain several lines of code, you need to understand how to find the error. It happens a lot with functions, so we'll show you this as an example. Let's write a function `is_positive()` that prints whether the given number is positive or negative:

```
# this code is in the file 'numbers.py'
def is_positive(number):
    if number > 0:
        return number + " is positive"
    else:
        return number + " is negative"


print(is_positive(1))
```

Can you spot the error? If executed, it will throw the following `TypeError`:

```
Traceback (most recent call last):
  File "/full/path/to/numbers.py", line 9, in <module>
    print(is_positive(1))
  File "/full/path/to/numbers.py", line 4, in is_positive
    return number + " is positive"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Let's go up starting from the last line and divide the traceback into parts:

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

  The last line of the traceback always contains the name of the exception that was raised and the reason why it might have happened.

- ```
  File "/full/path/to/numbers.py", line 4, in is_positive
      return number + " is positive"
  ```

  Then there's the line of code that triggered the exception. This part of traceback contains the full path to the `numbers.py` file, the line number where the error occurred, and the name of the function where this line is located.

- ```
  File "/full/path/to/numbers.py", line 9, in <module>
      is_positive(1)
  ```

  This block is similar to the previous one, but it refers to the line of code that called the function with the broken code. The location of the function call is also given: the module name and the line inside the file. The `<module>` means that an error occurred in the executable file.

As you might have guessed, the `number` needs to be turned into a string value using the `str()` function to make this code valid.

Current topic:

How to read a traceback

Topic depends on:

✓ Declaring a function Stage 1 10★

✓ Else statement Stage 1 15★

✓ For loop Stage 1 12★

✓ Create module Stage 1

✓ Exception handling 3★

Table of contents:

↑ How to read a traceback

§1. Traceback insides

§2. Find a way

§3. "Long" problems

§4. Recap

Feedback & Comments
  ```

§2. Find a way

Traceback can be extra useful with other imported modules. Let's create a new file `import_numbers.py`, import and then run the very same `is_positive()` function from our previous module, `numbers.py`:

```
from numbers import is_positive

is_positive(1)
```

We are going to get the same `TypeError` again:

```
Traceback (most recent call last):
  File "/full/path/to/import_numbers.py", line 3, in <module>
    is_positive(1)
  File "/full/path/to/numbers.py", line 3, in is_positive
    return number + " is positive"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

This traceback is very similar to the previous one, but this time, the path to the module with the error will be changed `"/full/path/to/numbers.py")` as well as the module with the function (`"/full/path/to/import_numbers.py")`. The main thing, though, remains — thanks to the traceback, we can find out the exact line of the exact module where the error occurred.

Now we understand that our traceback consists of several blocks. In our example, we got the following bottom-up structure:
1. The name of the error and its description.
2. The location of the module that contains the "broken" function code line and the line itself.
3. The location of the executable file and the executed function.

In practice, a traceback can contain a good number of blocks. It simply means that many functions were called until the exception was thrown.

§3. "Long" problems

Let's add another function to our code; `check_numbers()` accepts a list of elements and prints the result for each number. The user may want to specify a number in a string format, such as `"2"`, so we use the *try-except statement* to handle this situation. Otherwise, we would have had problems in the line "`if number > 0:`", because we can not compare `str` and `int` values. Take a look at the code below:

```
# numbers.py
def is_positive(number):
    if number > 0:
        return str(number) + " is positive"
    else:
        return str(number) + " is negative"


def check_numbers(numbers):

    for num in numbers:

        try:

            print(is_positive(num))

        except TypeError:

            if int(num) > 0:

                print(str(num) + " is positive")

            else:

                print(str(num) + " is negative")


# the call works correctly

check_numbers([1,-1,"2"])


# 1 is positive

# -1 is negative

# 2 is positive
```

Now, what if a user wants to specify a number in words? Like this:

```
check_numbers([1,-1,"Two"])
```

Then we get something that is not quite legible:

```
1 is positive
-1 is negative
Traceback (most recent call last):
  File "/full/path/to/numbers.py", line 12, in check_numbers
    print(is_positive(num))
  File "/full/path/to/numbers.py", line 3, in is_positive
    if number > 0:
TypeError: '>' not supported between instances of 'str' and 'int'


During handling of the above exception, another exception occurred:


Traceback (most recent call last):

  File "/full/path/to/numbers.py", line 20, in <module>

    check_numbers([1,-1,"Two"])

  File "/full/path/to/numbers.py", line 14, in check_numbers

    if int(num) > 0:

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

What happened? In fact, this traceback clearly tells us that `During handling of the above exception, another exception occurred`. The first part of the traceback above these words tells us about the `TypeError` exception, which we tried to catch with the `try` - `except` statement. The thing is, during the execution of

this block, another exception was raised, the one that we see in the second part of the traceback. A new `ValueError` exception occurred in the `except TypeError:` block because we tried to pass a string `"Two"` to the function `int()`. That is, when handling the first `TypeError` exception, we received the second `ValueError` exception.

As you can see, tracebacks can contain a lot of information. It is important not to get lost in it and be able to find errors in your code, as well as to be ready to go back to our past mistakes and correct them.

§4. Recap

Let's not forget the "through hardships to the stars" mantra, or, in our case, through a traceback to the working code:

1. It is important to be able to read the traceback and locate your errors.
2. The traceback is divided into "blocks" that contain information about the error and its location.
3. It is more useful to read the "blocks" from the bottom up.
4. If several exceptions occur, the traceback will show all of them, from the earlier to the more recent errors, so that the last error is shown in the last line.